UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Departamento de Ingeniería Electrónica

Tesis Doctoral

Estudio, Modelado e Implementación Paralela de Sistemas Celulares Utilizados en Microfabricación

Presentada por:
D. Néstor Ferrando Jódar
Dirigida por:
Dr. D. Joaquín Cerdá Boluda
Dr. D. Rafael Gadea Gironés

Valencia, Abril 2011.

Help yourself by helping others.

Agradecimientos

Siempre he he creído que el estudiar una carrera en la universidad servía en buena parte para adquirir una nueva forma de pensar y de afrontar los problemas, técnicos, científicos o de otra índole. De la misma manera, pienso que haber realizado esta tesis me ha llevado, aparte de adquirir conocimientos y poder profundizar científicamente en un determinado campo, a comprender una serie de valores tales como la independencia, la plena responsabilidad sobre mis actos así como la paciencia y la fuerza espiritual necesaria para emprender un proyecto a largo plazo como es la realización de un doctorado.

De algo de lo que sí estoy seguro es que esta labor no podría haberla hecho por mi cuenta, muchos tipos de apoyo se me han brindado a lo largo de estos años: Científico, moral y económico, todos ellos importantes sin los cuales no hubiera podido concluir un trabajo del que sinceramente me siento orgulloso del resultado.

En primer lugar debo agradecer su apoyo a Joaquín Cerdá y Rafael Gadea, mis directores y mentores, los cuales han confiado en mí desde el primer día que entré en el grupo de investigación. Ellos me han tratado como una persona adulta y madura, me han dado total libertad en mi periplo investigador: me han permitido ser lo que he querido ser y me han dejado hacer lo que he querido hacer. Gracias por ofrecerme apoyo siempre que se lo he pedido y ayudarme en aquellas cosas en las que solo no hubiera sido capaz de triunfar.

Muy especialmente tengo que darle las gracias a Miguel Angel Gosálvez por su inmensa ayuda y compresión así como por haberme permitido el placer de colaborar con él, algo que he hecho con extrema ilusión e interés durante estos últimos dos años, no sólo por los resultados obtenidos sino por lo agradable y satisfactorio que resulta trabajar con alguien tan excelente en el campo personal como en el profesional. Hoy no estaría aquí escribiendo estas líneas sin su ayuda constante, algo de lo cual siempre estaré agradecido.

Así mismo debo darle las gracias a Vicente Herrero, Ricardo Colom, Jose María Monzó, Christoph Lerche, Jorge Martinez y Ángel Tebar por hacer todo lo posible por ayudarme siempre que se lo he pedido. Ellos han sido unos compañeros de viaje envidiables de los que guardo y guardaré un extremadamente buen recuerdo.

Durante mi corta estancia en la universidad de Nagoya, una de las experiencias más interesantes de mi vida, debo dar las gracias por su apoyo y generosidad, no sólo a Miguel Ángel y su familia, sino también a Kato-san, Prem y Sato-sensei. Todos ellos mostraron una generosidad e interés hacia mí que agradezco profundamente.

Por supuesto debo agradecerle a mis padres Rafa y M^{a}José su apoyo y cariño, siento mis ocasionales estados de mal humor ocasionados por las muchas frustraciones que sacar adelante un doctorado supone. Gracias por apoyarme y ayudarme siempre.

Por último pero no por ellos menos importante (más bien al contrario), tengo que darle gracias a María, mi amada, la cual ha soportado durante estos años mis jornadas maratonianas de trabajo y mi comportamiento obsesivo hacia el mismo, tratándome con cariño y comprensión siempre que lo necesitaba. Ella ha sido para mi como una pincelada de color (rojo cochinilla) en un cuadro gris.

A todos: muchas gracias por todo.

Resumen

La presente tesis toma como eje central el modelado de sistemas dinámicos mediante Autómatas Celulares (ACs). Los ACs permiten modelar un sistema enunciando el comportamiento microscópico a fin de obtener un comportamiento macroscópico correcto. Una de los principales campos donde esta metodología ha sido aplicada (y la cual forma otro de los puntos centrales de esta tesis) es el modelado del Grabado Anisótropo Húmedo (GAH). El GAH es un proceso químico el cual permite realizar microestructuras de silicio tridimensionales, lo que le ha permitido convertirse en una importante técnica de microfabricación.

El GAH se utiliza para el micromecanizado de Sistemas Micro-Electro-Mecánicos (MEMS), los cuales consisten en la integración de elementos mecánicos, sensores, actuadores y electrónica en un substrato de silicio común a través de la tecnología de microfabricación. Los MEMS tienen una gran influencia en la industria puesto que dispositivos fabricados mediante esta tecnología se utilizan de forma intensiva en diversos campos tales como: sistemas de seguridad en automoción, sensores de movimiento en electrónica de consumo o inyectores en sistemas de impresión.

El GAH es un proceso complejo cuyo resultado depende en gran medida de los diversos parámetros del proceso: (atacante, temperatura, tiempo), por lo que la utilización de un simulador previo a la realización del experimento puede suponer un gran ahorro en cuestión de tiempo y material. Los simuladores actuales de GAH basados en ACs poseen diversas limitaciones: Tiempos de computación muy elevados debido a los altos requisitos

computacionales de los ACs, un reducido conjunto de calibraciones existentes, así como la imposibilidad de simular el GAH basado en nuevos atacantes tales como TMAH+Triton. La resolución de estas limitaciones es abordada en diversos capítulos de la tesis. En concreto, la tesis realiza las siguientes aportaciones:

- Análisis de la técnica de modelado de ACs usando como caso de estudio la ecualización de ganancias de un detector de fotones gamma basado en cristales contínuos. Asimismo, se evalúa la implementación de ACs sobre FPGAs con el fin de acelerar su simulación.
- Aceleración de los modelos más recientes de AC usados en GAH mediante la computación basada en Unidades de Procesado Gráfico (GPUs).
- Definición de una nueva metodología de calibración de los ACs utilizados en GAH basada en algoritmos evolutivos, así como su aplicación a un amplio rango de procesos basados en GAH tales como los basados en KOH, TMAH, KOH+IPA y TMAH+Triton.
- Redefinición del modelo AC con el fin de eliminar errores intrínsecos a las implementaciones actuales, basados en pasos de tiempos contantes.

La tesis proporciona soluciones a las temáticas enunciadas definiendo nuevas metodologías y demostrando su validez con su implementación y obtención de resultados validados de forma experimental.

Abstract

The main topic of this thesis is the modeling of dynamic systems by Cellular Automata (CA). CA can model a system by defining a set of microscopic rules in order to obtain an adequate macroscopic behavior. One of the main fields where this methodology has been applied (and which is other of the main topics of this thesis) is the modeling of Anisotropic Wet Etching (AWE): a chemical process which allows the creation of three-dimensional silicon microstructures. This feature has made AWE to become an important microfabrication technique.

AWE is used for the micromachining of Micro-Electro-Mechanical Systems (MEMS). MEMS consists of the integration of mechanical elements, sensors, actuators and electronics on a common silicon substrate through microfabrication techniques. MEMS have great influence in the industry as devices manufactured by this technology are used intensively in various fields such as automotive security systems, motion sensors in consumer electronics or injectors in printing systems.

AWE is a complex process and the final result depends largely on various process parameters like the etchant type, process temperature or etch time. The usage of a simulator before performing experiments could result in a great reduction of design time and material utilization.

Existing AWE simulators based on CA have several limitations: very long simulation times due to high computational requirements of the CA, the small set of existing calibrations and the inability to simulate new etchants such as TMAH+Triton. The resolution of these limitations is

addressed in various chapters of the thesis. In detail, this thesis makes the following contributions:

- Analysis of the modeling technique based on CA, using as a case of study the equalization of a gamma ray detector based on a continuous scintillator crystal. The implementation of CA on FPGAs for accelerating CA-based simulations is also evaluated.
- Acceleration of the latest models of CA used in AWE simulations by using graphics processing units (GPUs).
- Definition of a new calibration methodology for AWE-CA based on evolutionary algorithms and its application to a wide range of AWE-based etchants such as KOH, TMAH, KOH+IPA and TMAH+Triton.
- Redefining the AWE-CA model in order to eliminate inherent errors in current implementations, based in constant time steps.

This thesis provides solutions to the issues pointed out by defining new methodologies and proving their correctness with tests and comparisons with experimental data.

Resum

Aquesta tesis toma com eix central el modelatge de sistemes mitjançant Autòmats Cel·lulars (ACs). Els ACs permeten modelar un sistema enunciant el comportament microscòpic amb el objectiu d'obtindre un comportament macroscòpic adequat. Una de les principals àrees on aquesta metodologia ha sigut acceptada com a vàlida (i la qual es altre dels punts centrals d'aquesta tesis) es el modelat del Gravat Anisotròpic Humit (GAH): un proces químic el qual permet realitzar microestructures tridimensionals de silici, la qual cosa li ha permés convertir-se en una important tècnica de microfrabricació.

El GAH s'utilitza per al micromecanitzat de Sistemes Micro-Electro-Mecànics (MEMS). Els MEMS consisteixen en la integració d'elements mecànics, sensors, actuadors i electrònica en un substrat de silici comú a través de la tecnologia de microfabricació. Els MEMS tenen una gran influència en la indústria ja que dispositius fabricats mitjançant aquesta tecnologia s'utilitzen de forma intensiva en diversos camps com ara sistemes de seguretat en automoció, sensors de moviment en electrònica de consum o injectors en sistemes d'impressió.

El GAH és un procés complex i el resultat depèn en gran mesura dels diversos paràmetres del procés: (tipus de atacant, temperatura, temps d'atacat), de manera que l'utilització d'un simulador previ a la realització de l'experiment pot suposar un gran estalvi en qüestió de temps i material. Els simuladors actuals de GAH basats en ACs tenen diverses limitacions: Temps de computació molt elevats a causa dels alts requi-

sits computacionals dels ACs, un reduït conjunt de calibratges existents, així com la impossibilitat de simular el GAH basat en nous atacants com ara TMAH+Triton. La resolució d'aquestes limitacions és abordada en diversos capítols de la tesi. En concret, la tesi realitza les següents aportacions:

- Anàlisi de la tècnica de modelatge de ACs usant com a cas d'estudi l'equalització de guanys d'un detector de fotons gamma basat en cristalls continus. Així mateix, s'avalua la implementació de ACs sobre FPGAs per tal d'accelerar la seva simulació.
- Acceleració dels models més recents d'AC usats en GAH mitjançant la computació basada en Unitats de Processat Gràfic (GPUs).
- Definició d'una nova metodologia de calibratge dels ACs utilitzats en GAH basada en algoritmes evolutius, així com la seva aplicació a una amplia varietat d'atacants com ara KOH, TMAH, KOH+IPA i TMAH+Triton.
- Redefinició del model AC utiliztat en GAH per tal d'eliminar errors intrínsecs a les implementacions actuals, basats en passos de temps comptants.

La tesi proporciona solucions a les temàtiques enunciades definint noves metodologies i demostrant la seva validesa amb la seva implementació, la obtenció de resultats i la comparació amb dades experimentals.

Índice de Siglas

AC	Autómata Celular
ACC	Autómata Celular Contínuo
AG	Algoritmo Genético
API	Interfaz de Programación de Aplicaciones *(Application Programming Interface)*
ASIC	Circuito Integrado de Aplicación Especifica *(Application Specific Integrated Circuit)*
BST	Árbol de Búsqueda Binario *(Binary Search Tree)*
BT	Árbol Binario *(Binary Tree)*
CAM	*Cellular Automata Machine*
CM	Clúster de Memoria
CPH	Clúster de Procesado de Hilos
CPU	Unidad de Procesado Central *(Central Processing Unit)*
CSDL	Lenguaje de Descripción de Estructuras Celulares *Cellular Structure Description Language*
CTS-CCA	Autómata Celular Contínuo de Paso de Tiempo Constante *(Constant Time Step-Continuous Cellular Automata)*
CUDA	*Compute Unified Device Architecture*
DRAM	Memoria de Acceso Aleatorio Dinámica *(Dynamic Random Access Memory)*
DRIE	Grabado Profundo por Iones Reactivos

	(Deep Reactive Ion Etching)
FDTD	*Finite-Difference Time-Domain*
FPGA	*Field Programmable Gate Array*
GM	Grupo de Memoria
GPU	Unidad de Procesado Gráfico
	(Graphic Processor Unit)
GPGPU	Computación de Propósito General sobre GPUs
	(General-Purpose Computing on GPUs)
IPA	Alcohol Isopropílico
	(Isopropyl Alcohol)
KOH	Hidróxido de potasio
LMB	Retícula de Boltzmann
	Lattice Boltzmann Method
LGA	Autómata de Gas Reticular
	Lattice Gas Autómata
MEMS	Sistemas Micro-Electro-Mecánicos
	(Microelectromechanical Systems)
MIMD	Múltiple Instrucciones Múltiples Datos
	(Multiple Instruction Multiple Data)
MS	Multiprocesador *Streaming*
PET	Tomografía por Emisión de Positrones
	(Positron emission tomography)
PMT	Tubo Fotomultiplicador
	(Photomultiplier Tubes)
PRT	Tiempo Anticipado de Extracción
	(Predicted Removal Time)
PRT-CCA	Autómata Celular Contínuo de Tiempo Anticipado de Eliminación
	(Predicted Removal Time-Continuous Cellular Automata)
ROP	Operaciones de Raster
	(Raster Operations)
RPF	Función de Probabilidad de Extracción
	Removal Probability Function
SB-BST	Árbol de Búsqueda Binario Autoequilibrado
	(Self-Balanced Binary Search Tree)
SIMD	Única Instrucción Múltiples Datos

	(Single Instruction Multiple Data)
SPE	Elementos Sinergísticos de Proceso
	(Synergistic Processing Elements)
SPECT	Tomografía Computerizada por Emisión de Fotones Individuales
	(Single Photon Emission Computed Tomography)
TBK	Tareas de Mantenimiento del Árbol
	(Tree Book Keeping)
TMAH	Hidróxido de Tetrametilamonio
	(Tetramethylammonium hydroxide)
TVT	Atravesado vertical del árbol
	(Tree vertical Traversal)
VTS-CCA	Autómata Celular Contínuo de Paso de Tiempo Variable
	(Variable Time Step-Continuous Cellular Automata)

Índice general

Capítulo 1

Introducción

En este capítulo queremos mostrar las razones que han llevado a la realización de toda la labor de investigación presentada a lo largo de esta tesis, así como ofrecer de forma detallada los objetivos marcados a la hora de la definición de la misma. Tras esta primera presentación, se expone la metodología de trabajo desarrollada. Por último se ofrece un pequeño esquema donde se resume de forma breve el desarrollo de la tesis a lo largo de los distintos capítulos.

1.1. Motivación

El modelado de sistemas dinámicos mediante Autómatas Celulares (ACs) ha sido un campo especialmente activo en las últimas décadas. Los ACs han ofrecido un nuevo punto de vista el cual ha permitido modelar sistemas centrándose en el comportamiento microscópico en vez de intentar emular el comportamiento a escala macroscópica. Los ACs han sido aplicados a un amplio rango de campos de la ciencia, tales como reacciones químicas, modelos sociales o ecológicos.

Uno de los campos especialmente interesantes donde esta nueva técnica de modelado ha tenido éxito es el proceso químico denominado *grabado anisótropo húmedo*. Este proceso es utilizado ampliamente en la fabricación

de Sistemas Micro-Electro-Mecánicos (MEMS, del inglés *Electromechanical Systems*) basados en silicio los cuales están cada vez más presentes en una gran cantidad de dispositivos electrónicos tales como teléfonos móviles, impresoras de inyección o sistemas de seguridad en automóviles. Este proceso, en el cual se sumerge la oblea de silicio en una solución, se caracteriza por el hecho de que la velocidad de eliminación de silicio por el atacante depende en gran medida de la orientación del cristal de silicio. Este hecho, unido a procesos litográficos para escoger de forma selectiva las regiones de grabado, hace posible la creación de microestructuras tridimensionales de silicio.

Pese a que los ACs han conseguido simular el proceso de micromecanizado basado en grabado anisótropo húmedo de forma satisfactoria, este método de simulación posee todavía ciertas limitaciones que frenan su utilización en aplicaciones de ingeniería. Estas limitaciones son, en mayor medida, la lentitud de las simulaciones debido a la alta intensidad computacional de los autómatas celulares, así como la falta de una metodología de calibración del modelo para todo el rango de atacantes distintos existentes para este proceso. En la actualidad, los simuladores disponibles pueden tardar varias horas en realizar una simulación. Esto ralentiza en gran medida el proceso de diseño de un MEMS, el cual necesita habitualmente de muchas simulaciones hasta obtener los parámetros óptimos del experimento, tales como el tipo o temperatura de atacante, el tiempo de atacado o la geometría de las máscaras aplicadas. La aparición en los últimos años de nuevas estructuras computacionales masivamente paralelas tales como los procesadores gráficos ofrecen nuevas oportunidades para solventar este problema. Asimismo, cabe la posibilidad de que aplicando una reformulación del modelo, sea posible aumentar la eficiencia de este tipo de simuladores.

Por otro lado, en los últimos años han aparecido nuevos atacantes para el grabado anisótropo húmedo, los cuales ofrecen nuevas características geométricas muy interesantes. Estos nuevos atacantes han evidenciado los problemas existentes en los actuales métodos de calibración de los ACs para el grabado anisótropo húmedo, los cuales no son capaces de obtener un comportamiento adecuado del AC. Es, por tanto, necesaria una nueva metodología que permita calibrar el AC para cualquier atacante, pudiendo

así incrementar la versatilidad del modelo.

Creemos, por tanto, que el solucionar las limitaciones actuales de las simulaciones basadas en ACs para este proceso químico puede acelerar en gran medida los procesos de diseño de MEMS, lo cual facilitaría el proceso de fabricación, o incluso permitir la obtención de nuevas estructuras a partir de la experimentación con un simulador fiable y eficiente.

1.2. Objetivos

La presente tesis, la cual se centra en los apartados teóricos y de implementación de los modelos atomísticos utilizados para la simulación del grabado anisótropo húmedo, persigue los siguientes objetivos:

- **Demostrar la viabilidad de la utilización de Unidades de Procesado Gráfico (GPUs, del inglés *Graphics Processing Units*) para la simulación eficiente de algoritmos basados en ACs.** Las GPUs ya han demostrado en la actualidad su buena adaptación a los AC. En este punto queremos centrarnos en la posibilidad de simular ACs los cuales emulan superficies dinámicas. Para ello debemos intentar adaptar la estructura de datos del árbol octal para su uso en GPUs. Los árboles octales son ampliamente utilizados en este tipo de modelos debido a que proporcionan una gran reducción en la utilización del memoria por parte del AC.

- **Acelerar la velocidad de las simulaciones del proceso de grabado anisótropo húmedo utilizando nuevas arquitecturas computacionales que exploten el paralelismo inherente del modelo.** Una de las grandes limitaciones de los simuladores actuales de este proceso son los grandes tiempos de proceso requeridos (hasta varias horas) para obtener resultados. La aplicación de nuevas arquitecturas tales como las GPUs puede suponer un gran avance en los procesos de diseño de MEMS que se basen en la utilización de este proceso químico.

- **Obtener un procedimiento de calibración con el que poder**

aumentar la funcionalidad de los modelos atomísticos para el grabado anisótropo húmedo. En especial es importante calibrar las nuevas soluciones aplicadas a este proceso tales como el atacante TMAH+Triton. Las limitadas calibraciones actualmente existentes de los modelos atomísticos para grabado anisótropo húmedo, así como la imposibilidad de simular nuevos atacantes debido a limitaciones en los métodos vigentes de calibración es otra de las grandes problemáticas de hoy en este tipo de modelos. La obtención de alguna nueva metodología de calibración incrementaría la utilidad de estos modelos para aplicaciones de ingeniería en gran medida.

- **Intentar minimizar los errores de cálculo debido a la actual implementación no exacta del modelo.** La eficiencia computacional de la versión exacta de los modelos atomísticos para grabado anisótropo húmedo (y en definitiva cualquier método atomístico que se base en la reducción de ocupaciones de sus átomos), es extremadamente baja, haciendo obligatoria la utilización de modelos aproximados que introducen errores de simulación. Uno de los objetivos de esta tesis es cuantificar la cantidad de error introducida por esta aproximación, así como proponer modificaciones en el modelo teórico que permitan reducir el coste computacional de la implementación exacta del modelo.

Estos objetivos son abordados a lo largo de los distintos capítulos de la tesis.

1.3. Metodología

La resolución de los objetivos propuestos para esta tesis ha requerido el uso de una metodología de trabajo la cual abarca distintas facetas: análisis del problema, estudio de trabajo previo, diseño de nuevas metodologías así como la implementación y verificación de las mismas.

La figura 1.1 muestra un esquema de la metodología utilizada en la obtención de todos los aportes obtenidos en esta tesis. Tras una breve

etapa de identificación del problema al que nos deseamos enfrentar, se ha realizado una etapa de obtención de información bibliográfica con el fin de descubrir si se ha intentado solucionar dicho problema con anterioridad, o obtener más detalles sobre la magnitud o el origen de la problemática.

La etapa de definición de nuevas metodologías que solventen los problemas propuestos es eminentemente teórica. En ella se aborda la problemática definida, ideando nuevos procedimientos que superen las actuales limitaciones.

Una vez definida una nueva metodología, es necesario implementarla con el fin de comprobar los buenos resultados. En los algoritmos implementados en la presente tesis se ha utilizado el lenguaje de programación Java, utilizando también C y CUDA C en el caso de la implementación de algoritmos sobre GPUs. La elección de CUDA C sobre Nvidia como plataforma de desarrollo sobre GPUs ha sido escogida debido a la gran cantidad de documentación existente, en forma de libros, manuales y ejemplos. Los algoritmos secuenciales que requerían un alto rendimiento, tal como el manejo del árbol octal explicado en el capítulo 4, han sido programados en C. Por otro lado, aquellos que no presentaban grandes requisitos de computación, tales como las interfaces gráficas, se han implementado en Java. Las GPUs utilizadas a lo largo de la presente tesis ha sido los modelos Nvidia GeForce 9800GT, Tesla C1060 y un grupo de cinco GeForce GTX260.

Una vez desarrollado el algoritmo, la etapa de evaluación comprueba tanto la corrección en su implementación como sus características relativas a la resolución del problema propuesto.

Una vez verificada la corrección de la solución propuesta se ha procedido a su correcta documentación mediante la escritura de artículos y de la presente tesis.

1.3.1. Herramientas Utilizadas

En la etapa de estudio de trabajo previo, se ha recurrido a los portales científicos habituales para la obtención de documentación. Estos portales son: *www.sciencedirect.com*, *www.ieeexplore.com* y el nuevo portal *www.scholar.google.es* Asimismo, se ha utilizado con frecuencia la aplica-

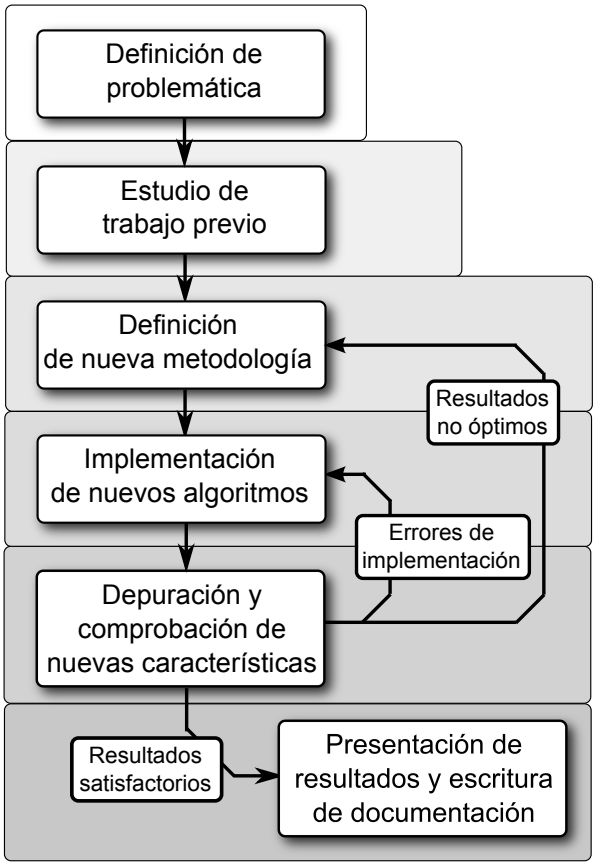

Figura 1.1: Metodología de diseño utilizada a lo largo del proceso de realización de la tesis.

ción científica MATLAB para comprender mejor los modelos matemáticos presentados en ciertos artículos, tales como pueden ser los autómatas celulares continuos o los algoritmos genéticos. MATLAB ofrece un lenguaje de programación de muy alto nivel de abstracción, una metodología que permite operar con matrices de forma natural, así como una gran cantidad de funciones matemáticas de todo tipo que facilitan el tratamiento de datos.

En el proceso de implementación de nuevos algoritmos, la programación en Java se ha realizado sobre el Entorno de Desarrollo Netbeans 6, desarrollado por Oracle, utilizando la versión 1.6 del JDK de Java. Para el desarrollo de aplicaciones sobre GPUs se ha utilizado el Kit de desarrollo 2.3 de Nvidia CUDA junto al entorno de desarrollo Microsoft Visual C++ Express 2008.

Por último, en la etapa de comprobación de los nuevos algoritmos implementados, aparte de las propias aplicaciones desarrolladas, para las aplicaciones basadas en GPU se ha usado de forma frecuente el software Nvidia CUDA Visual Profiler, el cual es capaz de mostrar gráficamente una gran cantidad de detalles del proceso de ejecución de un algoritmo sobre una GPU de Nvidia. Esta herramienta es extremadamente útil para optimizar la implementación de un algoritmo debido a que provee información detallada de efectos típicos en la ejecución de un algoritmo sobre una GPU tales como divergencia de los hilos, patrones de acceso poco eficientes o el alto/bajo flujo de instrucciones por ciclo. Si los resultados de la verificación no son satisfactorios a alguno de los niveles, será necesario volver atrás y rediseñar el algoritmo.

1.4. Esquema de la tesis

Los capítulos de la presente tesis muestran el proceso de investigación incremental llevado a lo largo de la realización de la misma.

En primer lugar, en el capítulo 2 se describe de forma breve la información obtenida en la tarea de documentación de la tesis, realizando un estado del arte sólido el cual trata todos los campos utilizados en la tesis: Teoría y modelado de AC, simulación eficiente de AC, computación con GPUs y el proceso de grabado anisótropo húmedo para tareas de microme-

canizado del silicio. Numerosa documentación del trabajo previo realizado en estos campos es aportada en forma de referencias.

En el capítulo 3 se muestra el trabajo de investigación realizado previo a la principales aportaciones. Las tareas realizadas fueron dos:

- El facilitar la implementación de ACs en dispositivos tipo *Field Programmable Gate Arrays*(FPGAs), una estructura hardware que se ha utilizado de forma tradicional para acelerar la simulación de ACs.
- Aplicar la metodología de modelado de ACs en una aplicación real: Modelar un detector indirecto de fotones gamma para aplicaciones de medicina nuclear.

El capítulo 4 ahonda en dos de los objetivos principales marcados por esta tesis: la implementación eficiente de modelos atomísticos que modelan superficies dinámicas en GPUs, así como la aceleración de los modelos atomísticos actuales para el grabado anisótropo húmedo. En este capítulo se detalla la metodología propuesta para solventar el primer objetivo, y se describen las características (algoritmos y caracterización de la ejecución) de la implementación llevada a cabo con el fin de solucionar el segundo objetivo. El punto final del presente capítulo ha sido el desarrollo de una aplicación completa la cual ofrece todas las características necesarias para simular de forma eficiente una gran variedad de procesos de microfabricación de MEMS basados en este proceso químico.

El capítulo 5 presenta un algoritmo evolutivo, el cual, basándose en el simulador explicado e implementado en el capítulo 4, permite la calibración del modelo atomístico para un amplio rango de atacantes. Junto a una caracterización detallada del método propuesto, un amplio rango de resultados para distintos atacantes es mostrado. El método propuesto resuelve de forma satisfactoria las limitaciones actualmente existentes de las técnicas de calibración actuales.

El capítulo 6 ahonda en el problema de la ineficiencia de las implementaciones exactas de los modelos atomísticos usados en el grabado anisótropo húmedo. Un análisis detallado de los costes de simulación es realizado. Asimismo, se presenta una reformulación del modelo que permite una simulación secuencial del modelo atomístico con tiempos de ejecución simi-

lares a las implementaciones aproximadas actuales sin introducir errores intrínsecos.

Finalmente, en el capítulo 7 se resume todo el trabajo aportado en esta tesis y se presentan las publicaciones publicadas a raíz de las contribuciones realizadas. El trabajo presentado en esta tesis no es cerrado, por lo que en este capítulo se sugieren líneas futuras de investigación donde continuar la labor realizada. Finalmente, la tesis se cierra con los indices de figuras y tablas y con las referencias bibliográficas utilizadas.

Capítulo 2

Estado del Arte

2.1. Introducción a los Autómatas Celulares

Los *Autómatas Celulares* (ACs) son el eje central de esta Tesis, por lo que creemos adecuado realizar en primer lugar, una introducción breve y concisa de lo que son los AC, así como hacer comprender al lector su relevancia en las últimas décadas como herramienta de modelado.

2.1.1. Definición de Autómata Celular

Los ACs se han convertido en las últimas décadas en una poderosa herramienta de modelado físico. Matemáticos, físicos e ingenieros entre otros han encontrado en los ACs un nuevo entorno de trabajo, un nuevo punto de vista con el que modelar un amplio rango de sistemas dinámicos que abarcan, entre otros muchos campos, reacciones químicas, modelos sociales o dinámica de gases. Asimismo, los ACs se han utilizado como un nuevo método de computación para operaciones como procesado de imágenes o criptografía.

Los ACs son un modelo matemático utilizado para definir sistemas dinámicos, teniendo como característica fundamental que el espacio y el tiempo son variables discretas. Los ACs pueden ser entendidos como idealizaciones de sistemas físicos.

Realizando una definición formal, un AC está formado por los siguientes elementos [1]:

- Una retícula regular de *células* las cuales cubren una porción de espacio n-dimensional.
- Un conjunto $\phi(\vec{r},t) = \phi_1(\vec{r},t), \phi_2(\vec{r},t), ..., \phi_m(\vec{r},t)$ de m variables booleanas relacionadas con cada célula $\vec{r}$ de la retícula, el cual representa el estado local de cada célula en el instante de tiempo t.
- Una *regla de evolución* $R = R_1, R_2, ..., R_m$ la cual especifica la evolución de los estados $\phi(\vec{r},t)$ de la siguiente forma:

$$\phi_j(\vec{r},t+1) = R_j(\phi(\vec{r},t), \phi(\vec{r}+\vec{\delta_1},t), \phi(\vec{r}+\vec{\delta_2},t), ..., \phi(\vec{r}+\vec{\delta_q},t)) \quad (2.1)$$

  donde las células situadas en $\vec{r}+\vec{\delta_k}$ afectan al estado futuro de la célula situada en $\vec{r}$ y son denominadas la *vecindad* de dicha célula.

Expresado en términos cualitativos: En un AC, el espacio está discretizado. Cada porción de espacio está representada por una célula, las cuales están agrupadas en una retícula n-dimensional. Cada célula alberga un estado booleano (similar a una variable de estado de un sistema dinámico). El estado de todas las células es modificado de forma paralela y sincronizada en pasos de tiempo discretos acorde a una regla de evolución la cual depende del estado de un conjunto de células, habitualmente adyacentes, denominadas *vecinas*, en el instante de tiempo previo.

En la práctica, un AC no puede ser infinito, por lo que a los elementos anteriormente descritos es necesario añadir también unas condiciones de contorno las cuales se deberán aplicar en los casos donde la vecindad de cierta célula quede fuera de los límites definidos para el AC. Estas condiciones de contorno son aplicadas sobre las reglas de evolución. Tradicionalmente se han propuesto cuatro posibles condiciones de contorno distintas:

- Periódicas: La retícula asemeja un plano n-dimensional cerrado, donde no existen bordes o límites.

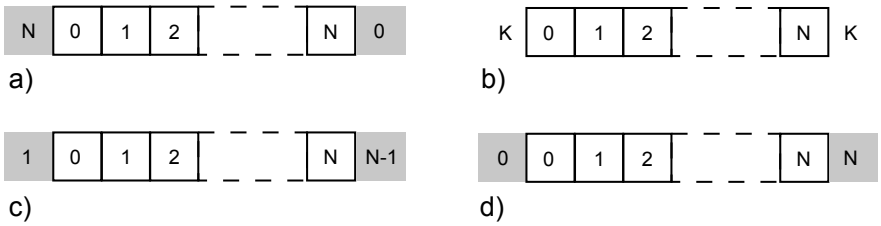

Figura 2.1: Condiciones de contorno típicas utilizadas en los AC. (a) Periódicas. (b) Fijas. (c) De reflexión. (d) Adiabáticas

- Fijas: Las células vecinas localizadas fuera de los límites de la retícula son representadas con valores fijos.
- Reflexión: En este caso, la célula vecina $\vec{r} + \vec{\delta}$ fuera de los límites es substituida por la célula en el sentido contrario, $\vec{r} - \vec{\delta}$.
- Adiabáticas: También llamada *condición de gradiente cero*, donde la célula vecina no existente se substituye por la propia célula.

Una representación gráfica de estos cuatro casos típicos aparece en la figura 2.1.

La aplicación de condiciones de contorno es un claro ejemplo de adición de inhomogeneidades en la regla de evolución. Una regla es llamada *homogénea* cuando la operación aplicada es similar para todas las células de la retícula, independientemente de su posición en $\vec{r}$. Como veremos mas tarde, la adición de inhomogeneidades espaciales o temporales es una posibilidad más aplicable a los AC.

Con respecto a la vecindad, teóricamente no existe ninguna restricción sobre la cantidad o posición relativa de las células pertenecientes a la misma, sin embargo, habitualmente suelen ser las células mas cercanas en el espacio las que pertenecen a la vecindad. Tres de las vecindades más usadas en muchos de los modelos basados en ACs son: Vecindad de Von Neumann (fig. 2.2 (a)), vecindad de Moore (fig. 2.2 (b)) y vecindad de Margolus (fig.

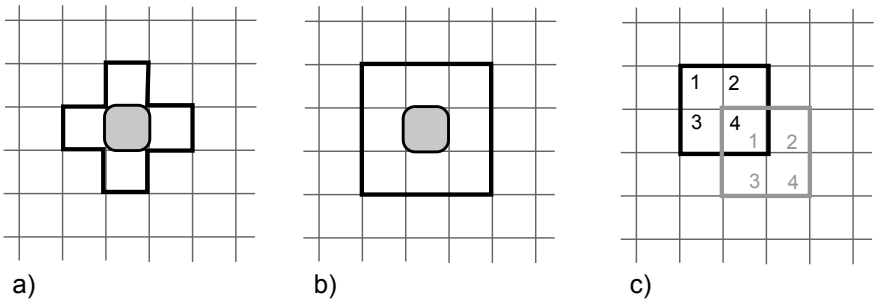

Figura 2.2: Tres típicos casos de vecindad. El reborde grueso representa las células presentes en la regla de evolución de la célula coloreada. (a) Vecindad de Von Neumann 2D de rango 1. (b) Vecindad de Moore 2D de rango 1. (c) Vecindad de Margolus, las particiones gris y negra se alternan en el tiempo. La posición de cada célula es distinta según la partición.

2.2 (c)). Particularizando para un plano 2D, la vecindad Von Neumann de rango r de una célula $\vec{r} = (x, y)$ serán aquellas células localizadas en $\vec{r} + \vec{\delta} = (x + \delta_x, y + \delta_y)$ tal que $|\delta_x| + |\delta_y| \leq r$. La vecindad de Moore de rango r está delimitada por aquellas células que cumplan: $|\delta_x| \leq r, |\delta_y| \leq r$.

La vecindad de Margolus es un caso particular de los ACs de partición, la cual ha sido utilizada especialmente en retículas 2D y está basada en la partición de la retícula en bloques de 2x2 átomos. La evolución del estado de las células es decidida de forma global para toda partición. La partición varía según el paso de tiempo, lo que permite la propagación de información entre bloques. La utilidad de este tipo de vecindad reside en la posibilidad de simplificar las reglas de evolución para algunos modelos como, por ejemplo, aquellos que modelan dinámica de gases o física de materiales granulares [2].

Pese a que la definición de ACs que ha sido presentada en este apartado es válida y ampliamente aceptada, el concepto de AC ha sido modificado o manipulado desde sus primeras definiciones para adecuarlo a nuevos escenarios de modelado. Dos modificaciones ampliamente extendidas que ejemplifican este hecho son las de representar $\phi(\vec{r}, t)$ mediante variables

continuas y no booleanas. Este tipo de ACs son llamados *Autómatas Celulares Continuos* (ACC). Del mismo modo, la aplicación de la Regla R para la obtención de $\phi(\vec{r}, t+1)$ puede depender, no únicamente de estados de la vecindad en el paso de tiempo t, sino también en $t-1, t-2, ..., t-k$. Este tipo de ACs son denominados de orden $(k+1)$.

Sin embargo, pese a las múltiples y variadas modificaciones de los ACs que se han presentado en distintos trabajos de investigación, la filosofía de modelado subyacente de los ACs permanece intacta. Esta metodología, explicada en detalle en la sección 2.1.2.3, se basa a grandes rasgos en definir la forma en la que interactúan las células con el objetivo de obtener un determinado comportamiento global del sistema.

2.1.2. Historia de los Autómatas Celulares

En esta sección describimos el papel de los ACs a lo largo de la historia, su descubrimiento y el gran campo de aplicaciones donde han sido utilizados, desde sus orígenes hasta las últimas aportaciones.

2.1.2.1. Autómatas Autoreproducibles de von Neumann

El concepto de AC fue presentado por primera vez por John von Neumann. von Neumann es reconocido en la actualidad como uno de las personas que, gracias a sus aportes en ciencia computacional, ayudaron a definir la arquitectura hardware que poseen las computadoras hoy en día. La idea de von Neumann cuando definió los *Autómatas Celulares Autoreproducibles* era la de imitar la estructura del cerebro humano con el objetivo de resolver problemas de gran complejidad. El deseo de von Neumann era el de proponer un nuevo punto de vista en los aspectos fundamentales de la computación.

Él pensaba que una máquina de complejidad equiparable al cerebro humano debía tener mecanismos de autoajuste y autoreparación, así como el hecho de que en este nuevo tipo de sistema de computación, los procesadores y los datos no debían ser dos entes diferenciados, sino que los datos almacenados y estructura del sistema debían estar fuertemente interrelacionadas. Siguiendo las indicaciones de S. Ulam [3], von Neumann

definió su nueva arquitectura de computación a partir de un universo discreto formado por células [4]. Cada célula poseía un estado interno que típicamente consistía en una cadena de bits. Este sistema de células debía evolucionar en pasos de tiempo discretos, como autómatas simples que computan su nuevo estado interno. La regla que determinaba la evolución debía ser la misma para todas las células: una función que dependía del estado de la propia célula y de las células vecinas. Del mismo modo que sucede en un sistema vivo, todas las células debían evolucionar de forma simultánea. Este sistema dinámico discreto definido por von Neumann es el tipo de sistema que hoy denominamos AC.

Von Neumann consiguió su objetivo de encontrar una estructura discreta capaz de generar nuevos individuos idénticos a si mismos. Con los autómatas celulares auto-reproducibles, abrió las puertas de nuevas máquinas capaces de otras máquinas de igual complejidad.

La regla de evolución definida por von Neumann tiene la propiedad de *Computación Universal*. Esto significa que existe una configuración inicial para el autómata autoreproducible el cual llega a la solución de cualquier algoritmo de computación. Pese a que esta característica es más interesante desde el punto de vista teórico que desde el práctico, significa que cualquier circuito computacional puede ser simulado por la regla del autómata. Esto puede llevar a pensar que es posible obtener comportamientos complejos e inesperados a partir de una regla de un AC.

Otros científicos han seguido el trabajo de von Neumann en esta línea de investigación y el problema sigue siendo de interés. Actualmente, el campo de la vida artificial sigue siendo estudiado en profundidad. Los ACs fueron un primer intento en esta dirección.

2.1.2.2. Creación de Universos Sintéticos

En 1970 John Conway presentó un AC que captó una gran atención del público. Dicho AC, denominado *el Juego de la Vida* [5], tenía la característica de que, pese a poseer una regla de evolución muy simple, era capaz de generar un comportamiento extremadamente complejo.

Morfológicamente, el AC del juego de la vida está compuesto por una

retícula cuadrada de células 2D. El estado de cada célula está definido por un único bit, representando (1) una célula viva y (0) una célula no viva. La regla de evolución del Juego de la vida es la siguiente:

$$\phi(\vec{r}, t+1) = \begin{cases} 1, & \text{si } \phi((i,j),t) = 1 \text{ y } S((i,j),t) = 2,3 \\ 1, & \text{si } \phi((i,j),t) = 0 \text{ y } S((i,j),t) = 3 \\ 0, & \text{en otro caso} \end{cases}, \tag{2.2}$$

$$\text{donde} \quad S((i,j),t) = \sum_{\delta_1=-1}^{1} \sum_{\delta_2=-1}^{1} \phi((i+\delta_1, j+\delta_2), t).$$

La variable S almacena la cantidad de células vecinas (vecindad von Neumann) que tiene la célula evaluada. Una célula pasará a viva si tiene tres vecinas vivas. Se mantendrá viva si tiene dos o tres vecinas vivas y morirá en cualquier otro caso.

Pese a esta sencilla regla, la evolución del AC a lo largo del tiempo puede llegar a mostrar un comportamiento realmente complejo. Una gran cantidad de estructuras han sido identificadas en este sistema [6]. Una de las estructuras mas famosas son los *gliders*, los cuales son capaces de moverse a lo largo de la retícula en dirección recta. La figura 2.3 muestra la evolución a lo largo del tiempo de varios estados iniciales para este AC. Del mismo modo que los autómatas de von Neumann, el AC Juego de la Vida también posee la cualidad de computación universal.

La propiedad de muchas reglas de ACs de ser computacionalmente universales hizo a varios autores pensar en la posibilidad de que el mundo podría ser pensado como un AC extremadamente grande. T. Toffoli comparó los ACs a un universo sintético donde las leyes físicas son expresadas como reglas de evolución en una estructura donde el espacio y el tiempo son discretos [7]. Toffoli, N. H. Margolus y E. Fredkin reconocieron la importancia de los ACs como un entorno de modelado de sistemas físicos. Su investigación se centraba en describir la analogía existente entre la teoría de la información y las leyes de la física. Los ACs supusieron un marco excelente para desarrollar sus ideas. En concreto, una de sus primeras aportaciones fue la de demostrar la posibilidad de construir lógica

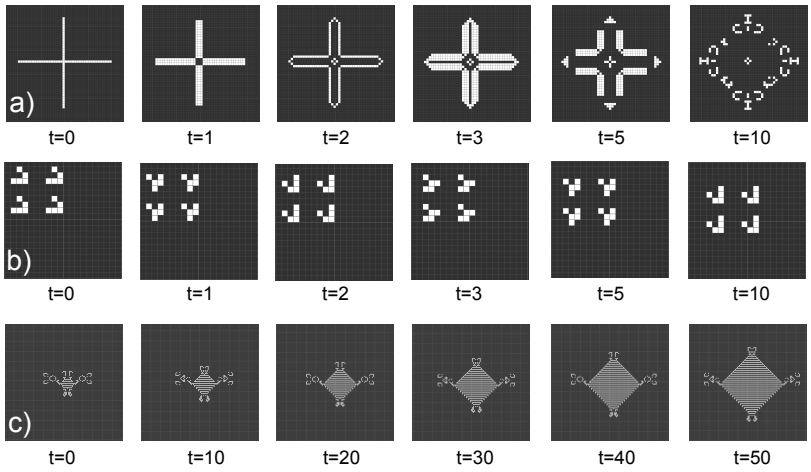

Figura 2.3: Simulación del AC Juego de la Vida para varios estados iniciales. (a) Una cruz como estado inicial. (b) 4 Gliders avanzando en dirección diagonal. (c) Space Filler.

completamente reversible a partir de la cual es posible construir cualquier operación numérica sin pérdida de información, es decir reversible en el tiempo. El paradigma de esta investigación es el modelo de computación de *Bolas de Billar* introducido por Fredkin [8] y modelado como un AC por Margolus [9]

La investigación del matemático S. Wolfram también consiguió fijar la atención del gran público en los ACs. Wolfram se dedicó a analizar sistemas extremadamente sencillos, los llamados *Autómatas Celulares Elementales* [10]. Dichos autómatas unidimensionales, donde cada célula sólo puede estar viva (1) o muerta (0), tienen como vecindad sus dos células adyacentes y la regla de evolución puede ser entendida como una simple tabla de verdad de tres entradas (la propia célula y la vecindad). Una de las conclusiones más sorprendentes que emerge de los estudios de Wolfram es que sistemas realmente sencillos como los que él analizó, son capaces de mostrar un comportamiento extremadamente complejo a escala macroscópica.

2.1.2.3. Modelado de Sistemas Físicos Mediante Autómatas Celulares

Fue en los años 80 donde se realizó un paso importante en la teoría de los AC. Se descubrió que un tipo de modelo de gas reticular, llamado HPP en honor a sus autores J. Hardy, Y. Pomeau y O. Pazzis [11], era realmente un AC. Los modelos de gases reticulares consisten en una retícula discreta 2D donde las partículas se mueven y colisionan a lo largo de las direcciones designadas por la retícula, conservándose el momento y la cantidad de partículas. Los modelos HPP fueron ideados inicialmente para observar características estadísticas de un gas donde las partículas interactúan. La implementación de este modelo como un AC dio lugar al planteamiento de la posibilidad de modelado de un sistema real de partículas mediante un AC.

De esta forma, los ACs son considerados como una forma alternativa de realidad microscopia la cual daba lugar al comportamiento macroscópico deseado. Fue en 1986 cuando U. Frisch, B. Hasslacher y Y. Pomeau presentaron el modelo conocido como FHP, en honor a sus autores [12]. Los autores demostraron que este modelo, a pesar de ser totalmente discreto, obedecía las ecuaciones de Navier-Stokes a nivel macroscópico.

Este tipo de AC como HPP o FHP son llamados *Autómatas de Gases Reticulares* (LGA, del inglés *Lattice Gas Automata*), para distinguirlos de las definiciones menos restrictivas de los ACs. Los LGA han sido muy exitosos en modelar situaciones complejas donde las técnicas de computación tradicional no son aplicables.

Un modelo relevante derivado de los LGA para la simulación de dinámica de fluidos son las *retículas de Boltzmann* (LBM, del ingles *Lattice Boltzmann Method*) [13]. Las LBM pretendían eliminar el ruido estadístico obtenido de los LGA debido a su naturaleza discreta. La filosofía de los LBM es la de substituir en una región espacial, la cantidad de variables booleanas (partículas) por un valor que determine el valor medio, la denominada *función de distribución de densidad*.

La verdadera innovación de los ACs fue la de ofrecer un punto de vista sencillo a la hora de modelar sistemas físicos. Una de las grandes

cualidades de los ACs es la posibilidad de diferenciar entre comportamiento microscópico y macroscópico. Una descripción microscópica de un sistema es aquella descripción completa donde se define el comportamiento de cada partícula del sistema. Por ejemplo, la descripción microscópica de un gas sería la lista del estado de todas las partículas de dicho gas (posición y velocidad). La descripción macroscópica del mismo sistema se limita a definir un comportamiento colectivo del sistema. La técnica de modelado con AC se centra en presentar una simplificación de las leyes microscópicas de determinado fenómeno con la finalidad de que el sistema presente un comportamiento adecuado a escala macroscópica.

Modelar un sistema a nivel microscópico tiene diversas ventajas importantes. La interpretación de la dinámica de un sistema en términos de sencillas reglas microscópicas ofrece una forma intuitiva de modelar fenómenos que son difíciles de representar con métodos de modelados más tradicionales como las ecuaciones diferenciales. Claro ejemplo de esto es la definición de las condiciones de contorno, las cuales son definidas en los ACs de forma natural ya que tienen una interpretación directa a este nivel.

El diseño correcto de un AC implica que los aspectos esenciales de un fenómeno complejo han sido reconocidos y reducidos a una forma tratable con sencillez. Este proceso de reducción de un comportamiento complejo a una suma de mecanismos simples es un paso esencial en cualquier investigación científica. Por lo tanto, los ACs se nos muestran como una herramienta muy poderosa en campos de investigación, ingeniería y docencia.

2.1.2.4. Autómatas Celulares en la Actualidad

En las últimas dos décadas los ACs han demostrado ser una herramienta realmente poderosa en una gran variedad de áreas de conocimiento. Dentro de la gran cantidad de investigación relacionada con el modelado basado en AC, existe una serie de campos donde el volumen de producción y relevancia de los resultados obtenidos de la aplicación de los mismos es notoria. Estos campos lo forman:

- Generación de números aleatorios y criptografía

El comportamiento complejo que puede mostrar un AC a nivel macroscópico ha sido explotado para la generación de números aleatorios. Ya S. Wolfram en 1986 comprobó la habilidad de algunos ACs básicos de generar secuencias lo suficientemente complejas y diferentes entre sí a medida que evolucionaban como para superar una buena cantidad de test de calidad de números aleatorios [14]. Posteriormente M. Tomassini obtuvo varias combinaciones y fórmulas con las que conseguir generadores de números aleatorios de alta calidad [15]. La simplicidad de diseño y la facilidad con la que es posible realizar su implementación paralela hardware hacen de los ACs una buena opción a la hora de crear generadores de números aleatorios hardware con una ocupación reducida de recursos [16, 17].

Por otro lado, la criptografía es otro campo donde las cualidades de los ACs han sido propuestos como posible método. En concreto, los ACs se han utilizado en métodos criptográficos de clave secreta. Fue también Wolfram en 1986 quien propuso por primera vez la utilización de ACs con este fin [18]. Desde entonces, numerosas aportaciones han aparecido analizando la seguridad de los AC[16, 17]. Asimismo, buen porcentaje de los trabajos presentados en relación a este campo están orientados en parte a la implementación hardware con el fin de conseguir el proceso de cifrado-descifrado en tiempo real [19].

- Modelos sociales

Uno de los campos que han hecho a los ACs famosos debido a su adaptabilidad al problema es el de los modelos sociales. Simulación de procesos de evacuación [20], flujos de peatones a través de una calle [21], crecimiento de las ciudades [22], e incluso el comportamiento del mercado de valores [23], han sido realizados mediante modelos basados en AC. En concreto, un área de aplicación donde los ACs han sido especialmente exitosos ha sido el de la simulación de tráfico dentro de una ciudad [24]. El modelo a escala microscópica es realmente sencillo: un coche, que avanza con cierta dirección, avanzará a través de la retícula hasta llegar a las proximidades de otro elemento, como un semáforo u otro coche. Pese a la simplicidad de la regla de evolución, los resultados a escala macroscópica son lo bastante precisos.

- Modelos biológicos y epidemiológicos

En este campo, los ACs se han utilizado para estudiar la propagación de las enfermedades. Principalmente, dos ámbitos han sido modelados: El crecimiento de una enfermedad dentro del cuerpo humano y la propagación de una enfermedad a través de la población. Con respecto al primer ámbito, en los últimos años una fuerte corriente se ha centrado en el modelado de la evolución de tumores dentro del cuerpo humano [25, 26, 27]. Por otro lado, el modelado de epidemias en la población ha sido otro punto importante [28]. Enfermedades tales como el HIV y la hepatitis B han sido claros ejemplos donde la técnica de modelado de ACs ha sido aplicada satisfactoriamente [29, 30]

- Modelos ecólógicos

Otra de las aplicaciones de los ACs es la de su utilización como un paradigma para el modelado de sistemas ecológicos [31]. En concreto, el desarrollo de la vegetación [32] y la simulación de incendios forestales [33] han sido dos ámbitos donde los CA han mostrado su valía.

- Modelado de reacciones químicas

En 1989 M. Gerhardt y H. Schuster presentaron un AC el cual emulaba las estructuras formadas a partir de la reacción de Belousov-Zhabotinsky [34]. Los ACs han sido utilizados para el modelado de sistemas de reacción-difusión [35], así como en procesos de solidificación [36] y recristalización[37].

Otro tipo de reacciones químicas donde se ha demostrado la valía de los ACs sobre otros métodos ha sido el grabado anisótropo del silicio, cuyos primeros modelos se mostraron en 1995 [38]. Este proceso químico es de utilidad para la fabricación de microestructuras basadas en silicio. Este último campo es de especial interés puesto que parte de la presente tesis se centra en ACs utilizados para este propósito.

Pese a la gran heterogeneidad de los campos presentados en los anteriores puntos, todos tienen en común su buena adaptabilidad al modelado

basado en ACs. En todos ellos es sencillo discretizar el espacio y encontrar de forma clara unas interacciones microscópicas entre las distintas posiciones de la retícula que dan lugar al comportamiento macroscópico deseado. En esta sección se ha mostrado la versatilidad de los ACs: sus células pueden representar elementos tan distintos como coches que avanzan por la calzada, células del cuerpo humano, átomos de silicio o incluso una porción espacial de un bosque o una selva.

En conclusión, desde su descubrimiento, el modelado basado en ACs ha ido extendiéndose a distintas áreas de conocimiento, siendo hoy en día una herramienta de modelado multidisciplinar de reconocida valía.

2.1.3. Implementación de Autómatas Celulares en Distintos Recursos Computacionales

Del mismo modo que los ACs se han convertido en una poderosa herramienta de modelado, la simulación de estos sistemas ha sido un campo el cual también ha recibido la atención de la comunidad científica. Los ACs son característicos por el gran coste computacional que puede suponer su simulación. Para avanzar la simulación un paso de tiempo, todas las células de la retícula deben ser evaluadas, calculando de forma individual su estado para el siguiente paso. Esto da lugar a que el coste computacional de un AC es de $O(N^d)$ por paso de tiempo, siendo d el número de dimensiones de la retícula y siendo N la dimensión del AC por dimensión. Cuando se realiza el modelado mediante ACs, en la mayoría de los casos no existe una relación directa con las dimensiones macroscópicas del modelo y el tamaño de N, sino que habitualmente se aplica un factor de escalado, donde cada célula representa una parcela arbitraria de espacio. En estos casos, el parámetro N es utilizado para seleccionar la resolución del sistema. Un sistema con mayor N realiza una discretización mas detallada del espacio, y por ende, la discretización del tiempo sufre el mismo efecto (determinado efecto requiere más pasos de tiempo para propagarse determinada distancia).

Para estos ACs, un mayor valor de N consigue desacoplar mejor interferencias que la escala microscópica puede producir en los resultados macroscópicos del AC (fig. 2.4). Para el modelado de sistemas de dos y tres

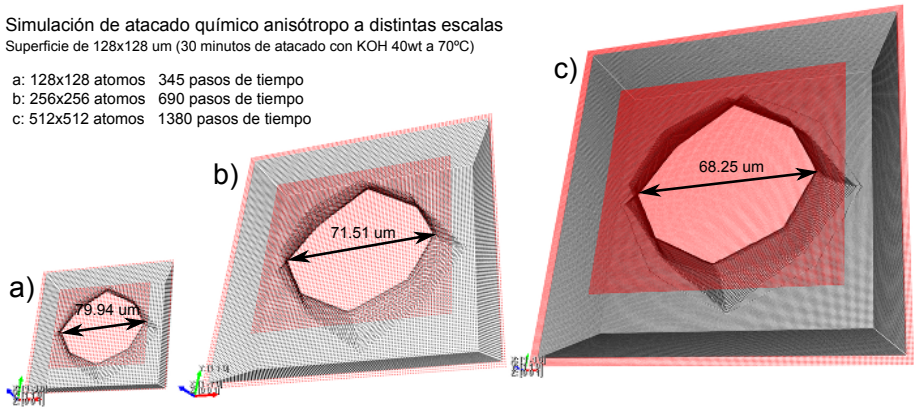

Figura 2.4: Simulación de grabado anisótropo de silicio mediante un AC a tres resoluciones distintas. Los resultados macroscópicos, tales como la anchura de la estructura final, se ve afectada por la resolución del AC.

dimensiones, el coste computacional es $O(N^2)$ y $O(N^3)$ respectivamente, por lo que en modelos de estas características, aumentar la resolución del modelo conlleva un aumento en el coste computacional enorme.

El alto coste computacional de los ACs ha propiciado la búsqueda de distintos métodos de acelerar la simulación de estos modelos. Estos métodos tratan de implementar los ACs en otras arquitecturas de computación más allá de los microprocesadores estándar, los cuales procesan la evolución del AC en un paso de tiempo mediante un bucle que atraviesa todos los átomos uno a uno. Estos métodos se han basado siempre en buscar recursos de computación que exploten de forma eficiente el paralelismo que existe de forma intrínseca en los ACs. En los siguientes apartados realizaremos un análisis de las distintas técnicas que tradicionalmente se han utilizado para reducir los tiempos de simulación de los ACs.

2.1.3.1. Diseño de Hardware Ad-hoc

La simplicidad conceptual de los ACs, así como el paralelismo inherente en ellos hizo que, a partir de la creación de los primeros modelos físicos

basados en ACs, se pensara en el diseño de arquitecturas computacionales especialmente orientadas a su simulación. Las adaptaciones hardware de un AC tienen una arquitectura común. Están compuestas por:

- Un conjunto interconectado de unidades lógicas que representan las células. Cada célula posee lógica la cual modela las reglas de evolución indicadas para el AC.
- Uno o varios registros que almacenan el estado de la célula. Todos los registros son alimentados por la misma señal de reloj local: los ACs son grandes sistemas síncronos, donde todos los estados evolucionan en el mismo *tick* de reloj.

En la figura 2.5 mostramos un esquema del diseño lógico de la regla 30 de los autómatas elementales de Wolfram [10], donde $s(i,t)$ representa el estado interno de la célula i en el paso de tiempo t. Pese a que este es un sencillo ejemplo, la complejidad de la lógica depende directamente de las reglas microscópicas definidas. Por ejemplo, a medida que el el AC tenga un rango más amplio de posibles estados, más registros por célula serán necesarios, convirtiendo simples líneas en buses de datos. Del mismo modo, la lógica se complicará para trabajar con los nuevos formatos de datos. El caso extremo sería la implementación de ACCs o retículas de Bolzmann, donde las variables de estado pretenden ser continuas. En este caso enteros de 32 bits o formatos de coma flotante podrían ser necesarios, aumentando en gran medida la complejidad del diseño.

La opción de implementar en hardware ACs para acelerar su ejecución fue sugerida ya en 1984 por T. Toffoli [39]. En la misma fecha, Toffoli presentó el sistema CAM, del inglés *Cellular Automata Machine* [40]. CAM es un procesador paralelo especializado para la computación de ACs. Este procesador está compuesto por una colección de unidades lógicas compuestas por Memoria de Acceso Aleatorio Dinámica (DRAM, del inglés *Dynamic Random Access Memory*) local y una tabla de consulta o tabla *look-up* la cual implementaba la regla de evolución. El procesador CAM en su época sobrepasaba en mucho la potencia de los ordenadores clásicos, demostrando su utilidad en múltiples aplicaciones [41].

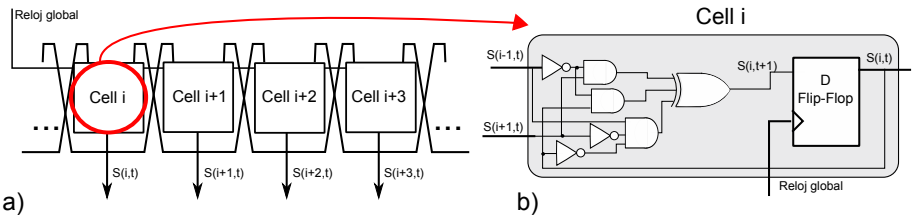

Figura 2.5: Diseño lógico de un AC elemental regla 30 [10] para su implementación hardware. (a) Interconexión entre células. (b) Esquema lógico de una célula.

Asimismo, con la aparición a mediados de los 80 de los dispositivos denominados *Field Programmable Gate Arrays* (FPGAs), los diseños hardware de ACs se redujeron significativamente en dificultad y coste. Una FPGA es un dispositivo semiconductor que contiene bloques de lógica distribuida cuya interconexión y funcionalidad se puede configurar. Las FPGAs se utilizan en aplicaciones similares a los *Circuitos Integrados de Aplicación Específica* (ASICs, del inglés *Application Specific Integrated Circuit*) sin embargo son más lentas, tienen un mayor consumo de potencia y no pueden abarcar sistemas tan complejos como ellos. A pesar de esto, las FPGAs tienen las ventajas de ser reprogramables (lo que añade una enorme flexibilidad al flujo de diseño), sus costes de desarrollo y adquisición son mucho menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor.

Al poco de su aparición, las FPGAs fueron utilizadas para la simulación de ACs. En 1992 aparecieron los primeros documentos utilizando FPGAs para la implementación hardware de sistemas celulares [42]. En 1996 se presentó CEPRA-1X, un procesador celular basado en FPGA [43]. La gran ventaja que los creadores de CEPRA defendían sobre CAM era su mayor versatilidad debido a poder reprogramar completamente la regla de evolución, al estar embebida en la FPGA.

Durante estas dos últimas décadas, las FPGAs han evolucionado tanto en velocidad como en capacidad, por lo que han seguido siendo utilizadas como plataforma para la implementación hardware de ACs con el fin de

acelerar su simulación [44].

2.1.3.2. Entornos de Desarrollo de Sistemas Celulares

Las implementaciones hardware no han sido el único intento de acelerar las simulaciones con respecto a los procesadores secuenciales tradicionales. En los 80 ya existían arquitecturas de computación paralelas, las cuales fueron aprovechadas para la simulación de ACs gracias a lenguajes de programación y entornos de desarrollo que surgieron. A continuación nombramos los entornos de programación más significativos que fueron desarrollados especialmente para la simulación de ACs:

- **CAMEL**[45] El entorno CAMEL fue desarrollado a principios de los 90; tenia como objetivo proporcionar un entorno de simulación de ACs en ordenadores paralelos de tipo *Multiple Instrucción Múltiple Datos* (MIMD). En este tipo de computadoras, los procesadores funcionan de forma independiente y no sincronizadamente, lo cual puede suponer una ventaja a la hora de manejar las posibles irregularidades que pueden surgir a nivel microscópico entre distintas células. CAMEL utiliza como lenguaje de programación CARPET para la definición de ACs [46]. CAMEL ha sido portado a computadoras como Meiko CS-2, Cray T3E, SGI Origin 2000 y clústeres basados en procesadores Intel Pentium [47]

- **StarLogo**[48] Starlogo es un lenguaje con un fuerte componente educacional. StarLogo es un lenguaje *agent-based*, lo que significa que entidades individuales son definidas de forma directa, teniendo cada una de ellas sus propios estados y un conjunto de comportamientos. Los conceptos de *retícula* y *célula* eran adaptados bajo los conceptos de *colonia* y *criatura*, haciendo ver que el AC es como una colonia de muchos individuos que interactúan bajo las reglas definidas. Las primera versión de StarLogo era capaz de ejecutar aplicaciones sobre la computadora Connection Machine 1 CM-1, basada en la aproximación *Instrucción Única, Múltiples Datos* (SIMD, del inglés *Single Instruction, Multiple Data*). En este tipo de computadoras los múltiples elementos de proceso ejecutan las mismas instrucciones de

forma simultanea sobre distintos datos. Otras plataformas como los PCs eran soportadas, aunque el paralelismo en este caso era simulado ejecutando las operaciones de las distintas criaturas de forma secuencial.

- **DEVS-C++**[49] DEVS-C++ es un entorno que aplica el formalismo DEVS [50] el cual soporta el análisis, diseño y simulación de sistemas dinámicos de eventos discretos. Dicho formalismo, basado en la teoría de ACs y en la simulación de eventos discretos, proporciona un método para definir un objeto matemático llamado *sistema*. El sistema está definido por una base de tiempos, entradas, estados, salidas y funciones de transición. DEVS-C++ ofrece la posibilidad de simular sistemas basados en DEVS en computadoras Thinking Machines CM-5 y IBM PS/2.

Además de los entornos descritos, existen otros lenguajes que poseen el mismo objetivo tales como NEMO[47], P-CAM [51] y PECANS [52].

2.1.3.3. Nuevas Arquitecturas Computacionales

En los últimos años, el paralelismo ha irrumpido de forma intensiva en las arquitecturas de computación orientadas al gran público. Los procesadores actuales, tales como los Intel Core i7 [53] pueden poseer más de 4 procesadores por núcleo los cuales pueden ejecutar código de forma independiente. Dentro de estos mismos núcleos existen unidades de cálculo SIMD [54], las cuales también pueden realizar múltiples cálculos en paralelo. Asimismo, arquitecturas de computación más complejas y especializadas en computación masivamente paralela han aparecido en el gran mercado. Las dos arquitecturas más significativas son:

- La arquitectura Cell BE, desarrollada por IBM
- Las *Unidades de Procesado Gráfico* (GPUs, del inglés *Graphics Processing Units*).

La arquitectura Cell BE está basada en ocho *Elementos Sinergísticos de Proceso* (SPEs, del inglés *Sinergetic Processing Element*) los cuales pue-

den realizar operaciones SIMD. Sus registros de entrada son de 128 bits, pudiendo operar sobre datos de 8, 16, 32 y 64 bits. Estos ocho elementos son controlados por un procesador maestro basado en arquitectura Power PC. El procesador maestro trabaja con sistemas operativos convencionales debido a su similitud con otros procesadores Power PC de 64 bits, mientras que las SPEs están diseñadas para la ejecución de código vectorizado en coma flotante. Esta arquitectura ha demostrado ser extremadamente rápida en algoritmos paralelizables que puedan hacer uso simultáneo de los SPE [55].

Por su lado, las GPUs se han convertido en la actualidad en una opción interesante de cara al procesado masivamente paralelo. En el presente doctorado nos hemos centrado en el análisis de esta arquitectura para la implementación de ACs debido no sólo a su buena adaptación a los algoritmos basados en ACs, sino a su reducido precio y al simple hecho de que en la actualidad, las GPUs ya están integradas como parte de cualquier computadora personal, aumentando de forma espectacular la difusión de cualquier avance realizado en este campo hacia el publico general. En la próxima sección hablamos detalladamente de las GPUs y su viabilidad para la implementación de ACs.

2.2. Unidades de Procesado Gráfico

Una GPU es un procesador dedicado al cálculo de las operaciones aritméticas relacionadas con la representación de gráficos por parte de una computadora. En la actualidad, todas las computadoras personales poseen uno o varios procesadores de este tipo y su función es aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos y/o aplicaciones 3D interactivas. De esta forma, mientras gran parte de lo relacionado con los gráficos se procesa en la GPU, la CPU puede dedicarse a otro tipo de cálculos (como la inteligencia artificial o los cálculos mecánicos en el caso de los videojuegos).

Una GPU implementa ciertas operaciones específicas llamadas *primitivas* optimizadas para el procesamiento gráfico. Existen primitivas para dibujar rectángulos, triángulos, círculos y arcos. Las GPUs actualmente

disponen de gran cantidad de primitivas, buscando mayor realismo en los efectos. Una de las más comunes para el procesamiento gráfico en 3D es el efecto de reducción de dientes de sierra (en inglés *antialiasing*), que suaviza los bordes de las figuras para darles un aspecto más realista.

Impulsados en mayor parte por la industria de los videojuegos, los cuales requieren habitualmente de renderizado de entornos gráficos complejos, estos procesadores han aumentado su potencia computacional de forma vertiginosa en los últimos años. Este proceso ha propiciado la visión de estos procesadores como unidades de procesado masivamente paralelo de alto rendimiento, superando en al menos un orden de magnitud la potencia de cálculo de los procesadores centrales. Las razones que han llevado a la rápida propagación de la *Computación de Propósito General en GPUs* (GPGPU, del inglés *General Purpose Computing on GPUs*) son:

- La gran relación potencia de cálculo/precio.
- La flexibilidad de programación que ofrecen respecto a otras arquitecturas tales como las FPGAs.
- Su gran potencial de implantación puesto que cualquier computadora moderna tiene la capacidad de ejecutar algoritmos de propósito general en su GPU [56].

En la actualidad las GPUs de las últimas generaciones son denominadas también procesadores basados en muchos núcleos (llamados de tipo *many-core*). Esta denominación, junto a los procesadores multi-núcleo (llamados de tipo *multi-core*) forma parte de las dos tendencias de diseño actuales de microprocesadores [57]. Los procesadores *multi-core* intentan maximizar el rendimiento de la ejecución secuencial de instrucciones, habilitando la ejecución simultánea de un número reducido de procesos o hilos. Claro ejemplo de este tipo de procesadores son las últimas generaciones introducidas por los fabricantes Intel y AMD para computadoras personales. Por otro lado, los procesadores *many-core* tiene como objetivo maximizar el flujo de ejecución de algoritmos paralelos. Estas arquitecturas están formadas por un gran número de procesadores. Como ejemplo, la GPU

Nvidia GeForce GTX460 posee 336 procesadores que ejecutan código de forma simultánea [58].

2.2.1. Evolución Histórica de las GPUs

Debido a la naturaleza secuencial del renderizado de gráficos 3D y a la complejidad de los cálculos, dicho proceso se divide en un conjunto de etapas, el cual es llamado tradicionalmente *graphics pipeline* o *pipeline* gráfico. Las GPUs pueden verse como implementaciones hardware de estas etapas de renderizado. Tradicionalmente, la renderización de gráficos 3D por una aplicación se realiza a través de un *interfaz de programación de aplicaciones* (API, del inglés *Application Programming Interface*) tales como Direct3D o openGL. La utilidad de estas APIs es la de proveer de un estándar de forma que a un programador que escriba una aplicación para el renderizado de gráficos 3D, le sea posible ejecutarlo de forma independiente en cualquier hardware que soporte dicha API. La conexión entre las instrucciones generadas por la API y el hardware implementado en la GPU es realizada por los controladores que manejan el dispositivo hardware, los cuales deben ofrecer soporte a la API determinada.

El *pipeline* implementado en las tarjetas gráficas de Nvidia GeForce consiste en las siguientes etapas [59]:

1. Interfaz con el sistema anfitrión (en inglés *Host*): Comunica la GPU con el PC. A través de esta interfaz se reciben tanto los comandos provenientes de la API como datos.

2. Control de Vértices: La GPU recibe los datos de las figuras 3D como un conjunto de triángulos. Esta etapa transforma los datos de los triángulos recibidos a un formato que el hardware entienda y almacena los datos en la caché de vértices.

3. Transformación e iluminación/sombreado de vértices: Se modifican los objetos 3D actuando sobre sus vértices. Estas modificaciones comprenden: transformación (rotación, escalado), asi como la asignación de valores a los vértices tales como colores, normales para iluminación o coordenadas de texturas.

4. Cofiguración de triángulos: Realiza operaciones paras las posteriores operaciones de rasterizado.

5. Rasterizado: Determina qué píxeles de la imagen final están contenidos en cada triángulo. Para cada píxel se interpolan valores por vértice necesarios para sombrear el píxel, tales como color, posición o posición de textura.

6. Sombreado de Píxel: Determina el color final de cada píxel. En esta etapa se aplican efectos que pueden hacer la imagen más realista.

7. Operaciones de Raster (ROP, del inglés Raster Operation): Realiza las últimas operaciones de raster sobre los píxeles. Aplica operaciones que mezclan el color de objetos adyacentes para aplicar efectos de trasparencia y suavizado de bordes.

8. Interfaz con la memoria de fotograma (en inglés *framebuffer*): Administra las lecturas y escrituras a la memoria de *framebuffer*. Esta memoria almacena la información utilizada para representar la imagen por pantalla.

Durante los últimos años, cada generación sucesiva de GPUs y de APIs han ido aportando mejoras al *pipeline* gráfico con el objetivo de ofrecer renderizado de gráficos de mayor calidad.

La necesidad de hacer las APIs más flexibles, así como obtener mayor realismo en los gráficos, hizo que se añadiera la posibilidad de programar las etapas de sombreado de vértice y píxel. Hasta la fecha estas unidades eran configurables, pero no programables. La versión 8 de la API DirectX ofrecía un lenguaje similar a ensamblador el cual poseía 127 instrucciones que los desarrolladores podían utilizar en las GPUs que las soportaran. La primera GPU en soportar la programación de estas etapas fue la Nvidia GeForce 3 en el año 2001. Asimismo, el requisito de realizar operaciones sobre millones de triángulos o píxeles en una fracción de segundo, siendo muchas de estas operaciones independientes entre sí, impulsó a los fabricantes de hardware a aprovechar esta característica, resultando que varias etapas del *pipeline* estuvieran compuestas por múltiples elementos que procesaban los datos de forma simultánea.

En el año 2006, la GPU Nvidia GeForce 8800 implementaba un vector de procesadores unificados los cuales procesaban las distintas etapas programables del *pipeline*. Este vector estaba dentro de una ruta recirculable, de forma que los datos pasaban varias veces por el vector. Esto se vio reflejado en la versión 10 de la API DirectX, en la cual la funcionalidad de los sombreadores de píxel y vértice se había hecho similar. Otras mejoras notables que se añadieron en esta versión fueron: el soporte de datos enteros y de coma flotante de 32 bits, el soporte de instrucciones de control de flujo y la posibilidad de leer cualquier cantidad de veces datos de la memoria principal.

Con cada nueva generación donde los requerimientos de realismo gráfico son mayores, las GPUs se vuelven más complejas. Las GPUs actuales poseen como núcleo de su arquitectura vectores de cientos de procesadores los cuales son programables, funcionan en paralelo y poseen una gran capacidad de cálculo, especialmente en operaciones de coma flotante. Esto ha dado lugar a que una GPU posea una gran potencia de cálculo que, durante los últimos años, se ha explotado en un gran rango de aplicaciones distintas.

2.2.2. Lenguajes de Programación para GPGPU

El hecho de que las últimas generaciones de GPUs posean una potencia de cálculo mucho mayor que las CPUs abrió las puertas a la posibilidad de ejecutar algoritmos no relacionados con gráficos sobre las GPUs. Este tipo de computación fue llamada Computación de Propósito General en GPUs (GPGPU, del inglés *General Purpose Computing on GPUs*). Fue en 2003 la primera vez que se vio la GPGPU como una opción interesante para la ejecución de algoritmos donde el paralelismo permitiera aprovechar de forma eficiente la arquitectura de las GPU [60]. Aplicaciones que ya se indicaban como factibles para su ejecución en una GPU eran: cálculos basados en la transformada de Fourier, funciones relacionadas con la inteligencia artificial en los juegos o la simulación del crecimiento de cristales. Como desventaja, la única forma de programar las GPUs en aquel momento era utilizando las API gráficas, simulando que los datos almacenados en la GPU eran elementos gráficos. Esto hacía de la computación en las GPUs

muy limitada y complicada.

Este hecho propulsó el surgimiento de numerosos lenguajes de programación orientados hacia la GPGPU. Dos de los primeros en surgir fueron BrookGPU [61], desarrollado por la universidad de Standford y RapidMind [62], el cual puede utilizarse también en procesadores CELL BE o CPUs *multi-core*. Pronto, los fabricantes de GPUs ofrecieron también entornos de programación para sus respectivas arquitecturas. Especialmente exitoso ha sido el entorno desarrollado por Nvidia que presentó en 2007: la arquitectura CUDA [63].

Muy recientemente, dos nuevos lenguajes orientados a la GPGPU han aparecido de mano de los desarrolladores de las APIs de OpenGL (Grupo Khronos) y DirectX (Microsoft): OpenCL [64] y DirectCompute [65] respectivamente. El hecho de que las compañías responsables de las dos grandes APIs de programación de GPUs hayan prestado su atención a las características de computación general de las GPUs modernas, ha sido un gran apoyo hacia este nuevo tipo de computación, aún en fase inicial, debido a que asegura una cierta continuidad y soporte a esta filosofía de computación a medio y largo plazo. Actualmente, los *Kits de Dessarrollo de Software* (SDKs, del inglés *Software Development Kit*) proporcionados por Nvidia soportan tanto CUDA como openCL, mientras que el SDK proporcionado por AMD, AMD Stream SDK [66], ofrece OpenCL como lenguaje principal de programación.

Finalmente es necesario comentar que, pese a la diversidad de lenguajes, debido que a la arquitectura de computación sobre la que todos ellos trabajan es similar, todos ellos comparten unas mismas directrices y características.

En esta tesis se ha escogido la arquitectura CUDA como soporte para la implementación de algoritmos sobre GPUs. La razón de dicha elección es el gran soporte que Nvidia ofrece a CUDA en forma de libros, manuales, ejemplos de desarrollo, integración con entornos de desarrollo como la familia Visual Studio de Microsoft y la gran comunidad de desarrolladores que posee.

2.2.3. Arquitectura CUDA

CUDA (*Compute Unified Device Architecture*), es un modelo de computación desarrollado por Nvidia para sus GPUs a partir del modelo GeForce 8800. Este modelo, presentado en 2007, permite el aprovechamiento del vector de procesadores albergado en la GPU para la computación de algoritmos de propósito general mediante la definición de un lenguaje de programación, similar a C++ con ciertas variaciones, así como un compilador de dicho lenguaje para su ejecución en GPUs.

Desde el punto de vista de CUDA, una GPU es una colección de *multiprocesadores de flujo* o *Multiprocesadores Streaming* (MS). Estos MS son agrupados en *Clústeres de Procesado de Hilos* (CPH), los cuales comparten unidades de textura y cachés para datos de textura y datos constantes. Cada MS asimismo se compone de ocho procesadores escalares (32 en la nueva arquitectura Fermi [67]) dos unidades de funciones especiales (4 en Fermi) y una unidad de instrucciones la cual administra la ejecución de los hilos de ejecución en los procesadores. Estos procesadores están acompañados por un banco de registros de 32 bits, una memoria de datos paralela compartida, una caché para datos constantes y una caché para la lectura de texturas. Todo el conjunto de CPH está interconectado a la memoria principal de la GPU, denominada *memoria global*. La figura 2.6 muestra un diagrama de la arquitectura CUDA para la familia de GPUs Nvidia GeForce serie GT200.

Esta arquitectura está diseñada para ejecutar una gran cantidad de hilos de ejecución en paralelo. Generalmente son necesarios varias decenas de miles de hilos para aprovechar de forma eficiente la GPU. La filosofía de funcionamiento de esta arquitectura hace que la GPU funcione como un coprocesador del procesador central de la computadora. En la aplicación es la CPU la que invoca ejecutar determinada función sobre la GPU (denominada *semilla* o en inglés *kernel*).

El modelo de ejecución de la arquitectura CUDA divide los hilos en tres niveles de jerarquía:

1. Rejilla (del inglés *Grid*) de bloques.

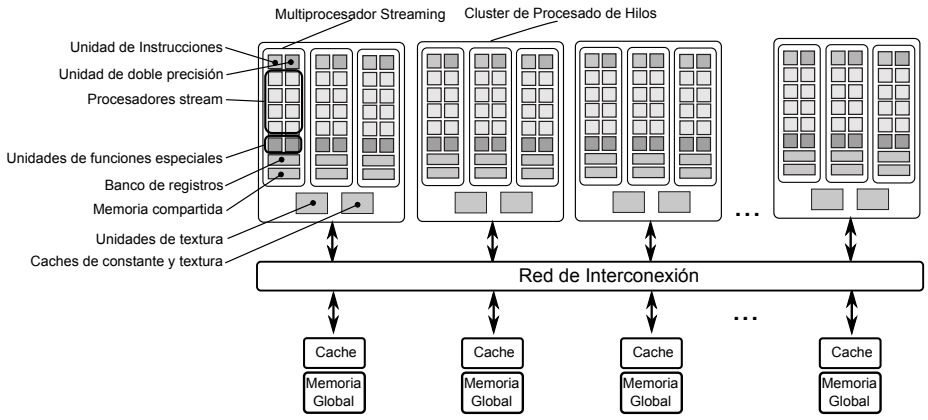

Figura 2.6: Arquitectura CUDA de las GPUs Nvidia modelo GT200 [68].

2. Bloque (del inglés *Block*) de hilos.

3. Urdimbre (del inglés *Warp*) de hilos.

Cuando una aplicación lanza la ejecución de un *kernel* en una GPU, define las dimensiones del bloque en hilos por eje, y las dimensiones de la rejilla en bloques por eje. La combinación de ambas variables define la cantidad total de hilos. La posibilidad de configurar los tamaños y dimensiones de los bloques y la rejilla tiene la función de: (1) poder adaptar la estructura de hilos al hardware de una forma eficiente y (2) poder particionar el algoritmo que se desea ejecutar de una forma natural entre los distintos hilos.

El algoritmo masivamente paralelo lanzado en una GPU puede entenderse como una nube de hilos configurable en tamaño y dimensiones. Como ejemplo, supongamos que deseamos realizar una operación sobre una matriz bidimensional de 1024x1024 elementos. Una posible forma de particionar el problema es que cada hilo procese el dato correspondiente a una única celda, por lo que el problema será resuelto por una matriz de 1024x1024 hilos. El bloque de hilos se puede definir con un tamaño de 16x16 hilos y la rejilla con un tamaño de 64x64 bloques. Múltiples solucio-

nes pueden escogerse definiendo el trabajo a realizar por hilo, la dimensión de los bloques y de la rejilla:

- Procesado de una celda por hilo, bloques de 8x8 hilos, rejilla de 128x128 bloques
- Procesado de 2x2 celdas por hilo, bloques de 16x16 hilos, rejilla de 32x32 bloques

El tamaño máximo del bloque de hilos está limitado por la arquitectura CUDA y es de 512 hilos, siendo las configuraciones más eficientes las que tienen un múltiplo de 32 hilos [69]. Valores habituales son 128 y 256 hilos por bloque.

La ejecución de un *kernel* por parte de una GPU puede ser comprendida, por tanto, como la necesidad de procesar cierta cantidad de bloques de hilos los cuales tienen asignados la ejecución de un determinado *kernel*. Estos bloques son como *paquetes de trabajo* que son asignados a los distintos multiprocesadores para su ejecución. En función de las características del bloque, *kernel* y arquitectura de la GPU, el multiprocesador podrá ejecutar varios bloques de hilos en paralelo (hasta 8 en en modelo GeForce 8800GTX). Cuando todos los hilos de un bloque finalizan su proceso, el bloque se elimina del multiprocesador, liberando recursos y dejando sitio libre para otro. De esta forma, todo el conjunto de bloques van siendo paulatinamente asignados y procesados en los multiprocesadores disponibles de la GPU, finalizando la ejecución del *kernel* cuando todos los bloques han terminado su ejecución. La figura 2.7 muestra una representación gráfica de este proceso para el procesado de una simple operación sobre un vector 1D.

La intercomunicación entre hilos del mismo bloque es altamente eficiente debido a la existencia de memorias rápidas orientada a ello. Sin embargo, la comunicación entre hilos de distintos bloques debe hacerse a través de la memoria global. Una intercomunicación eficiente entre la totalidad de los hilos sería excesivamente cara desde el punto de vista de implementación hardware. La definición en dos jerarquías propuesta por Nvidia permite que grupos de hilos trabajen de forma conjunta para resolver una porción del problema total.

Dentro de cada multiprocesador, los bloques sufren una nueva división: grupos consecutivos de 32 hilos son juntados en un *warp*. De esta manera, un multiprocesador puede procesar en paralelo una cantidad concreta de *warps* (hasta 24 en el modelo GeForce 8800GTX). Debido a la existencia de únicamente una cantidad reducida de procesadores, todos los *warps* asignados a un multiprocesador se alternan entre ellos para ejecutar código. Cuando un *warp* accede a los recursos de computación, todos los hilos ejecutan la misma instrucción, por lo que los procesadores ejecutan en paralelo siempre el mismo tipo de operación. Esto hace que, dentro de un mismo *warp*, todos los hilos estén completamente sincronizados, sin embargo, el orden de ejecución de código entre dos hilos de distintos *warps* o bloques es indefinido, por lo que en estos casos es necesario recurrir a operaciones de sincronización en el caso de que requieran compartir información.

La razón de utilizar este método de alternancia entre *warps* es debido a que el retardo causado debido al acceso a memoria global de un *warp* tiene habitualmente un valor de ente 400 y 600 ciclos de reloj [70]. Teniendo una alta cantidad de *warps* compitiendo por el recurso de computación, otro *warp* puede ocupar su lugar, manteniendo el multiprocesador activo (figura 2.8 (a)). De esta forma es posible aprovechar el paralelismo para ocultar las latencias de acceso a memoria sin la necesidad de utilización de grandes cachés (figura 2.8 (b)).

Es evidente que, a partir de la arquitectura definida, no todos los algoritmos pueden ser implementados eficientemente en una GPU: El principal requisito (pero no el único) es poder dividir el problema a resolver en una gran cantidad de hilos de ejecución que apliquen el mismo algoritmo a distintos elementos. La cantidad de hilos debe ser lo suficientemente grande para mantener a todos los multiprocesadores de la GPU ocupados. Valores habituales rondan entre las decenas y cientos de miles.

La arquitectura CUDA proporciona, además del modelo de ejecución presentado, varias interfaces de memorias con las que interactuar. Las describimos a continuación:

- Memoria Global: Esta memoria representa la memoria principal de la GPU. Su capacidad puede variar entre pocos cientos de MB y varios GB. Cualquier hilo tiene acceso a esta memoria. Es una memoria re-

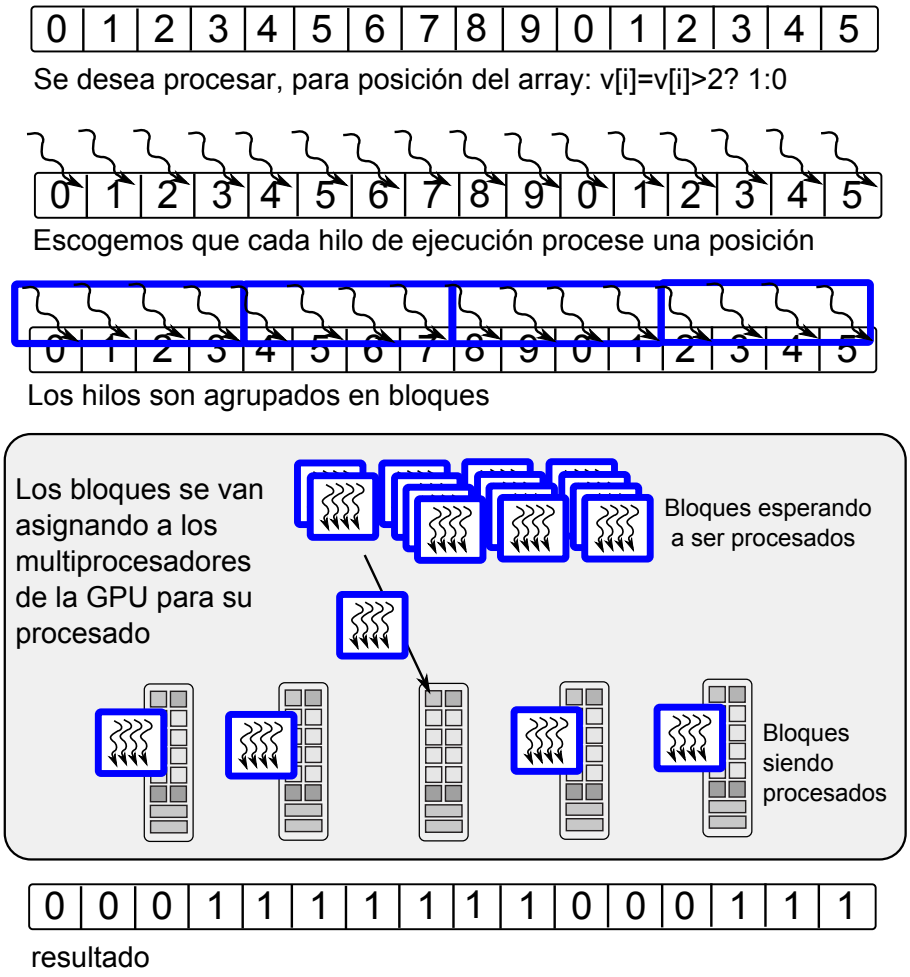

Figura 2.7: División de un algoritmo paralelo para su ejecución en una GPU

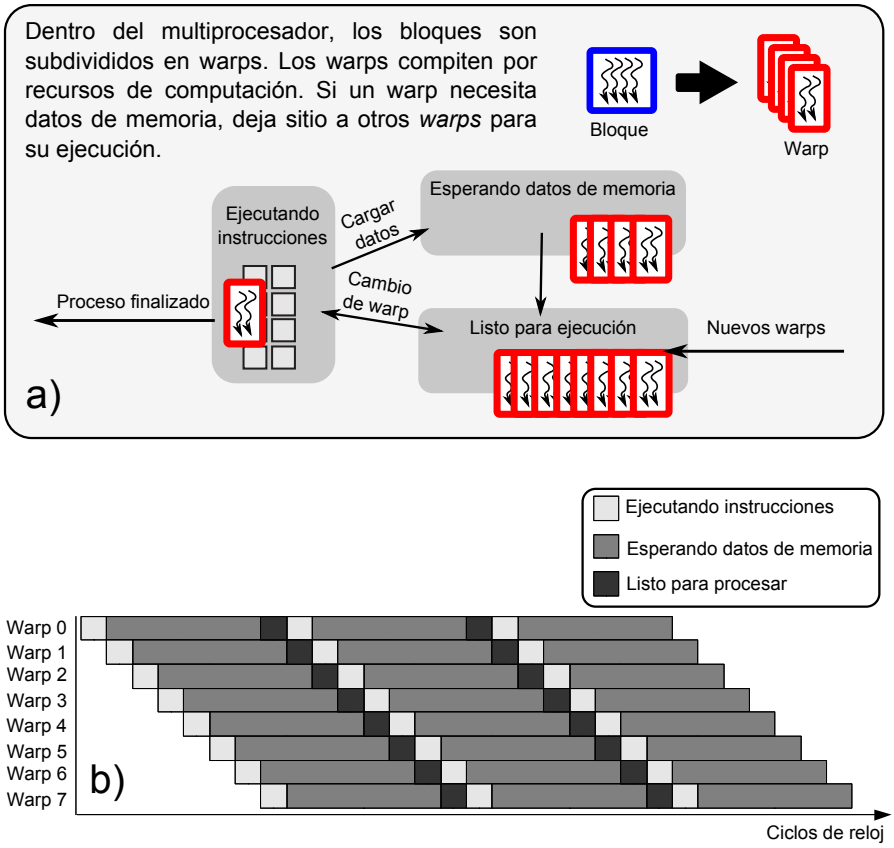

Figura 2.8: Procesado de un diagrama de bloques en un multiprocesador. (a) Diagrama de ejecución. (b) Diagrama de tiempos.

lativamente lenta, con grandes latencias. Los anchos de banda de esta memoria suelen estar alrededor de 100GB/s para el modelo Nvidia GeForce GTX460 [58]. La reserva/liberación de variables dentro de esta memoria es realizada por la CPU. Dichas variables son consistentes durante todo el tiempo de vida de la aplicación. Para obtener un ancho de banda óptimo para este tipo de memoria es necesario un determinado patrón de acceso a memoria. Si los hilos de un mismo *warp* acceden a posiciones contiguas de memoria cumpliendo ciertas restricciones en el tipo de dato y la dirección inicial, es posible unir los accesos a memoria de los distintos hilos en una única operación, minimizando la cantidad de lecturas de memoria. Esta operación es llamada *acceso unificado a la memoria* (en inglés *memory coalescing*) [69].

- Memoria Compartida: Esta memoria de datos está incluida dentro del chip de la GPU. Cada multiprocesador posee su propia memoria compartida, con una capacidad de 16KB para arquitecturas anteriores a las GTX400 y de 48KB para GTX400. Los tiempos de acceso a esta memoria son de alrededor de 38 ciclos por hilo si se accede con un patrón óptimo de acceso [69]. Las variables reservadas en esta memoria son visibles por todos los hilos del mismo bloque, y su tiempo de vida es el tiempo que dicho bloque está siendo procesado. Esta variable suele ser vista como una caché administrada por el programador [71]. Sus principales funciones son:

  1. Reutilizar datos provenientes de la memoria global.
  2. Almacenar de forma temporal datos con el fin de adaptar la lectura de datos de los hilos a un acceso eficiente de la memoria global.
  3. Intercomunicación rápida entre hilos del mismo bloque.

- Registros: Memoria donde se almacena las variables internas de cada hilo. Los tiempos de acceso de estas variables son similares a la memoria compartida. La cantidad de memoria de registros por multiprocesador varía entre 8 y 32KB, la cual se reparte entre todos

los hilos activos en el multiprocesador. *Kernels* más complejos resultarán en la necesidad de más registros por hilo y una reducción de la cantidad máxima de hilos activos por multiprocesador. Esto puede ser un factor importante cuando los algoritmos acceden frecuentemente a memoria global debido a que se necesita una alta cantidad de *warps* activos para ocultar las latencias a esta memoria. Una parte importante de los algoritmos implementados en GPU tienen su cuello de botella en el acceso a la memoria global [72], por lo que es importante mantener una alta cantidad de hilos activos en paralelo (llamado *ocupación*).

- Caché de constantes: La CPU puede definir variables en la GPU de tipo constante. Estas variables son almacenadas en una región de la memoria global la cual está cachéada y no puede ser escrita por la GPU. Aparentemente existen distintos niveles de caché, obteniéndose unas latencias de acceso variables entre 8 y 220 ciclos de reloj para un acierto de esta caché [70].

- Caché de textura: El uso de las unidades de textura para la lectura de datos global es recomendada cuando dichas lecturas poseen localidad espacial pero no cumplen los requisitos para conseguir *memory coalescing*. Estudios realizados evidencian que esta caché está dividida en dos niveles [70], obteniendo una latencia de alrededor de 280 ciclos para aciertos en la caché de primer nivel.

Por último, es importante comentar que los multiprocesadores son capaces de ejecutar cualquier tipo de instrucción utilizada en un algoritmo de propósito general: aritmética de punto fijo y flotante, operaciones de tipo *bit a bit*, comparaciones e instrucciones de salto. Estas últimas son delicadas especialmente si hilos dentro del mismo *warp* saltan a distintos puntos del programa. Un *warp* en proceso ejecuta la misma instrucción para todos los hilos, así que esto obliga a que el *warp* tenga que multiplicar la cantidad de instrucciones ejecutadas por la cantidad de hilos divergentes en un mismo *warp*. También es importante remarcar que en las generaciones más antiguas de GPUs con soporte CUDA, parte de las instrucciones

no cumplen con el estándar IEEE-754, por lo que los resultados pueden no ser exactamente iguales que los obtenidos con una CPU [69].

La arquitectura CUDA, descrita brevemente en este apartado, ha sido muy exitosa en su aplicación de algoritmos masivamente paralelos tales como métodos de conjunto de niveles (en inglés *Level-Set*) [73], métodos de elementos finitos [74], métodos de Montecarlo [75] o simulación de dinámica molecular [76]. Muy recientemente, han aparecido publicaciones mostrando la buena adaptabilidad de ACs a las arquitecturas GPU, obteniendo mejoras de velocidad de más de un orden de magnitud sobre CPUs uni o multiprocesador [77, 78, 79, 80].

En la presente tesis, la arquitectura CUDA es utilizada como base para proponer una nueva implementación de modelos que simulan la propagación de un frente. Dicha implementación es aplicada sobre un AC el cual modela un determinado tipo de reacción química utilizada ampliamente en procesos de micromecanizado, explicada en el siguiente apartado.

2.3. MEMS y Micromecanizado Basado en Grabado Anisótropo Húmedo

Si bien los ACs corresponden a la teoría de modelado de sistemas y constituyen el eje central de la tesis, el grabado anisótropo húmedo para la microfabricación de microestructuras de silicio corresponde a la aplicación práctica en la que todas las aportaciones de la presente tesis se centran. En este apartado realizamos una breve introducción al método de fabricación y su modelado basado en AC.

2.3.1. Introducción a los MEMS

MEMS son las siglas de *Microelectromechanical System*, es decir, sistema micro-electro-mecánico. Los MEMS son sistemas a escala microscópica los cuales incluyen microelectrónica, y microestructuras basadas habitualmente en silicio. Debido a que la microelectrónica es un campo bien definido, la mayor parte de la investigación relacionada con los MEMS está generalmente dedicada a la creación de microestructuras que forman

microsensores o microactuadores.

Entre los dispositivos basados en tecnología MEMS destacan los sensores de presión [81], acelerómetros [82], sensores de flujo de gas [83], microactuadores [84] o microinyectores [85]. La industria basada en MEMS ha sufrido un crecimiento continuo [86]. Campos en la industria donde los MEMS se utilizan de forma habitual son: sensores para automoción tales como acelerómetros para los sistemas de Airbag, cabezales de impresoras de inyección o sensores de movimiento en dispositivos electrónicos tales como teléfonos móviles o controladores de juegos. Asimismo, se espera que mantenga un crecimiento durante los próximo años mediante a una mayor implantación de estos dispositivos [87].

Pese a que es posible fabricar MEMS sobre una gran cantidad de materiales tales como titanio [88] o cerámicas [89], la mayoría de las estructuras de los MEMS utilizan silicio como material principal. Las razones son varias:

- Su coste es reducido comparado con otros materiales. El silicio es un material abundante. Asimismo, existe actualmente una producción en masa de silicio de gran pureza debido a la industria de circuitos integrados.

- Los MEMS se obtienen mediante procesos de microfabricación. Muchos de estos procesos, como la trasferencia de patrones o la oxidación térmica, ya eran aplicados con anterioridad a la fabricación de circuitos integrados, por lo que la utilización de obleas de silicio como base de la producción de MEMS permite la utilización de maquinaria ya existente. [90].

- El silicio constituye un material ideal para la realización de MEMS. Es un material elástico sin deformaciones, bastante inerte y con un alta resistencia a fracturas. Estas características mecánicas han influido de forma decisiva en la estandarización del silicio como material en la fabricación de MEMS [91].

- El silicio presenta interesantes propiedades químicas, obteniendo comportamientos anisótropos cuando se realiza un atacado con diver-

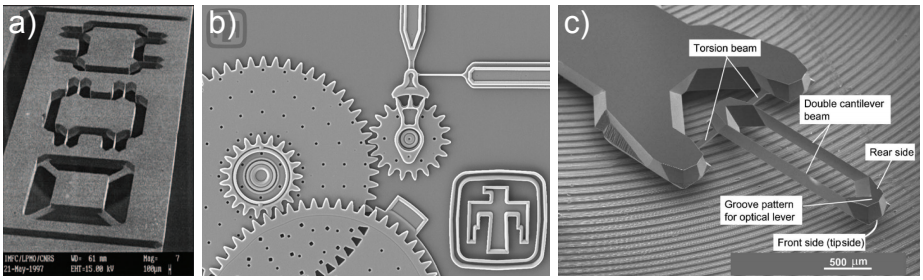

Figura 2.9: Microfotografías de MEMS. (a) Acelerómetro de 3 ejes [93]. (b) Reductor basado en ruedas dentadas [94]. (c) Microsonda de dos ejes [95].

sas soluciones. Estos comportamientos pueden ser aprovechados para crear estructuras complejas.

- Es posible modificar las propiedades eléctricas del silicio aplicando dopajes.

En la figura 2.9 mostramos varias imágenes de MEMS fabricados en silicio.

La unión de todas estas características ha hecho posible la producción en masa de MEMS. Los mayores fabricantes de componentes electrónicos ofrecen un amplio rango de componentes basados en MEMS con precios reducidos. Un ejemplo de este hecho es el micrófono omnidireccional ADMP401 que ofrece Analog Devices por 1.63$ (en pedidos de a partir de 1000 unidades) [92].

2.3.2. Métodos de Micromecanizado

Se denomina *micromecanizado* al conjunto de procesos de fabricación utilizados para la obtención precisa de estructuras a escala del micrómetro. Dentro del mundo de los MEMS existen dos filosofías de fabricación distintas: micromecanizado de volumen o de tipo *bulk* y micromecanizado superficial. En el micromecanizado *bulk* se utiliza el substrato de silicio como parte de la estructura, mientras que en el micromecanizado superficial,

la estructura del MEMS está realizada con capas de materiales depositadas en la superficie del substrato.

El proceso de fabricación de los MEMS consiste en una sucesión de etapas de deposición de materiales, aplicación de máscaras y eliminación de material. Explicamos a continuación los métodos más comunes en los procesos de micromecanizado de MEMS.

Los métodos utilizados para depositar capas de material sobre el substrato comprenden la pulverización catódica (*sputtering*), la epitaxia, la oxidación, la deposición química de vapor, la evaporación y los métodos de recubrimiento por rotación (en inglés *spin-on*):

- La *epitaxia* es un método de deposición utilizado para la creación de una capa de otro material sobre el sustrato elegido. Sus mayores ventajas son mantener la misma estructura cristalina que el substrato sobre el que se aplica (en el caso de que sea silicio) y la posibilidad que ofrece de crear capas de varias decenas de micras.

- La *oxidación* del silicio para la creación de una capa de dióxido de silicio (SiO_2) es un proceso muy utilizado en la creación de MEMS, puesto que el SiO_2 es resistente a los atacantes utilizados habitualmente para el grabado húmedo del substrato. Existen estudios detallados de entornos y tiempo necesarios para obtener una capa de un determinado grosor [96].

- La *deposición por pulverización catódica* consiste en la vaporización de los átomos de un material sólido denominado *blanco* mediante el bombardeo de éste por iones energéticos. Este efecto hace que partículas del material se desprendan y se depositen en la oblea. Este método permite la deposición de películas metálicas de materiales tales como aluminio, titanio, cromo, platino y paladio, así como aislantes como cristal o componentes cerámicos. Una de las grandes ventajas de este método es la uniformidad del grosor de la capa bajo variaciones de la geometría.

- Otro método de deposición es la *deposición por evaporación*, en el cual se calienta el material a una temperatura lo suficientemente alta

que genera vapor. Este vapor acaba condensándose en el substrato. Prácticamente cualquier material puede ser aplicado mediante este método. Como desventaja, es un método muy direccional, por lo tanto ofrece poca uniformidad del grosor cuando existen variaciones en la geometría.

- La *La deposición química de vapor* es realizada iniciando una reacción química en una cámara de vacío, resultando en la deposición. Materiales típicos aplicados con este método comprenden polisilicio, óxidos y nitruros de silicio, tungsteno, titanio, tántalo así como sus nitruros. Más recientemente se está aplicando este método con cobre y aislantes dieléctricos. La deposición de polisilicio, dióxido de silicio y nitruro de silicio son pasos muy comunes en la creación de MEMS. El polisilicio es utilizado habitualmente en el micromecanizado superficial debido a su facilidad para deposición y el hecho de que comparte muchas de las características con el silicio. Por su parte el dióxido de silicio y el nitruro de silicio son utilizados como materiales de protección para procesos tales como la eliminación de substrato mediante grabado.

- Los métodos *spin-on* son característicos para aplicar capas de resinas fotorresistivas, capas gruesas de polímeros o vidrio de depositado por rotación (en inglés *spin-on glass*). Una solución líquida se aplica en el centro de la oblea mientras que se la hace girar para extenderla.

Junto a los procesos de deposición, el proceso de litografía es fundamental puesto que es utilizado para la trasferencia de patrones sobre el substrato o las capas aplicadas sobre él. El proceso litográfico consta de tres pasos: aplicación de la película de una emulsión fotorresistiva, exposición óptica para imprimir la imagen de la máscara en la capa e inmersión en una solución para disolver la película expuesta. El patrón generado con la emulsión fotorresistiva puede ser utilizado como máscara para procesos de grabado. Un proceso típico de fabricación comprende varias etapas de deposición-aplicación de patrones mediante litografía-grabado.

Los procesos de grabado son fundamentales en la creación de MEMS. Su objetivo es el de eliminar selectivamente material para obtener la es-

tructura deseada. Dicha eliminación selectiva se consigue mediante el uso de resina fotorresistiva para la aplicación de máscaras. El patrón puede ser grabado directamente sobre el silicio o sobre una película de material que será usada como mascara para un posterior proceso de grabado. El atacado de películas metálicas es un proceso sencillo [97].

El grabado del substrato de silicio es un proceso más complejo del que existe una gran variedad de metodologías. Una de las metodologías consiste en el *grabado húmedo*[1], donde el substrato es sumergido en una solución que reacciona con los átomos de silicio, eliminándolos de la red cristalina. Según las propiedades de la solución, el atacado puede ser isótropo o anisótropo, siendo este último especialmente exitoso debido a la posibilidad de generar estructuras complejas.

Otra de las metodologías existente es denominada como *grabado seco*. Dentro de ellas, la más relevante es el *grabado profundo por iones reactivos* (DRIE, del inglés *Deep Reactive Ion Etching*). Su característica más destacable es la de poder realizar zanjas con paredes completamente verticales. El único método utilizado en producción actualmente es el desarrollado por Bosch GmbH, que consiste en alternar un atacado por plasma desde una dirección vertical al substrato, con la aplicación de una capa de pasivación que previene el ataque lateral [98]. Este método, pese a ser mas complejo, ha obtenido un gran éxito en el micromecanizado de MEMS debido a la posibilidad de realizar paredes totalmente verticales de gran profundidad, algo imposible de conseguir en la mayoría de casos con otros métodos como el grabado anisótropo húmedo.

Por último mostramos un ejemplo práctico de fabricación donde se aplican varias de las técnicas indicadas en esta sección (figura 2.10). El sistema que se desea fabricar es el acelerómetro mostrado en la figura 2.9 (a). El diagrama está basado en los pasos indicados en [82] y [93].

[1]Grabado húmedo está traducido del inglés *wet etching*. En ocasiones es también traducido por *grabado mojado*.

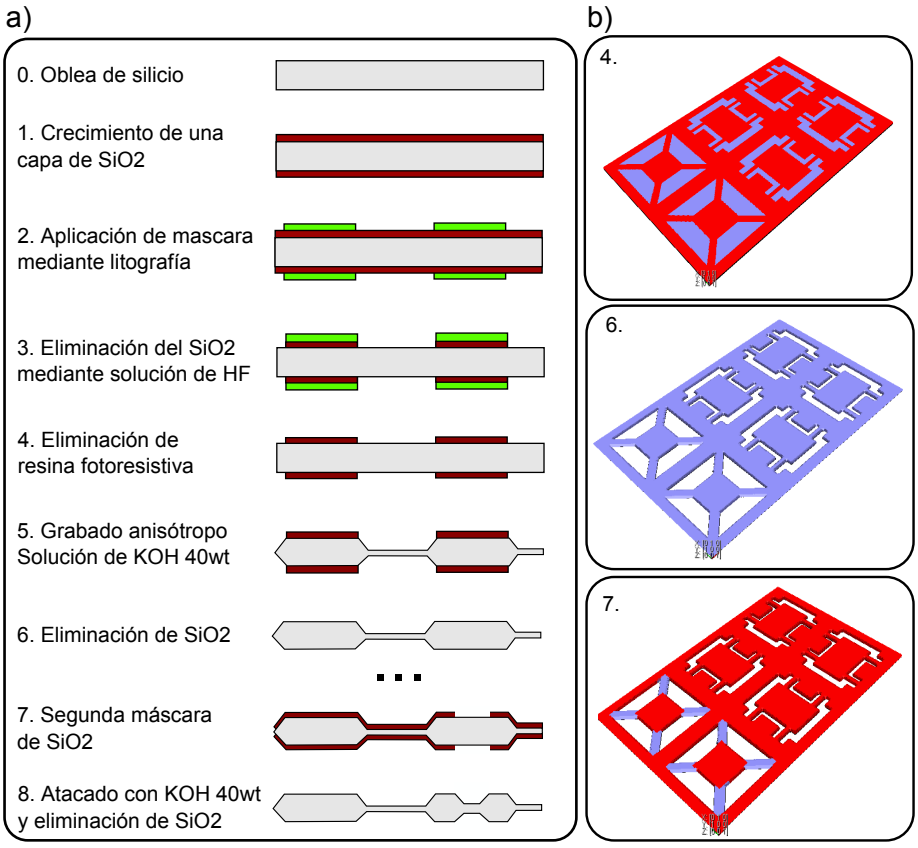

Figura 2.10: Pasos en el micromecanizado de un acelerómetro de 3 ejes [93]. (a) Explicación detallada de los pasos. (b) Representación gráfica del acelerómetro.

2.3.3. Fabricación Basada en Grabado Anisótropo Húmedo

El grabado ha sido tradicionalmente una técnica artística cuyos primeros usos datan del siglo XV. Esta técnica, que se plasma en distintos procedimientos, consiste en crear un relieve de una imagen sobre una superficie rígida llamada *matriz*. Posteriormente, la huella puede alojar tinta para transferir la imagen por presión a otra superficie como papel o tela.

En el campo de los MEMS, el grabado es utilizado para, a partir de un patrón inicial, transferir dicho patrón al silicio con el objetivo de conseguir la estructura deseada. La anisotropía del método define la dependencia con respecto a la orientación de la velocidad de grabado. La figura 2.11 muestra dos casos típicos de grabados anisótropo (a) e isótropo (b) sobre una superficie de silicio. El grabado isótropo avanza de forma uniforme a través del silicio. Por el contrario, en el grabado anisótropo, la velocidad de atacado puede cambiar drásticamente según la orientación con respecto al silicio, provocando la aparición de distintos planos que avanzan a velocidades diferentes.

El *grabado anisótropo húmedo* se ha aplicado exhaustivamente en la fabricación de MEMS. Esto es debido a su relativo bajo coste, su capacidad de obtener superficies planas y su posibilidad de procesado en lotes. El descubrimiento del grabado anisótropo del silicio monocristalino data de los años 60 [99]. La primera aplicación de este método fue presentada en 1980 por E. S. Ammar y T. J. Rodgers para la fabricación de transistores de potencia UMOS [100]. El éxito en el campo de los MEMS viene de la posibilidad de crear estructuras tridimensionales, tales como microcanales o puentes suspendidos. Los dispositivos de las imágenes (a) y (c) de la figura 2.9 están realizados íntegramente mediante grabado anisótropo húmedo.

Desde un punto de vista químico, tradicionalmente se utilizan soluciones alcalinas, tales como las basadas en Hidróxido de Potasio (KOH) o Hidróxido de Tetrametilamonio (TMAH, del inglés *Tetramethylammonium hydroxide*) [101]. La reacción química subyacente de este proceso está dividida en dos etapas: una de oxidación (de naturaleza química o electro-

[2] wt % se refiere al porcentaje de masa de KOH con respecto a la solución acuosa final.

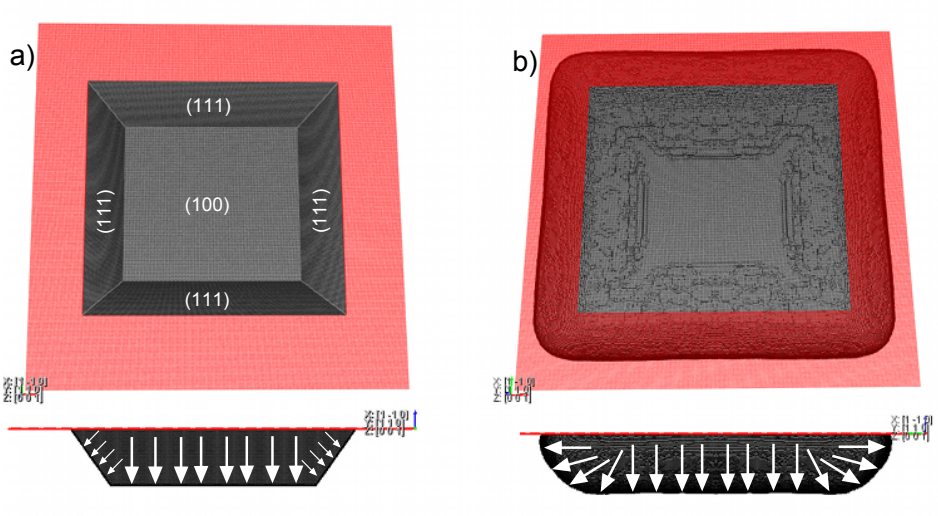

Figura 2.11: Grabado de una superficie de silicio. (a) Comportamiento anisótropo (KOH al 40wt % a 70°C)[2] . (b) Comportamiento isótropo

química) y otra de atacado [102, 103], siendo la primera la que limita la velocidad de grabado. Los átomos de silicio superficiales, los cuales están terminados con átomos de hidrógeno son los partícipes en este proceso.

En la etapa de oxidación, varias de las terminaciones de hidrógeno son substituidas por un grupo hidróxilo (OH^-). En el caso de la oxidación química (ecuación 2.3), el hidróxilo es obtenido de una molécula de H_2O, siendo iones OH^- los catalizadores de dicho proceso. Esta reacción es considerada altamente anisótropa debido a la distinta concentración de terminaciones H, que varía según la orientación de la superficie del silicio, así como a efectos estéricos que limitan la reacción. La oxidación electroquímica (ecuación 2.4) no depende de la cantidad de iones OH^- existentes. En este tipo de reacción se substituye la terminación H por OH, liberando dos átomos H^+ e inyectando dos electrones en la banda de conducción [101]. La oxidación electroquímica presenta un comportamiento isótropo.

Por último, en la etapa de atacado (ecuación 2.5) el átomo de silicio con alguna terminación OH es eliminado de la superficie y pasa a ser enlazado

con cuatro hidróxilos. Asimismo, los átomos de silicio que enlazaban con él pasan a tener una terminación H en dicho enlace. Las reacciones químicas de ambas etapas pueden ser expresadas de la siguiente forma para átomos superficiales con una única terminación H [101]:

$$\equiv \mathrm{Si} - \mathrm{H} + \mathrm{H_2O} + \mathrm{OH^-} \rightarrow \equiv \mathrm{Si} - \mathrm{OH} + \mathrm{H_2} + \mathrm{OH^-}(\text{ox. química}) \quad (2.3)$$

$$\equiv \mathrm{Si} - \mathrm{H} + \mathrm{H_2O} \rightarrow \equiv \mathrm{Si} - \mathrm{OH} + 2\mathrm{H^+} + 2\mathrm{e^-}(\text{ox. electroquímica}) \quad (2.4)$$

$$(\equiv \mathrm{Si})_3\mathrm{Si} - \mathrm{OH} + 3\mathrm{H_2O} \rightarrow 3(\equiv \mathrm{Si} - \mathrm{H}) + \mathrm{Si(OH)_4}(\text{atacado}) \quad (2.5)$$

La anisotropía revelada en la etapa de oxidación química da lugar, a escala macroscópica, a grandes variaciones en las velocidades de atacado del silicio en función de la orientación de la superficie de silicio. Asimismo, existen ciertos parámetros que afectan en gran medida a la anisotropía del atacado. Estos parámetros son: temperatura y concentración de la solución, orientación cristalina de la oblea de silicio utilizada, tipo de atacante y adición de elementos tales como alcoholes o surfactantes.

2.3.3.1. Velocidad de Atacado en Función de la Orientación Cristalográfica.

Debido a la anisotropía de este proceso químico, la especificación de la velocidad de atacado en función de la orientación de la estructura cristalina es la principal forma de caracterizar cualquier atacante. A la hora de definir orientaciones en el substrato de silicio, es común utilizar índices de Miller para definir la orientación de un plano dentro de la estructura cristalina del silicio. Una orientación viene descrita por una tupla de tres números. Los índices de un sistema de planos se indican genéricamente con las letras (hkl), los cuales representan los componentes del vector $< hkl >$ perpendicular al plano que se identifica (figura 2.12 (a)). Los índices de Miller son números enteros, que pueden ser negativos o positivos, y son primos entre sí. El signo negativo de un índice de Miller debe ser colocado sobre dicho número. En el caso del silicio, el cual posee la estructura cristalina del diamante (figura 2.12 (b)), cualquier orientación obtenida a partir de la permutación o cambio de signo de alguno de los elementos de

(hkl) da lugar a la misma estructura cristalina, y por tanto a la misma velocidad de atacado.

Dentro del campo de caracterización de los atacantes, a la hora de representar los resultados, habitualmente se utiliza una proyección de una esfera ideal de silicio. Sobre esta proyección se indican las velocidades de atacado. La figura 2.13 (a) muestra la correlación entre estas proyecciones y la orientación cristalina. Asimismo, mostramos en rojo las orientaciones existentes entre las tres orientaciones principales: $< 100 >$, $< 110 >$ y $< 111 >$. El área dentro de este triángulo contiene todas las posibles orientaciones del cristal de silicio, por lo que la esfera puede construirse a partir de réplicas de este triángulo.

La obtención de las velocidades de atacado para las orientaciones que pertenecen al perímetro del triángulo (2.13 (a)) ha sido una forma tradicional de definir la anisotropía del proceso. Estos datos se obtienen en muchos casos a partir de un método experimental consistente en aplicar un atacado anisótropo sobre una estructura similar a una rueda de carreta con muchas y finas aspas. Cada aspa ofrece una velocidad de retracción en función de su orientación. Los resultados de estos experimentos se muestran habitualmente en una gráfica donde el eje X se presenta los grados de orientación con respecto a cierta aspa (habitualmente la correspondiente a una de las orientaciones principales), y el eje Y representa la velocidad de atacado obtenida a partir de la retracción del aspa. En la figura 2.13 (c), mostramos las orientaciones obtenidas al aplicar este experimento sobre obleas del tipo (100) y (110).

Diversos grupos de investigación se han dedicado a caracterizar la anisotropía de distintos atacantes. Sato *et al.* analizaron la anisotropía de soluciones basadas en KOH y TMAH variando la concentración y la temperatura [104, 105] utilizando estructuras esféricas de silicio. K. A. Wind *et al.* analizaron detalladamente la anisotropía y la estructura resultante a escala mesoscópica del atacado de una estructura en forma de rueda de carreta en KOH para un amplio rango de temperaturas [106]. La adición de elementos a la solución con el fin de modificar la anisotropía también ha sido estudiado en profundidad.

Wind *et al.* e I Zubel *et al.* detallaron los efectos de añadir alcohol

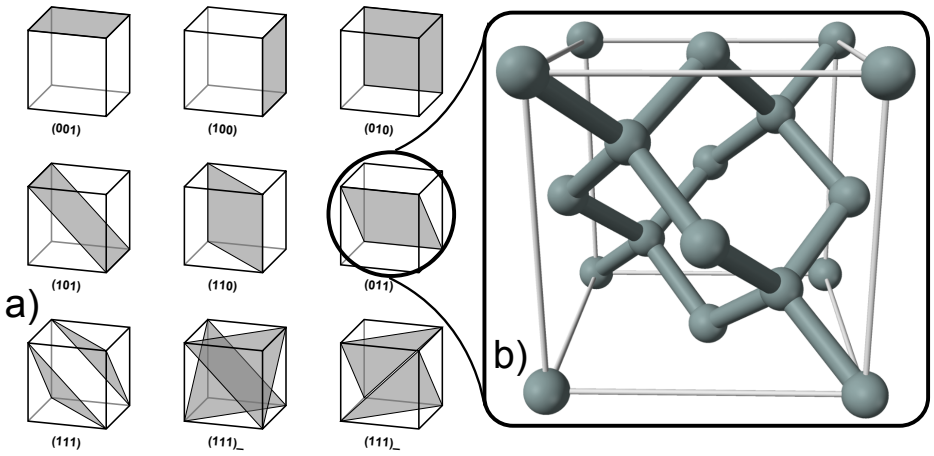

Figura 2.12: (a) Indices Miller para distintas orientaciones de plano que intersecta un cristal cúbico. (b) Estructura cúbica cristalina del silicio (diamante).

isopropílico (IPA) a soluciones basadas en KOH [107, 108, 109]. Asimismo, P Pal *et al.* estudiaron la anisotropía de la adición del surfactante Triton X-100 a soluciones basadas en TMAH [110]. En la figura 2.14 mostramos un resumen de las distintas anisotropías obtenidas en los trabajos anteriormente indicados. En las gráficas se aprecia la gran variación en la reactividad y la anisotropía al modificar el atacante en alguno de sus parámetros (temperatura, concentración, aditivos).

2.3.3.2. Morfología a Escala Macroscópica.

Esta anisotropía en los atacantes se resume en la aparición de ciertas formas características al realizar procesos de grabado. En superficies cóncavas, la orientación más rápida suele dominar el proceso de atacado (figura 2.15 (a)), mientras en las superficies convexas lo hace el plano mas lento, típicamente en la orientación $< 111 >$ (figura 2.15 (b)).

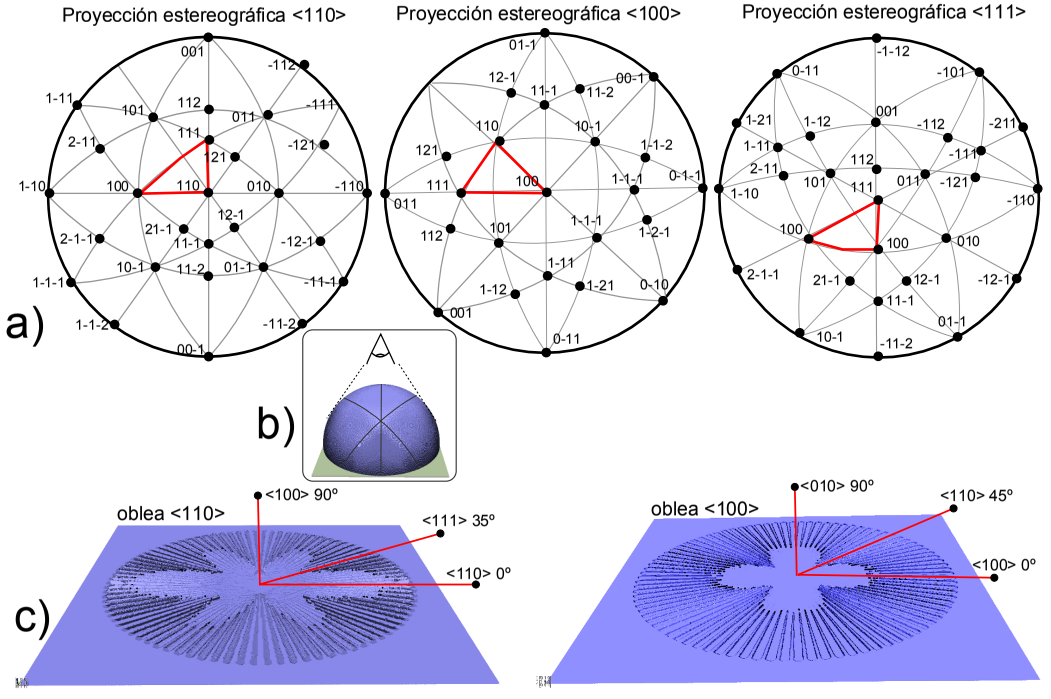

Figura 2.13: (a) Proyección estereográfica de esferas de silicio con las orientaciones $< 100 >$, $< 110 >$ y $< 111 >$ en el origen. (b) Vista 3D de la proyección estereográfica. (c) Orientaciones principales en el atacado de una rueda de carreta para obleas de tipo (100) y (110).

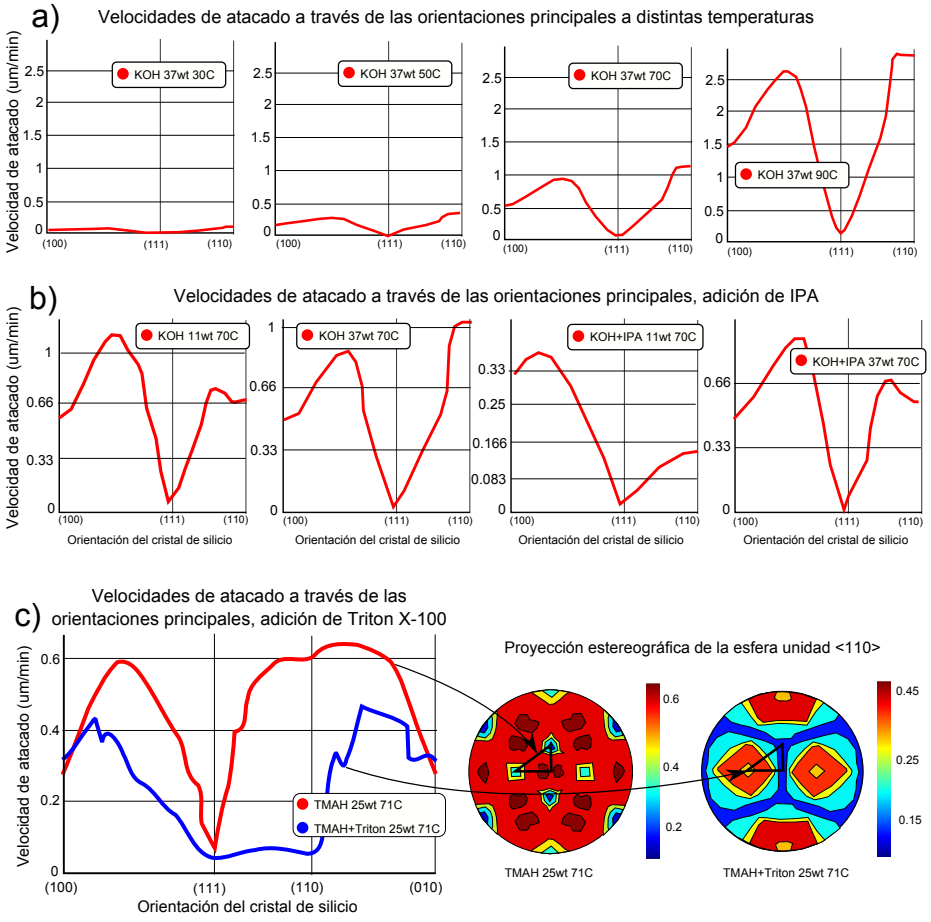

Figura 2.14: (a) Reactividad y anisotropía de un proceso de grabado con KOH 37 %wt a distintas temperaturas [106]. (b) Efectos de la adición de IPA a atacantes basados en KOH [107]. (c) Efectos de la adición de surfactante Triton X-100 a un atacante basado en TMAH [111].

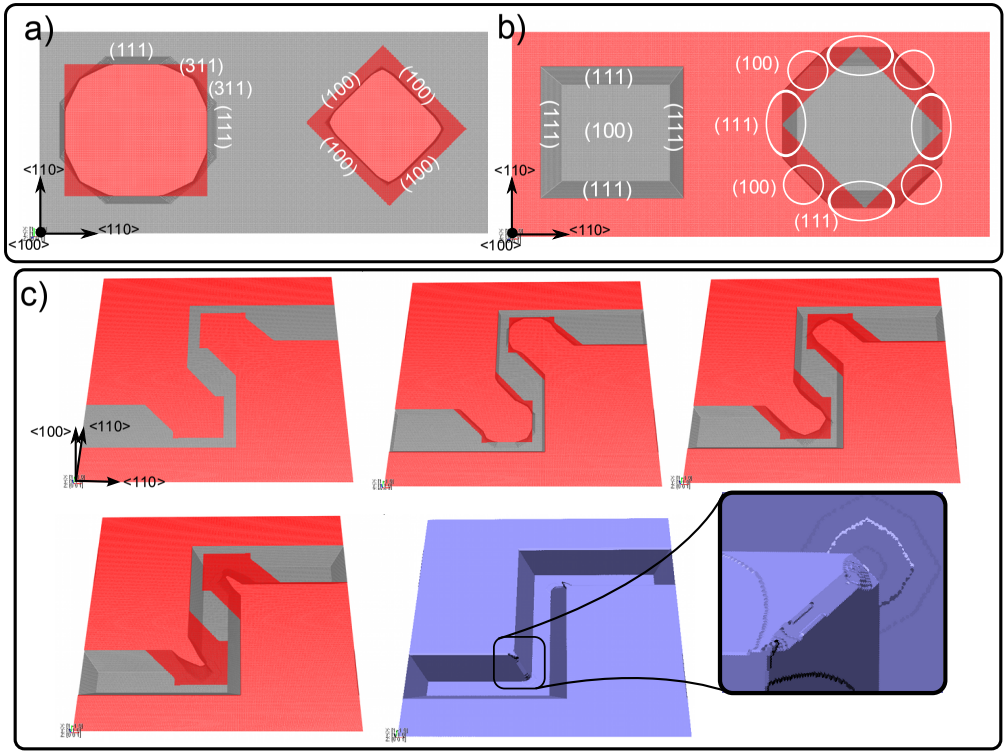

Figura 2.15: Morfología resultante del grabado de una superficie de silicio mediante KOH 40wt 70C [104] (a) Máscaras con esquinas cóncavas. (b) Máscaras con esquinas convexas. (c) Creación de un microcanal con ángulos de 90º utilizando compensación de esquinas.

Quizás el efecto más característico de este proceso de grabado es el llamado bajorecorte (en inglés *corner undercutting*), el cual consiste en el atacado rápido de esquinas convexas, especialmente cuando los bordes de la máscara están alineados con la dirección $< 100 >$ (figura 2.15 (a), izquierda). En este efecto aparecen habitualmente planos (311) [112], planos rápidos habitualmente en soluciones KOH que eliminan las esquinas. Dicho efecto dificulta la creación de estructuras rectangulares, y se resuelve aplicando compensaciones para las esquinas en las máscaras (figura 2.15 (c) izquierda), o utilizando aditivos en la solución.

Es, por lo tanto, la aparición de distintos planos lentos-rápidos los que han permitido obtener estructuras tridimensionales. En muchos casos, la forma resultante en función de la mascara aplicada no es evidente. Esto hace complejo el diseño de una microestructura mediante este proceso, teniendo que realizar muchos intentos hasta obtener el resultado adecuado.

Esta razón ha impulsado un notable esfuerzo en los últimos años por obtener modelos precisos de este proceso. De esta forma, un diseñador podría simularlo para obtener los parámetros adecuados antes de proceder a la etapa de fabricación. El siguiente apartado está dedicado a estos modelos.

2.3.4. Simulación del Grabado Anisótropo Húmedo

Los primeros modelos y simuladores que se desarrollaron para simular el grabado anisótropo eran los denominados *simuladores geométricos*, en los cuales los frentes de grabado eran descritos como una colección de planos propagándose a lo largo de sus normales de acuerdo con las velocidades de atacado indicadas [113, 114, 115, 116]. A pesar de que estos modelos eran capaces de describir adecuadamente el proceso de grabado, tienen la desventaja de requerir una base de datos completa de todas las orientaciones posibles para poder realizar adecuadamente las simulaciones, por lo que la calibración de dichos simuladores para nuevos parámetros del grabado es complicada. Asimismo, el coste de simulación de dichos modelos es especialmente alto en regiones donde varios planos se interceptan. Escenarios de simulación como la perforación de una oblea de silicio supone un gran problema en este tipo de simuladores.

Como alternativa a los simuladores geométricos, aparecieron modelos atomísticos basados en ACs. Estos modelos representan los átomos de silicio como células, teniendo cada átomo como vecino los cuatro átomos de silicio con los que comparte un enlace covalente. La retícula del AC, por lo tanto, emula la estructura monocristalina del silicio a escala microscópica. Los primeros modelos eran capaces de modelar las orientaciones principales. De acuerdo a la estructura cristalina del silicio, los átomos presentan una configuración de vecindad distinta en función de la orientación (figura 2.16). Esto daba la posibilidad de asignar una probabilidad de eliminación de los átomos en función de la orientación a la que pertenecía la superficie de silicio atacada [117, 118, 119]. Para obtener la configuración de cierto átomo se acudía a obtener la cantidad de primeros y segundos vecinos que poseían, siendo los primeros vecinos los átomos con los que tenían un enlace directo, mientras los segundos vecinos son aquellos átomos que están una posición mas allá, por lo que hay que acceder a ellos a través de un primer vecino (esté o no eliminado de la superficie). M. A. Gosalvez *et al.* formularon un método que relacionaba la configuración del átomo con su probabilidad de atacado a partir de consideraciones teóricas de la reacción química subyacente [120]. Este método conseguía simular de forma precisa grabados basados en KOH. Casi simultáneamente, Z. Zhu *et al.* formularon el proceso de atacado como la reducción de masa de los átomos de silicio [121]. En este modelo, denominado *Autómata Celular Continuo* (ACC), los átomos del substrato son marcados con una masa o ocupación inicial de valor 1.0. El proceso de atacado reduce la masa de los átomos en una cantidad concreta que modela de forma adecuada la velocidad de atacado del plano del que el átomo forma parte. Este nuevo punto de vista evita el ruido generado por el comportamiento estocástico de los modelos discretos.

De esta manera, en estos modelos atomísticos se diferencian de forma natural las dos escalas: la microscópica, la cual tiene en cuenta la estructura cristalina del silicio y donde el atacado consiste en la eliminación organizada de átomos, y la macroscópica, donde aparecen los distintos planos de atacado en función de la anisotropía del atacante. A partir de los primeros modelos descritos, el esfuerzo de investigación se centró en obtener una relación clara entre ambos niveles de realidad del modelo con el objetivo de obtener simulaciones más precisas.

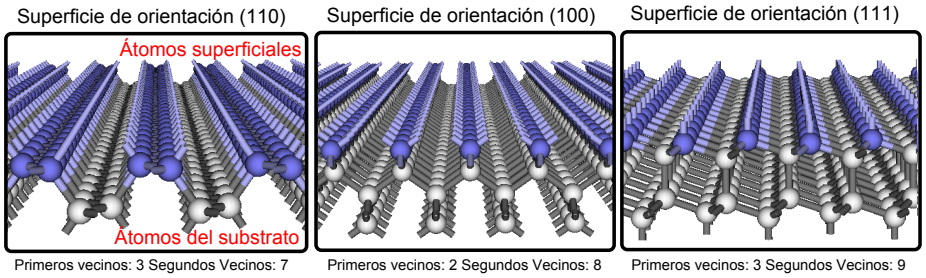

Figura 2.16: Morfología de la estructura del cristal del silicio para las orientaciones $< 100 >$,$< 110 >$ y $< 111 >$. Cantidad de primeros y segundos vecinos para los átomos superficiales indicados. Imágenes obtenidas con VisualTAPAS [122]

Fueron Z. Zhou *et al.* quienes propusieron una nueva definición de la configuración local de cada átomo basada en la clasificación de los átomos basado en los primeros vecinos superficiales (n_1^s), primeros vecinos del substrato (denominados también primeros vecinos enterrados o *bulk*) (n_1^b), segundos vecinos superficiales (n_2^s) y segundos vecinos del substrato (n_2^b), así como una relación entre las velocidades de decrecimiento de la ocupación a escala microscópica y las velocidades de atacado a escala macroscópica [123]. Esta nueva clasificación permitía una buena relación entre planos de más alto orden, (tales como (311)), y distintas configuraciones de vecindad.

Por último, M. A. Gosalvez *et al.* desarrollaron una nueva definición del ACC tomando en cuenta ciertos fenómenos físicos, lo que permitió la posibilidad de definir de forma precisa la velocidad de atacado de un amplio rango de orientaciones. Este modelo es explicado en detalle en la sección 2.3.4.1.

Finalmente, pese al extenso éxito de los modelos atomísticos recientemente, se han propuestos modelos de grabado basados en el método *level-set* [124], los cuales han obtenido resultados interesantes para el grabado basado en KOH [125] pero aún lejos de la precisión de las formulaciones más recientes del ACC.

2.3.4.1. Modelo Teórico del ACC Basado en *Step Flow*

M. A. Gosalvez *et al.* aplicaron en 2008 el concepto de propagación de escalones (en inglés *step-flow*) al modelado del grabado anisótropo mediante el ACC. El *step flow* formula la idea de que orientaciones de silicio de alto índice están formadas por una serie de terrazas de superficie (111) separadas por escalones, las cuales se van retrayendo debido a la sucesiva eliminación de los átomos que forman el escalón, menos ligados al cristal de silicio y por lo tanto mucho más fácil de ser eliminados (figura 2.17 (b)) [126]. El trabajo de Gosalvez consistió en dos tareas [127]:

1. Obtener una clasificación extensiva de todas las configuraciones de átomos que aparecían en todas las orientaciones existentes entre las tres principales, así como una clasificación y análisis detallado de la estructura morfológica a escala microscópica de dichas orientaciones (figura 2.17 (a)).

2. Interrelacionar el proceso de atacado a escala macroscópica de las distintas orientaciones de silicio con las velocidades de reducción de ocupación de todos los átomos obtenidos en la clasificación anteriormente enunciada (figura 2.17 (c)).

A partir de estas dos aportaciones, se obtuvo un sistema de ecuaciones en el cual, a partir de una cantidad limitada de orientaciones experimentales, es posible obtener la velocidad de reducción de ocupación de todos los átomos clasificados.

En este modelo, el proceso de grabado es definido como la reducción de una variable escalar, relacionada con la ocupación (Π), la cual es definida como estado interno de los sitios de superficie donde se alojan los átomos superficiales. Cuando un átomo del substrato pasa a ser átomo superficial, el sitio correspondiente es inicializado con el valor 1.0. A lo largo de la simulación, Π se reduce gradualmente hasta que alcanza el valor 0.0 (totalmente eliminado).

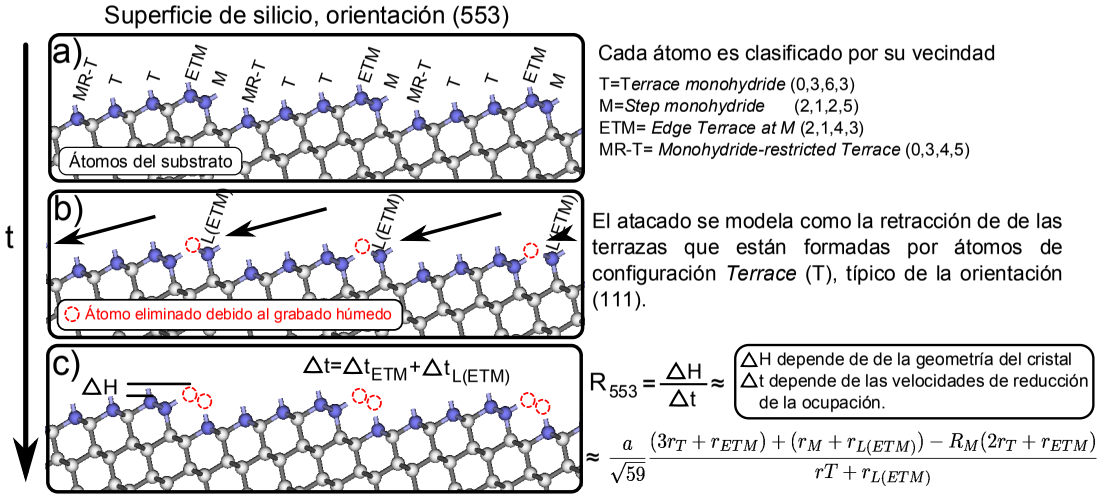

Figura 2.17: Proceso de eliminación de átomos de silicio para la orientación (553) mediante el ACC basado en *step-flow*. (a) Clasificación de los átomos superficiales en función de la cantidad y tipo de primeros y segundos vecinos. (b) Los átomos más débilmente enlazados al substrato, habitualmente en los bordes de las terrazas (es decir, en los escalones), son eliminados, retrayendo los escalones y propiciando el efecto de propagación de escalones *step-flow*. (c) Un segundo átomo se elimina y volvemos a la morfología inicial de la superficie. Este proceso se repite de forma cíclica eliminando sucesivamente átomos en el escalón. Es posible obtener el avance del plano (553) (escala macroscópica) a partir de la geometría del cristal. Asimismo, el tiempo transcurrido en este ciclo se puede relacionar con las velocidades de reducción de ocupación de los distintos tipos de átomos que han aparecido en el proceso (escala microscópica) [127]. Relacionando ambos conceptos podemos obtener las velocidad de los planos como función de las velocidades de atacado de los átomos. Imágenes obtenidas mediante VisualTAPAS [122].

En cada paso de tiempo (k), la reducción en la ocupación del sitio de superficie i equivale a la velocidad de extracción del átomo que puebla dicho sitio (r_i), multiplicado por el paso de tiempo (t).

$$\begin{aligned} \Pi_i(k+1) &= \Pi_i(k) - \Delta t \cdot r_i(k), \\ \text{donde} \quad r_i(k) &= R(n_i^{1s}(k), n_i^{1b}(k), n_i^{2s}(k), n_i^{2b}(k)). \end{aligned} \tag{2.6}$$

En esta ecuación, la tupla $(n_i^{1s}(k), n_i^{1b}(k), n_i^{2s}(k), n_i^{2b}(k))$ representa la configuración de la vecindad del átomo de acuerdo a la caracterización de primeros y segundos vecinos.

Las velocidades de extracción r_i dependen del número de vecinos como se indica en la segunda línea de la ecuación 2.6. En este caso, R es una función que depende del atacante simulado (KOH, TMAH, ect...), incluyendo su concentración y temperatura. Debido a que n_i^{1s} y n_i^{1b} pueden tomar valores de 0 a 3 y n_i^{2s} y n_i^{2b} toma valores de 0 a 11 para átomos superficiales, R puede ser vista como una tabla de 4x4x12x12 entradas. Muchas de las entradas de la tabla carecen de sentido físico y habitualmente basta retener el conjunto de 33 velocidades de extracción obtenido de la resolución del sistema de ecuaciones [127]. Hasta que un átomo superficial se elimine completamente, su velocidad de extracción continua cambiando a medida que su vecindad es modificada. Por esta razón, r_i, n_i^{1s} ... son función de k.

Este modelo ha permitido la simulación precisa del grabado del silicio incluyendo orientaciones de alto índice para una variedad atacantes, incluyendo KOH, TMAH y KOH+IPA (figura 2.18). A día de hoy, el modelo propuesto por Gosalvez *et al.* es reconocido como el más exacto para la simulación del proceso de grabado anisótropo. Dicho CCA ha sido el que mejor resultados ha presentado para atacado de planos de indice alto [127], y el único que ha conseguido simular el uso de aditivos [128].

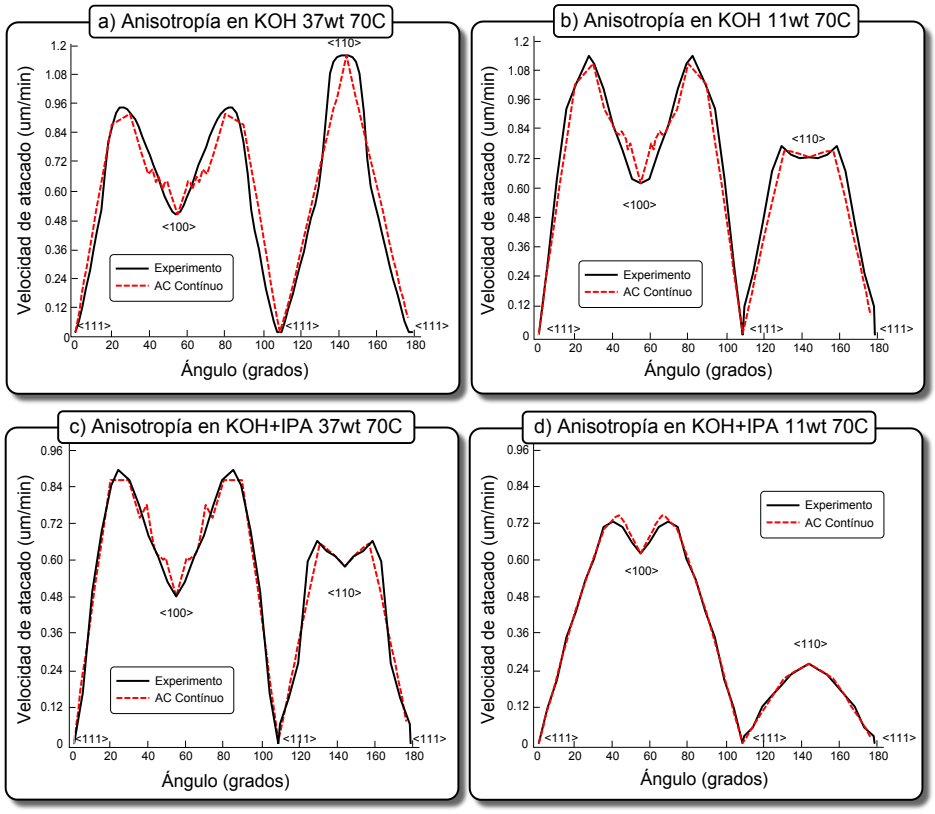

Figura 2.18: Velocidades de atacado a través de las tres orientaciones principales para datos experimentales [107] y simulación con el ACC definido por Gosalvez *et al.* calibrado con dichos datos.

Capítulo 3

Modelado de Sistemas Mediante ACs

Previo a los principales resultados presentados en esta tesis, que serán expuestos en capítulos posteriores, han sido realizadas tareas de investigación relacionadas con el modelado basado en AC, así como facilitar el proceso de implementación y simulación de ACs sobre FPGAs. En este capítulo se introducen brevemente dichas tareas, así como los resultados obtenidos.

3.1. Cellular Structure Description Language: un Lenguaje de Alto Nivel para la Implementación de ACs en FPGAs

Pese a que la utilización de FPGAs supuso un adelanto sobre la implementación de ASICs con el fin de acelerar la simulación de ACs debido a su naturaleza programable, su versatilidad queda todavía por debajo de la utilización de un lenguaje de programación de alto nivel: el diseño de hardware basado en FPGAs sigue siendo una tarea compleja en la que los distintos elementos físicos del dispositivo deben estar siempre presentes en la mente del diseñador.

Esta falta de versatilidad ha llevado al desarrollo de algoritmos para la automatización del proceso de implementación de ACs sobre FPGAs [129]. Pese a la existencia de estas metodologías, la definición de un lenguaje de alto nivel orientado específicamente a la implementación de sistemas celulares sobre FPGAs puede dar un nivel de versatilidad mayor, de forma que un gran rango de ACs pudieran ser definidos de forma simple y directa. En esta sección presentamos un lenguaje desarrollado para cumplir esta función. Este lenguaje se ha denominado *Lenguaje de Descripción de Estructuras Celulares* (CSDL, del inglés *Cellular Structure Description Language*). La definición de CSDL junto con con la creación de un compilador que genera código VHDL sintetizable a partir de la descripción, así como la creación de una herramienta de desarrollo gráfica basada en CSDL ha permitido facilitar el proceso de implementación de ACs sobre FPGAs.

3.1.1. Aspectos Generales de una Descripción CSDL

Una descripción CSDL se compone de tres partes fundamentales:

1. *Definiciones de los tipos de célula.* Cada célula es modelada como unidad lógica, con sus entradas y salidas. Es posible definir tantos tipos como se desee.

2. *Definición de redes celulares.* Las redes celulares se forman con las células creadas en la primera sección. De la misma manera que con las células, es posible definir tantas redes celulares como sea necesario.

3. *Definición una capa de recursos (opcional).* En la capa de recursos es posible interconectar todas las redes celulares que hayamos definido anteriormente, además de definir recursos globales que se pueden conectar a una o varias redes, como contadores o entradas y salidas serie.

La descripción se realiza dentro de un único fichero, que será luego procesado por el compilador. En los siguientes apartados veremos pormenorizadamente cómo se define cada sección de la descripción.

3.1.1.1. Definición de Células

La definición de una célula en CSDL es un proceso bien definido y secuencial el cual se divide en cuatro pasos:

- Definición de entradas de la célula.
- Definición de salidas de la célula.
- Definición de los elementos lógicos internos.
- Interconexión de todos los componentes

Las entradas y salidas se definen especificando nombre y tamaño en bits. Los componentes internos que CSDL permite definir para las células son:

- Registros, con entradas de *enable*, *clear*, carga y dato de carga.
- Operaciones booleanas.
- Operaciones aritméticas básicas, tales como suma, resta y multiplicación.
- Declaraciones IF-ELSE.
- Tablas de verdad.
- Reglas de evolución con la notación de Wolfram [10].
- Agrupación de datos.
- Multiplexores.

Cada componente interno queda definido por un nombre y el tamaño en bits de la señal sobre la que opera.

Asimismo, el proceso de interconexión de la entrada de un elemento con la salida de otro se realiza simplemente utilizando el operador <=. Es posible concatenar diversas entradas o salidas mediante el operador &. Mostramos a en la figura 3.1 la célula utilizada en el AC *el juego de la vida* (sección 2.1.2.2) descrita en CSDL.

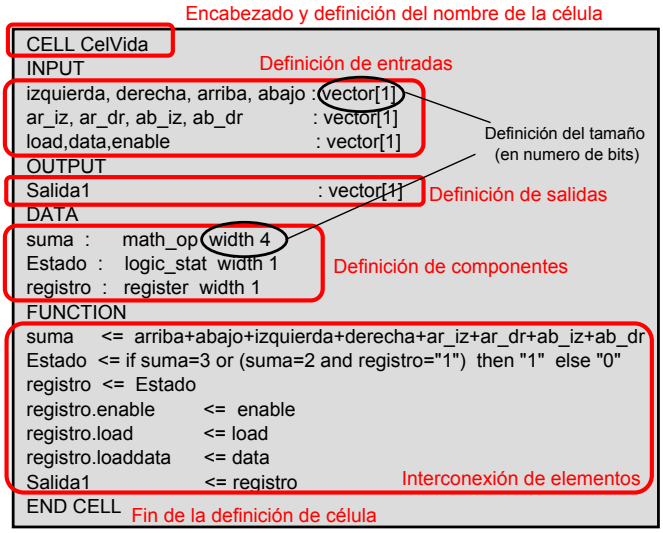

Figura 3.1: Definición de la célula del AC *juego de la vida* en CSDL.

3.1.1.2. Definición de la Red Celular

En un AC, las células se agrupan en redes regulares de una morfología definida. Una definición CSDL puede albergar una o varias redes celulares. Las características de una red que es necesario especificar a la hora de definir una red son las siguientes:

- **Disposición geométrica de las células.** CSDL admite cuatro estructuras de dichas redes, una unidimensional y tres bidimensionales. La figura 3.2 muestra gráficamente la disposición espacial de las células en cada una de las estructuras.

- **Células que componen la red.** La red puede estar formada por uno o varios tipos de células. En el segundo caso esto se realiza definiendo un bloque de células el cual es replicado hasta conseguir el tamaño de red deseado.

- **Tamaño de la red.** Se define indicando la cantidad de bloques en

cada uno de los ejes (X,Y).

- **Entradas y salidas de la red.** Las entradas y salidas definidas en las células pueden ser utilizadas para interconexión interna entre ellas o para leer/escribir datos desde el exterior, en este campo se indica cuales pertenecen al segundo grupo. Asimismo es posible especificar una entrada como *global*. Esto significa que la misma señal debe ser dirigida a todas las células que la posean. En caso de no definir una entrada como global, cada célula poseerá su propia línea de entrada.

- **Interconexión entre las células.** En este apartado es posible asignar entradas de las células a las salidas de otras células. Dicha vecindad se replica para todas las células del mismo tipo y no existe ninguna restricción en la posición espacial relativa de la vecindad. Dicha posición es definida como un camino. Cada camino es una sucesión de números separado por puntos. Cada número nos lleva a una célula adyacente a la que estamos. Estos números dependen de la disposición geométrica. La figura 3.2 muestra el número asignado a cada posición relativa de las células adyacentes para todas las morfologías.

- **Bordes de la red.** En CSDL es posible definir tres condiciones de contorno para las interconexiones que caen fuera del dominio de la red: red cíclica, la extracción de los bordes como señales externas de la red o valores constantes definidos como un número binario de longitud igual al tamaño de la señal.

La figura 3.3 muestra la definición de la red de tamaño 25x25 la cual implementa el AC *el juego de la vida* en CSDL.

3.1.1.3. Definición de la Capa de Recursos

De forma opcional, es posible definir una capa de recursos donde se pueden definir elementos globales tales como contadores. Asimismo, es posible interconectar distintas redes, crear entradas-salidas serie o definir una interfaz de conexión con el procesador *softcore* Nios II de Altera [130]. El

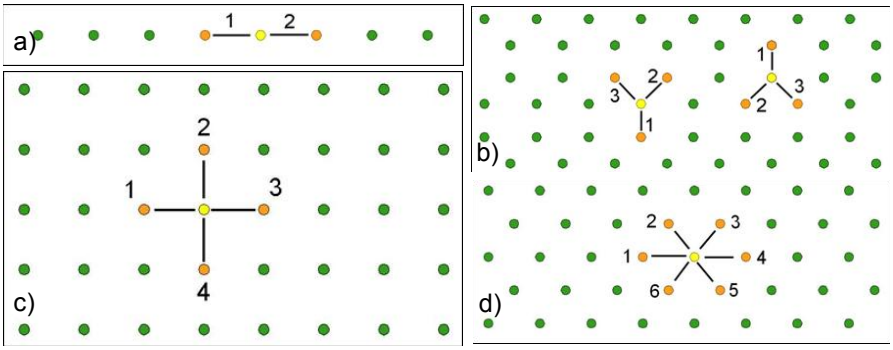

Figura 3.2: Morfologías posibles en las redes definidas en CSDL. Las células naranjas son las adyacentes de la amarilla. (a) 2 adyacentes. (b) 3 adyacentes. (c) 4 adyacentes. (d) 6 adyacentes.

objetivo de esta última funcionalidad es la de utilizar el AC como un coprocesador especializado conectado a un procesador secuencial. Nios II ofrece una mayor versatilidad a la hora de leer o escribir datos en el AC, asimismo posee numerosas interfaces de comunicación prediseñadas que pueden ser útiles a la hora de transferir los datos relacionados con el AC a otro sistema.

3.1.2. Compilador CSDL y Glider

Junto a la definición de CSDL, se ha desarrollado un compilador en modo texto que porta una descripción de un AC a un conjunto de ficheros los cuales poseen código VHDL sintetizable para su implementación sobre una FPGA. Dicho compilador está realizado en Java. Al iniciar el proceso de compilación, la aplicación realiza una serie de operaciones para encontrar errores en la descripción:

- **Errores de estructura.** Comprueba la morfología general del fichero fuente, como el adecuado uso de palabras reservadas de encabezado y fin de células y redes.

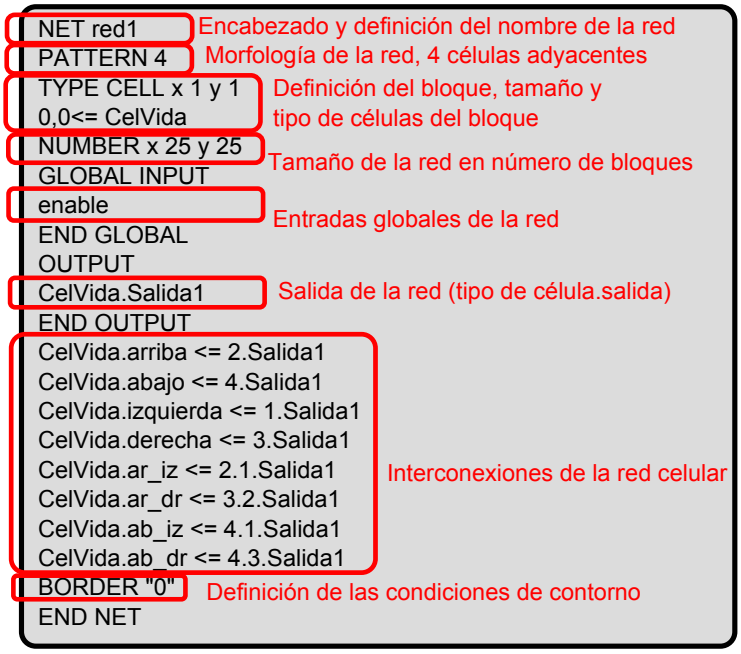

Figura 3.3: Código CSDL para la descripción de una red celular del AC juego de la vida de tamaño 25x25 células. La célula celVida, definida previamente, está siendo utilizada.

- **Errores de sintaxis.** Comprueba errores de sintaxis en la declaración de células o redes.
- **Interconexión intracelular.** Se comprueba que todos los elementos declarados en las células se utilizan y sus conexiones son correctas.
- **Errores de sintaxis e interconexión de la capa de recursos.** Se comprueban errores de sintaxis o de interconexión de elementos en la capa de recursos, si se ha definido.

La figura 3.4 (a) muestra los mensajes del compilador cuando el proceso ha sido correcto. En la figura 3.4 (b), la entrada *enableq* no ha sido definida, por lo que el compilador advierte sobre el error especificando la causa y su localización.

Una vez comprobada la corrección de la descripción se generan los siguientes ficheros VHDL:

- Un *Package* donde se definen tipos matriciales que se usan en los siguientes ficheros:
- Un fichero VHDL para cada tipo de célula.
- Un fichero VHDL para cada red celular.
- Un Fichero VHDL para la capa de recursos en caso de que haya sido definida.

Asimismo, junto a CSDL se ha desarrollado *Glider*, una herramienta de diseño gráfica la cual utiliza CSDL como lenguaje interno. Todas las posibilidades de una descripción CSDL están disponibles en *Glider*, pero definidas de una forma visual. *Glider* presenta un entorno amigable, de manera que es bastante más cómodo que escribir código CSDL y compilarlo posteriormente. La figura 3.5 muestra una captura de pantalla del menú de definición de red celular en *Glider*.

Información detallada sobre la sintaxis de CSDL puede ser encontrada en [131]. Asimismo, la aplicación *Glider* está disponible para su descarga en [132].

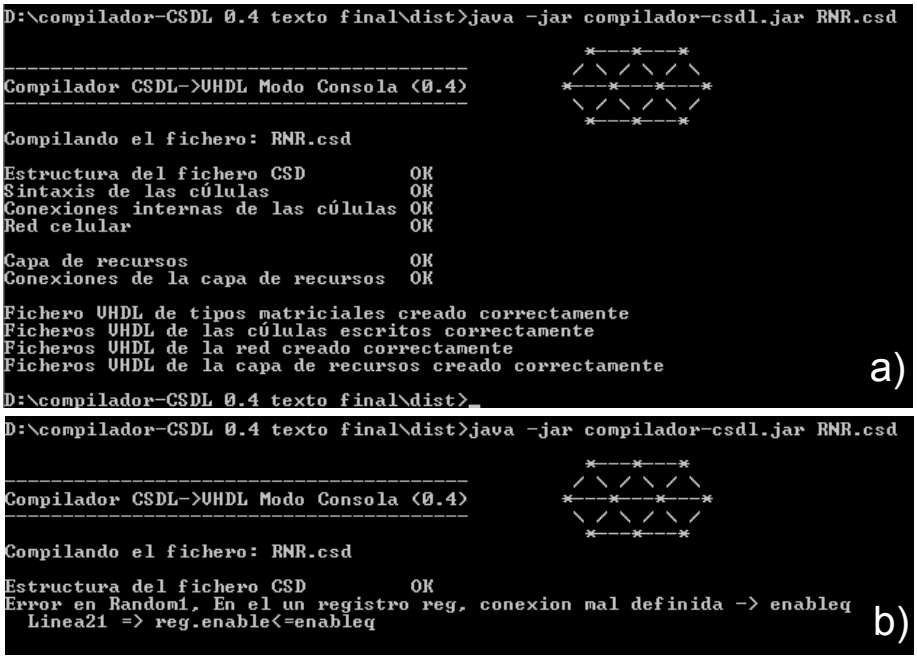

Figura 3.4: Ejecución del compilador CSDL. (a) Resultado de compilación satisfactorio. (b) Detección de un error de sintaxis en el código fuente.

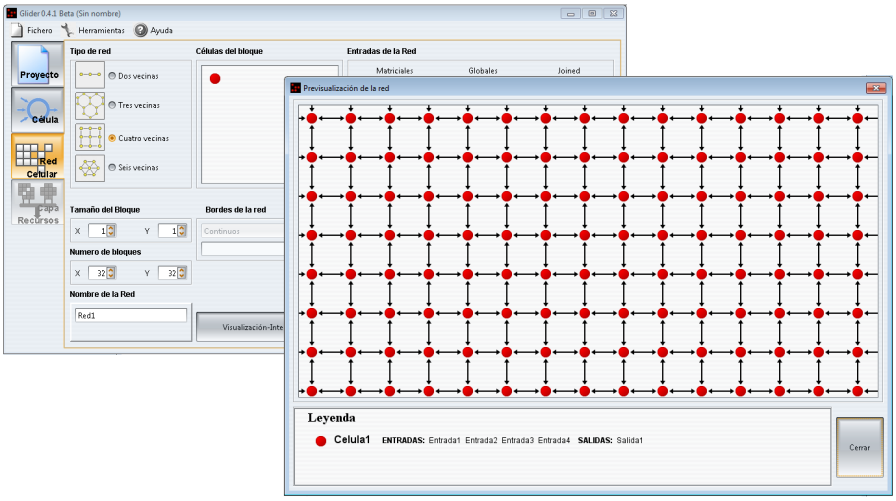

Figura 3.5: Captura de pantalla de la aplicación Glider.

3.1.3. Reflexiones Sobre la Utilización de FPGAs para la Aceleración de Modelos Basados en ACs.

Tanto las FPGAs como las GPUs se plantean como las dos plataformas más generalistas para el procesado eficiente de ACs. Existe trabajo previo donde se ha demostrado la buena adaptación de estas dos arquitecturas a sistemas masivamente paralelos como los ACs (secciones 2.1.3.1 y 2.2.3). Sin embargo, examinadas en detalle, ambas plataformas poseen características muy distintas.

En el aspecto de versatilidad de programación, pese a la introducción de lenguajes como CSDL, la programación en GPUs posee un mayor nivel de abstracción sobre el hardware, por lo que el mismo algoritmo resulta menos complejo programarlo en una GPU mediante CUDA C o openCL que intentar utilizar CSDL o programar directamente el AC en hardware mediante VHDL o Verilog.

Atendiendo a la potencia aritmética teórica que cada plataforma es capaz de desarrollar, este aspecto es directamente dependiente del formato de

datos con el que se quiera trabajar. Las unidades aritméticas de las GPUs actuales son de 32 bits (64 en la nueva arquitectura Fermi) y, por lo tanto, no obtenemos ninguna mejora al trabajar con formatos de datos menores. El hecho de trabajar directamente con hardware en las FPGAs permite adaptarse mejor al problema, por lo que para un AC que utilice datos de pocos bits, es posible implementar más células en paralelo, incrementando de forma drástica la cantidad de estados procesados por segundo. Por otro lado, trabajar con datos de coma flotante es extremadamente costoso a la hora de ser implementado en una FPGA y, sin embargo, este hecho es asumido de forma natural por parte de una GPU, que ofrece el mismo rendimiento para este tipo de operaciones.

Debido al intensivo patrón de acceso de memoria de los ACs, la velocidad de la memoria se convierte en un aspecto primordial a la hora de poder evaluar la eficiencia de una implementación. En caso de que se desee simular sistemas de miles o millones de células, una gran cantidad de memoria externa al núcleo de alta velocidad es necesaria, también son necesarios mecanismos para poder acceder a dicha memoria de forma eficiente. En este caso, las GPUs vendidas actualmente en el mercado ya poseen memorias de muy altas velocidades [58], con anchos de banda superiores a los que son posibles de implementar actualmente en FPGAs [133, 134]. Igualmente, las GPUs poseen una filosofía de funcionamiento que permite obtener un acceso eficiente a la memoria por parte de algoritmos masivamente paralelos.

En conclusión, la simulación de alta velocidad de ACs sobre FPGAs es indicada en sistemas empotrados o en dispositivos integrados los cuales necesitan funcionar en tiempo real. También puede resultar recomendable cuando la morfología del AC permita una alta inclusión de elementos de proceso en la FPGA, por ejemplo si se manejan tipos de datos de 1 o unos pocos bits.

En el resto de casos, las GPUs se muestran como una opción más versátil y más potente si se desea trabajar con datos como decimales de coma flotante o colecciones de millones de átomos.

3.2. Modelado con Autómatas Celulares: Ecualización de Ganancias en un *Front-end* Analógico para Aplicaciones PET

En esta sección, la técnica de modelado basada en ACs es utilizada para obtener un modelo y predecir parámetros que optimicen diversas características. El campo de aplicación es la ecualización de ganancias para determinada electrónica utilizada en sistemas de imagen médica basada en Tomografía por Emisión de Positrones (PET, del inglés *Positron Emission Tomography*). El fin de esta ecualización es la de mejorar la calidad de dicha imagen. Debido a la naturaleza de determinados detectores, la ecualización de las ganancias puede volverse un problema altamente acoplado (actuar sobre cierta ganancia afecta a todas las ganancias de posiciones vecinas), por lo que resulta difícil atacarlo de forma global. Es aquí donde la filosofía de modelado de los ACs puede resultar una herramienta útil.

3.2.1. Detectores Indirectos para Sistemas PET y Electrónica de *Front-end*

Para poder realizar un correcto diagnóstico de una enfermedad, el obtener toda la información posible acerca del paciente es una necesidad evidente. El campo de la imagen médica es la técnica y el proceso utilizado para crear imágenes del cuerpo humano con objetivos médicos tales como buscar, examinar o diagnosticar una enfermedad. Una vez obtenida una imagen del cuerpo de un paciente, dicha imagen puede darnos información estructural, pero también funcional. Métodos para ver la estructura del cuerpo pueden ser las radiografías, así como resonancias magnéticas o tomografías computerizada por rayos X. Sin embargo, sistemas PET o sistemas basados en Tomografía Computerizada por Emisión de Fotones Individuales (SPECT, del inglés *Single Photon Emission Computed Tomography*)) , dan la posibilidad de obtener información de la actividad metabólica del organismo. Esta funcionalidad es extremadamente útil en campos como la oncología (detección y seguimiento de tumores) o neurología (detección de fibrosis o necrosis).

Los detectores PET son utilizados en conjunto con radiofármacos que son inyectados al paciente. Estos fármacos están marcados con isótopos emisores de positrones, lo cual permite trazar la concentración del fármaco dentro del cuerpo del paciente a lo largo del tiempo. Cuando un positrón emitido por el isótopo se encuentra con un electrón (su antipartícula), se produce una aniquilación de ambas partículas, generando de forma simultánea dos fotones gamma que se desplazan en sentido contrario [135]:

$$e^+ + e^- \rightarrow \gamma + \gamma \tag{3.1}$$

Son dichos fotones los que son detectados y, a partir de un proceso de reconstrucción de imagen, es posible obtener imágenes de la evolución de concentración de radiofármaco que evidencien procesos fisiológicos del cuerpo del paciente.

La idea de la utilización de la radiación proveniente de la aniquilación positrón-electrón para fines médicos proviene de los años 50. A partir de entonces sucesivas generaciones de sistemas PET han aparecido, mejorando en cada nueva generación la resolución y eficiencia del sistema.

Estructuralmente, un sistema PET está compuesto por un anillo de detectores de fotones gamma (figura 3.6 (a)). Cuando dos fotones gamma, inciden en el anillo de forma simultánea se denomina *coincidencia*. Ambas detecciones se presuponen provenientes de la misma aniquilación, por lo que, a partir de la posición donde se han detectado los eventos, es posible reconstruir la posición donde la aniquilación ha ocurrido. La figura 3.6 (b) muestra una reconstrucción típica de un PET cerebral, el cual se utiliza habitualmente para medir la actividad del cerebro.

3.2.1.1. Detectores Indirectos Basados en Cristales Centelleadores y Fotomultiplicadores

Los detectores indirectos de positrones están basados en la detección de los fotones gamma. Los métodos tradicionales de detección se basan en la utilización de cristales centelleadores. Estos cristales tienen la cualidad de producir un destello cuando son excitados por radiación ionizante, cu-

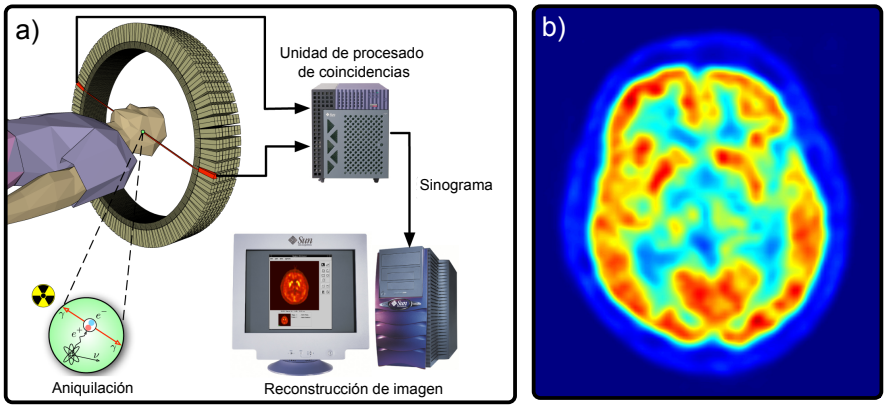

Figura 3.6: (a): Esquema de un sistema PET. (b): Imagen capturada en un PET cerebral

ya intensidad es habitualmente proporcional a la energía depositada en el cristal. Para aplicaciones PET, ciertas características son deseables: una alta eficiencia de detección, tiempos de caída cortos para incrementar la velocidad de detección de eventos, una buena resolución energética para identificar adecuadamente los eventos que han depositado toda la energía, así como una buena alta intensidad de luz de salida y una longitud de onda adecuada a las características del fotomultiplicador. Materiales utilizados habitualmente en estos cristales para la utilización en sistemas PET son: ortosilicato de lutecio (LSO), ortosilicato de lutecio-itrio (LYSO), germanato de bismuto (BSO) o ioduro de sodio dopado con talio NaI(Tl) [136].

Dentro del campo de aplicaciones PET, una de las características más importantes de los cristales es la *pixelización* o no de los mismos. Los cristales pixelados están divididos en muchas secciones, por lo que, cuando ocurre un centelleo, la dispersión de luz queda confinada en dicho píxel. Este tipo de cristales, pese a que posee la ventaja de fijar la resolución espacial, encarece en gran medida el coste de fabricación. La utilización de cristales continuos como alternativa a la pixelización de cristales obtiene buenos resultados en los sistemas PET [137]. En este trabajo nos centramos en los detectores basados en cristales continuos.

La siguiente etapa del proceso consiste en la detección de la luminiscencia producida por el cristal mediante un tubo fotomultiplicador (PMT), cuya funcionalidad es la de transformar una señal lumínica en señal eléctrica de una intensidad proporcional. Un tubo fotomultiplicador consiste en un tubo de vacío el cual posee en su interior varios componentes:

1. Un fotocátodo que, debido al efecto fotoeléctrico, al ser iluminado emite fotoelectrones que son enviados hacia el multiplicador de electrones.

2. Un multiplicador de electrones, consistente en un conjunto de dínodos. Estos dínodos son electrodos recubiertos de materiales emisores secundarios de electrones. Cada dínodo está a un potencial mayor que el anterior. En cada dínodo, la incidencia de electrones crea una emisión mayor de electrones que son acelerados hacia el siguiente dínodo. El objetivo de esta etapa es la de actuar como amplificador con el fin de poder facilitar la detección de la corriente generada por el fotocátodo.

3. Un ánodo, el cual recoge los electrones de la anterior etapa y los dirige hacia un circuito sensor externo.

Para las aplicaciones PET son utilizados tubos fotomultiplicadores sensibles a posición. Estos dispositivos poseen una matriz de ánodos, donde la intensidad que circula a través de ellos depende de la posición en la que el fotocátodo ha generado fotoelectrones. Los fotomultiplicadores sensibles a posición ofrecen la posibilidad de detectar con buena resolución espacial la posición de un centelleo proveniente de un cristal. Un ejemplo típico de estos sensores es el Hamamatsu H8500 [138] el cual ofrece un área efectiva de detección de 49mmx49mm y posee una rejilla de 8x8 ánodos.

En condiciones normales, los tubos fotomultiplicadores muestran buena linealidad en su respuesta [139]. Pese a la buena linealidad, los PMT multiánodo sufren de una intensa desviación en la ganancia de sus distintos ánodos [140], pudiendo causar deformaciones en la imagen obtenida en el sistema PET.

La última etapa consiste en obtener datos sobre la localización espacial del centelleo a partir de la carga emitida por los ánodos. El método más utilizado para cristales continuos es la obtención del centro de masas de la distribución de luz mediante la lógica de Anger [141], en la cual la posición es obtenida a partir de una implementación de las siguientes funciones:

$$X = \frac{\sum_i x_i \cdot S_i}{\sum_i S_i}; Y = \frac{\sum_i y_i \cdot S_i}{\sum_i S_i} \tag{3.2}$$

Donde i se refiere al ánodo i, (x,y) a la posición espacial de dicho ánodo y S a la cantidad total de carga emitida por el ánodo.

3.2.1.2. PESIC: *Front-end* Analógico Integrado

Las ventajas de la integración de la electrónica de *front-end* para la lectura de las intensidades emitidas por un PMT multiánodo sobre la utilización electrónica discreta son: La mejora de la velocidad de respuesta respecto a la electrónica discreta debido a la reducción de efectos tales como las capacidades parásitas y la posibilidad de ecualización de las entradas mediante un conjunto de preamplificadores de ganancia programable.

Nuestro trabajo se centra sobre PESIC [142], un ASIC el cual integra los siguientes elementos:

1. Una matriz de 8x8 amplificadores de intensidad de ganancia programable digitalmente,
2. Los amplificadores de la primera etapa están conectados a una red de resistencias proporcional a posición, la cual permite detectar la posición (X,Y) del centroide del centelleo a partir de las cuatro intensidades obtenidas de las salidas de la red (A, B, C, D) [143].
3. Las salidas (A, B, C, D) de la red de resistencias son convertidas a tensión mediante una última etapa de amplificadores de transresistencia.

Asimismo, PESIC posee electrónica adicional para la detección de la profundidad de interacción del centelleo.

3.2.2. Modelo Teórico: AC para Ecualización de Ganancias

La aparición de diferencias en las ganancias de los distintos ánodos del fotomultiplicador, así como diferencias entre las ganancias de los distintos amplificadores internos de PESIC debido a los procesos de microfabricación del ASIC, pueden afectar al resultado final de la imagen. Existe trabajo previo en relación a la calibración de las ganancias de los detectores para PET [144, 145], pero dichos métodos trabajan sobre cristales pixelados, facilitando este hecho en gran medida la calibración. Cuando se analiza la distribución de luz de un centelleo ocurrido en un cristal continuo, múltiples ánodos y múltiples amplificadores afectan a este proceso. Por tanto, el problema de calibración de ganancias está altamente acoplado debido a que la ganancia de cada ánodo afecta a todas las distribuciones de luz cercanas. Lo mismo sucede con las variaciones de ganancias de los coeficientes programables.

La discretización evidente del espacio, la dependencia espacial del resultado final de un centelleo, no solo del ánodo en su posición, sino también de los ánodos cercanos (vecinos), así como la complejidad de obtener alguna metodología de calibración a escala global (macroscópica), es una clara motivación para intentar definir el detector como un AC, intentando modelar la inter-relación de los elementos del sistema a escala microscópica.

En este caso, la magnitud macroscópica que deseamos medir y ajustar es la homogeneidad de la energía medida de un centelleo a lo largo del detector. A escala microscópica, cada célula representará una región 2D del espacio correspondiente a un ánodo.

La figura 3.7 (a) muestra un modelo 1-D de la energía del centelleo leída del ASIC(obtenida sumando la carga proveniente de la red de resistencias $(A + B + C + D)$), donde i es una determinada posición discretizada del espacio, la cual corresponde a un ánodo del PMT. K_i representa la ganancia relativa de cada amplificador debido a efectos de fabricación y M_i la ganancia programable.

El siguiente paso es substituir el centelleo por la distribución de luz que incide sobre el fotocátodo. Ignorando efectos como la propia absorción de luz debido al cristal o la existencia de una pequeña cantidad de luz

constante, la distribución de intensidad luz sobre el fotocátodo viene dada por la siguiente ecuación [146]:

$$L = \frac{\mathit{effd}}{\sqrt{((x - x_0)^2 + (y_0 - y)^2 + \mathit{effd}^2)^3}}; \tag{3.3}$$

En esta ecuación *effd* es la distancia entre el origen del centelleo y el fotocátodo (profundidad de interacción), (x_0, y_0) la posición en el plano formado por el fotocátodo donde ha sucedido el centelleo y (x, y) el punto donde deseamos obtener la intensidad de luz.

El detector al cual estamos aplicando el modelado, posee un cristal centelleador LYSO con un grosor de 1 cm. Analizando las tablas de eficiencia de absorción del cristal [147], escogemos 7 mm como profundidad media donde se producirá el evento. Asimismo, el fotomultiplicador utilizado por nuestro sistema detector es el modelo H8500 de Hamamatsu. Suponiendo un centelleo a 7 mm sobre el centro de un ánodo e integrando a lo largo del área cubierta por cada ánodo cercano, obtenemos la distribución de luz detectada para los píxeles del detector cercanos al punto del centelleo, el cual está situado centrado sobre la posición central de la matriz (figura 3.7 (b)).

Para el centelleo en cierta posición espacial, el efecto de la distribución de luz sobre el PMT puede ser implementado en el modelo como un conjunto de ganancias obtenido de la figura 3.7 (b) aplicada en la etapa previa al fotomultiplicador. De esta forma podemos modelar, para un centelleo ocurrido sobre cierta posición discreta del espacio (x, y), la energía del centelleo detectada por PESIC tal y como muestra la figura 3.7 (c). A_i describe la ganancia del ánodo del fotomultiplicador correspondiente a la posición espacial i y Out_i corresponde a la energía detectada por PESIC para un centelleo sucedido en la posición espacial i . La ecuación de la energía detectada por PESIC cuando un centelleo sucede sobre la posición (i, j) es:

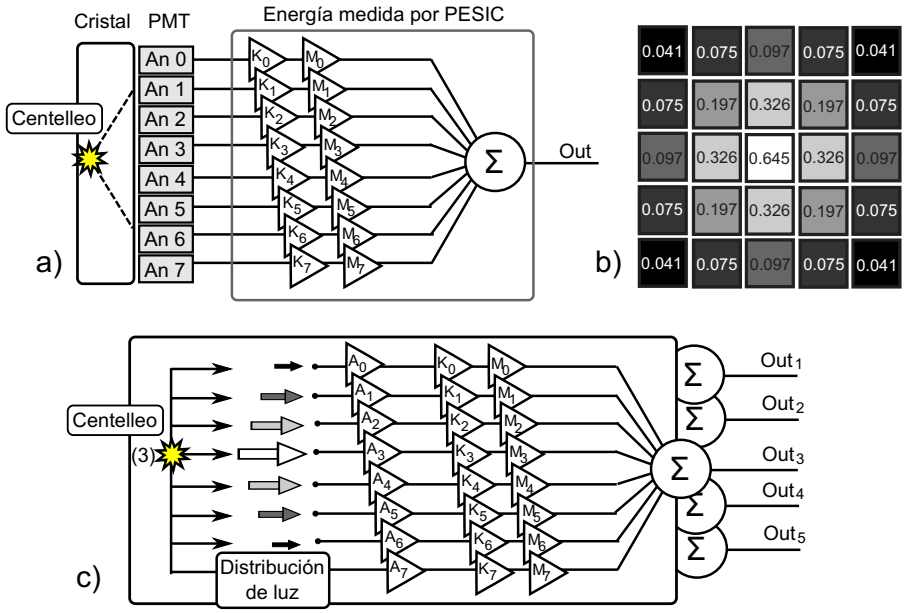

Figura 3.7: (a) Esquema de obtención de la energía leída del PESIC *Out* a partir de un centelleo. (b) Distribución de la cantidad de luz recogida por los ánodos para el cristal utilizado (LYSO de 1 cm de grosor). (c) Simplificación 1-D del detector. Aplicación de la distribución de luz al modelo presentado en (a). Out_i modela la energía detectada para un centelleo sobre la posición i.

$$Out_{i,j} = Cent \cdot \sum_{a=-2}^{2} \sum_{b=-2}^{2} (L_{(a,b)} \cdot A_{(i+a,j+b)} \cdot K_{(i+a,j+b)} \cdot M_{(i+a,j+b)}); \tag{3.4}$$

En esta ecuación *Cent* es un coeficiente que indica la cantidad de luz emitida por el centelleo y $L_{(a,b)}$ es la intensidad de luz detectada por el ánodo en la posición relativa (a,b) (figura 3.7 (b)). Debido a la distribución de luz, los distintos parámetros del modelo afectan conjuntamente a

la salida, lo que evidencia que el problema es altamente acoplado: la modificación de la ganancia de un ánodo en PESIC afecta a los resultados de todas las posiciones cercanas.

Antes de poder utilizar este modelo, es necesaria la caracterización de las ganancias relativas del fotomultiplicador y de los amplificadores de PESIC. Desde el punto de vista experimental la prueba más sencilla que podemos efectuar en nuestro banco experimental consiste en colocar una fuente puntual emisora de fotones gamma sobre cada posición discretizada del detector y leer el valor de energía obtenido por PESIC. Los coeficientes programables de PESIC se mantendrán a un valor constante. La salida se verá afectada por los efectos conjuntos de las ganancias de los ánodos y los amplificadores en cantidades que dependen de la intensidad de luz que recoge cada ánodo. Sin embargo, para poder comparar estos datos experimentales de forma directa con las ganancias del modelo, es necesario reformularlo.

La figura 3.8 (a) muestra una readaptación del modelo, donde el resultado de la salida para cada posición es equivalente (ecuación 3.4). En esta ecuación, para cada posición (i, j) del modelo, existe una combinación conocida de ganancias no variables (A y K) que afecta de forma única al conjunto de parámetros de entrada del modelo.

Para cada $Out_{i,j}$, definimos un único coeficiente $C_{i,j}$ tal que:

$$\begin{aligned}\sum_{a=-2}^{2}\sum_{b=-2}^{2}(L_{(a,b)}\cdot A_{(i+a,j+b)}\cdot K_{(i+a,j+b)}\cdot M_{(i+a,j+b)}) = \\ = C_{(i,j)}\cdot\sum_{a=-2}^{2}\sum_{b=-2}^{2}(L_{(a,b)}\cdot M_{(i+a,j+b)});\end{aligned} \tag{3.5}$$

De esta forma, $Out_{i,j}$ se puede simplificar por el producto de las entradas propia y vecinas (ponderadas por la distribución de luz), por un único coeficiente $C_{(i,j)}$ no conocido.

El objetivo de esta simplificación es el de aglutinar todos los efectos debido a las variaciones de ganancias en una única variable por posición discreta. Tras la simplificación, el modelo queda como se muestra en la

figura 3.8 (b). Cada $Out_{i,j}$ recoge información de las posiciones vecinas y aplica un factor de ganancia $C_{i,j}$. Este factor es directamente comparable con los resultados experimentales, de forma que es posible ajustar C con una comparación directa con los datos del barrido experimental debido a que para cada posición es necesario ajustar una única variable, siendo las distintas variables independiente entre sí.

Una vez realizada la calibración, tenemos en este punto un modelo que representa la energía medida por el PESIC para un centelleo ocurrido sobre cualquier ánodo del cristal. El estado de cada unidad discretizada del espacio, que representa la energía detectada por PESIC ante un centelleo en dicha posición, depende directamente no sólo de su coeficiente y las ganancias de su ánodo, sino de los ánodos vecinos. Las condiciones de contorno vienen dadas por los bordes del cristal, los cuales reflejan un porcentaje de la luz incidente [148]. En nuestra aproximación, el coeficiente de reflexión es definido como un valor constante obtenido a partir de los valores experimentales. Coeficientes de este valor demasiado altos o bajos, que no se adaptan bien al experimento, se traducen en una clara deriva de las ganancias $C_{i,j}$ que se acentúa a medida que nos acercamos a los bordes del detector. Esto es debido a que $C_{i,j}$ intenta compensar las falsas variaciones de energía debido al valor excesivamente alto o bajo del coeficiente escogido para el borde.

Una vez obtenido y calibrado el modelo, debemos obtener una regla de evolución que ecualice la ganancia a lo largo del detector. Para esto se debe actuar sobre las ganancias programables del ASIC. Cada célula, que comprende una sección 2D del espacio, define una regla de evolución cuya entrada son las energías calculadas por las células vecinas y que actúa directamente sobre la ganancia programable de dicha posición. La regla de evolución definida es la siguiente:

$$M_{i,j}(t+1) = M_{i,j}(t) - K'[Out_{i,j} - \bar{Out}], \tag{3.6}$$

donde $\bar{Out}$ representa la media de las salidas de las células vecinas, y K' define la velocidad de evolución de la regla.

Cada célula intenta que la energía producida por un centelleo en dicha

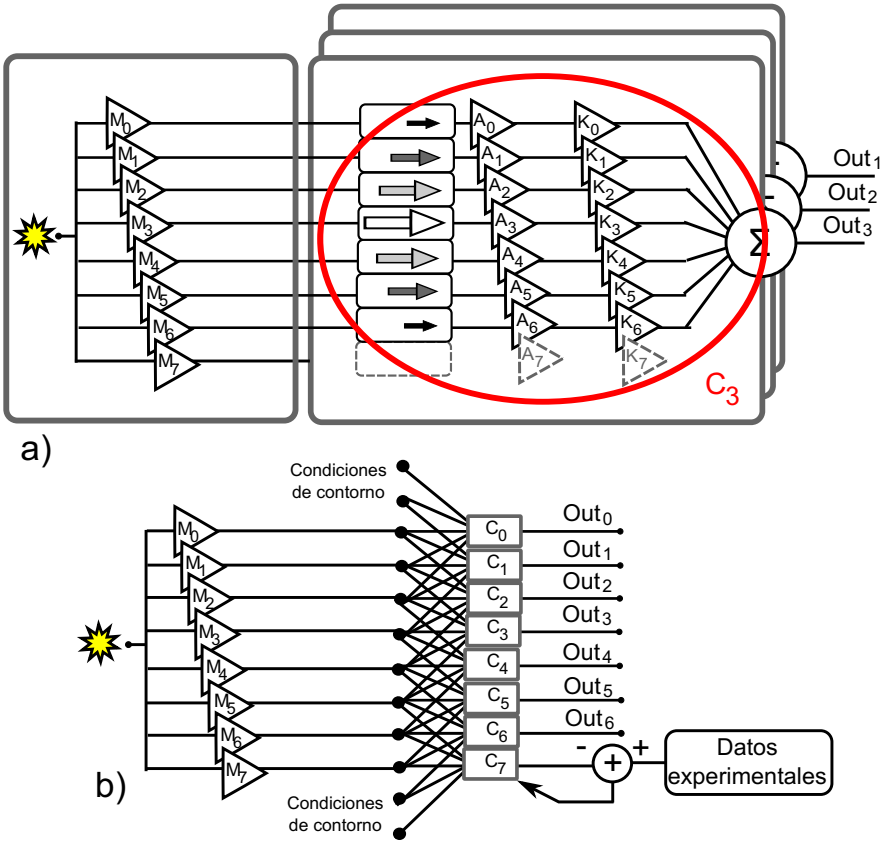

Figura 3.8: Reformulación del modelo. (a) Las ganancias no variables pueden ser reubicadas juntas (C_i). En función de la posición del centelleo i existe una única combinación de parámetros del modelo C_i. (b) C_i puede simplificarse como una ganancia que se aplica a la suma de las entradas cercanas multiplicadas por la distribución de luz correspondiente. Esta ganancia puede ajustarse de forma directa usando datos experimentales.

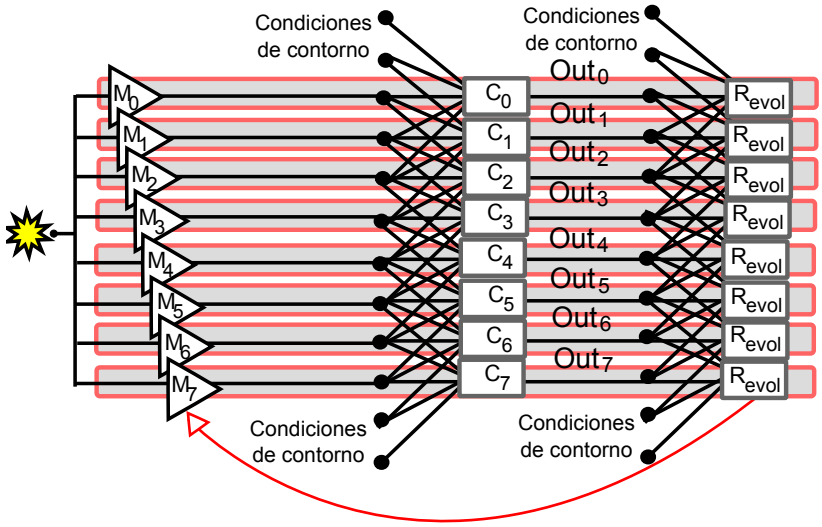

Figura 3.9: Modelo final, cada célula corresponde una posición discreta del detector. R_{evol} comprende la lógica de la regla de evolución que se aplica sobre las ganancias programables de PESIC.

posición sea similar a la media de las energías calculadas por las células vecinas actuando sobre la ganancia programable de su posición. El cambio en esta ganancia afecta en mayor medida a la propia célula, pero también en menor medida a las vecinas, que deberán reajustarse. El AC se estabilizará cuando las energías finales sean homogéneas. Asimismo se deben establecer límites en las ganancias mínimas y máximas programables debido a que el amplificador tiene un rango limitado (4 bits de precisión en el PESIC), así como valores de ganancia que tienden a 0 pueden desvirtuar la calidad de la salida. La figura 3.9 muestra una gráfica del modelo final.

3.2.3. Resultados

Para la implementación y simulación del modelo se ha desarrollado una aplicación en Java, mostrada en la figura 3.10. Esta aplicación permite de forma sencilla, a partir de datos experimentales, calibrar el modelo y

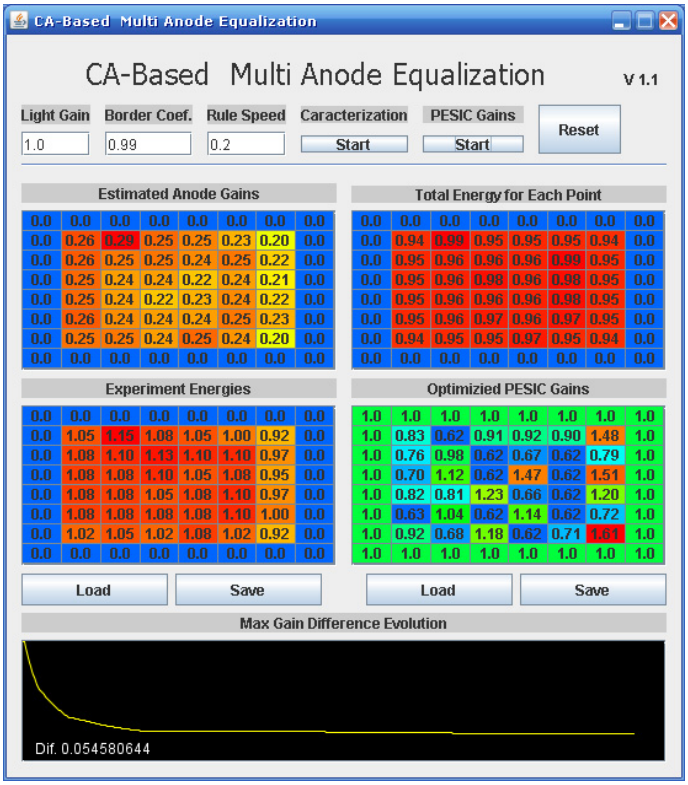

Figura 3.10: Captura de pantalla del software desarrollado que implementa el AC para la ecualización de ganancias del PESIC.

obtener las ganancias óptimas para ecualizar la energía.

La configuración utilizada para comprobar el modelo consiste en un sistema formado por dos PMT Hamamatsu H8500 montados sobre un cristal LSO de 42x42x10mm y el PESIC utilizado como electrónica de *front-end*. Debido a que el cristal cubre completamente el grupo de 6x6 ánodos internos, únicamente estos 36 ánodos han sido calibrados. Ambos detectores están enfrentados entre ellos, como muestra la figura 3.11. En primer lugar, para medir las energías sobre cada punto experimentalmente, se utilizó una fuente radioactiva de Na-22 la cual se fue desplazando a lo largo del de-

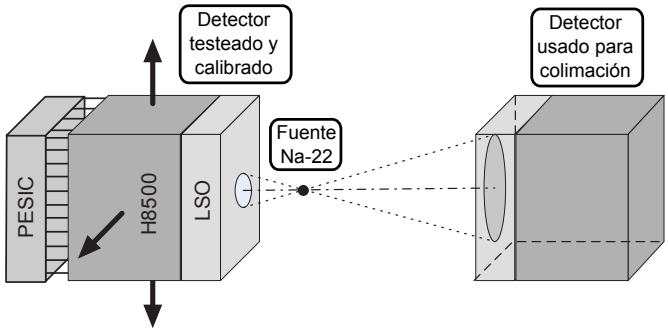

Figura 3.11: Esquema del sistema de pruebas montado: Dos detectores enfrentados para la detección en coincidencia del par de fotones gamma generados por la fuente radioactiva.

tector. Un total de 30000 eventos en coincidencia han sido obtenidos para cada posición. A partir de los eventos, la energía fue calculada sumando las componentes (A+B+C+D), leídas del detector y almacenadas en una computadora mediante un sistema de adquisición de datos. A partir de un histograma de las energías para cada posición, se identificó el *fotopico*, el cual indica la energía detectada por una interacción de un fotón gamma. Dichas energías fueron las utilizadas en el modelo como las energías experimentales.

Con esta información y por medio de la aplicación desarrollada, se ha estimado las ganancias óptimas para uno de los *front-ends*. PESIC ofrece un rango de amplificación programable de 0 a 1.875 en pasos de 0.125. Teniendo en cuenta esta limitación, se han fijado límites superior e inferior de 0.75 y 1.75 en las posibles ganancias obtenidas en el modelo.

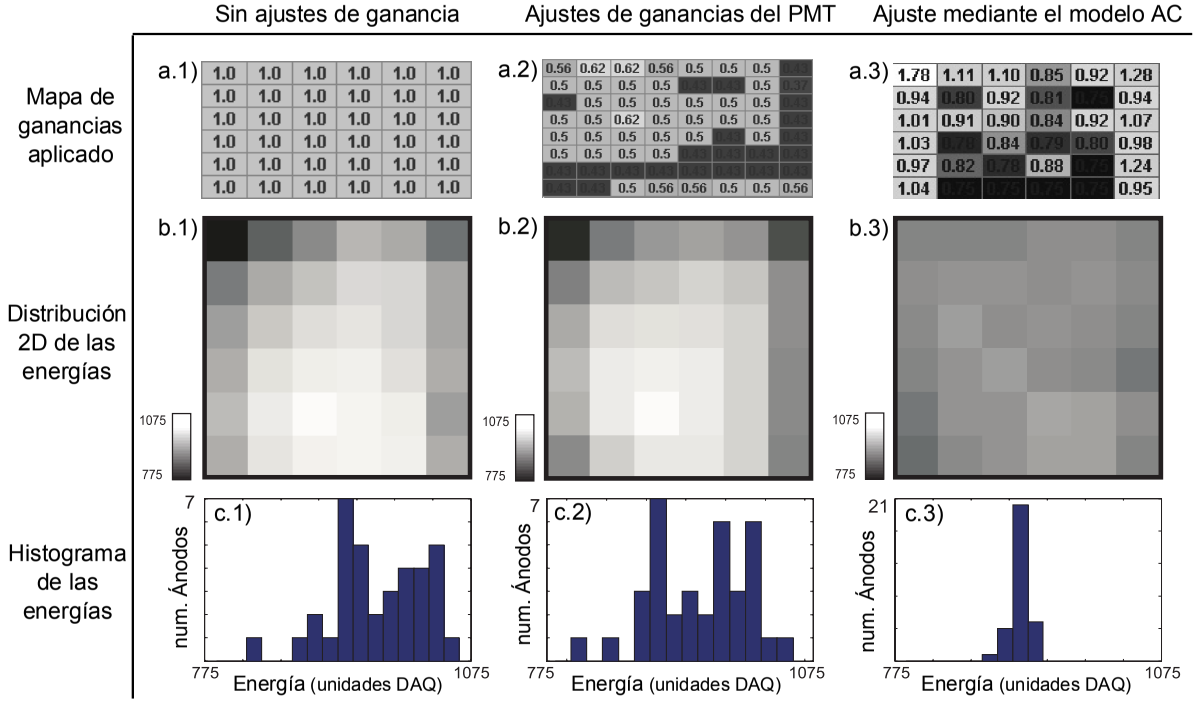

Figura 3.12: Resultado de la calibración del detector con PESIC mediante el modelo basado en AC. (a) Mapas de ganancias aplicados sobre los preamplificadores de PESIC. (b) Distribución 2D de la energía detectada mediante el uso de la fuente de Na-22 colocada sobre los distintos ánodos del fotomultiplicador. (c) Histograma de energías detectadas.

Finalmente, tres tests se han llevado a cabo con el objetivo de analizar las variaciones de energías en tres casos distintos: sin compensación de ganancias, compensando las inhomogeneidades de las ganancias del PMT, suministradas por el fabricante y aplicando el conjunto de ganancias obtenido con el modelo AC. La figura 3.12 muestra los resultados de los tres experimentos, así como el conjunto de coeficientes programables cargados en PESIC. Los resultados muestran que el ajuste de los preamplificadores de PESIC mediante el método propuesto da lugar a una homogeneización de la energía medida por PESIC a lo largo de la superficie del detector.

3.2.4. Conclusiones

En este apartado se ha analizado la posibilidad de aplicar la filosofía de modelado basada en ACs para la obtención de un sistema que permita ecualizar la ganancia a lo largo de un detector indirecto de rayos gamma basado en un fotomultiplicador multiánodo. Dicho modelo se sirve de las nuevas electrónicas diseñadas para este tipo de detectores, los cuales poseen una matriz de preamplificadores de ganancia programable.

El alto acoplamiento del sistema, así como la evidente discretización espacial del mismo hacen viable el enfocar dicho problema desde un punto de vista basado en AC.

Los buenos resultados de calibración indican que el modelo emula de forma adecuada la distribución de luz generada por el centelleo a lo largo del fotomultiplicador, así como los efectos de las variaciones de ganancias de los distintos elementos del detector en el cálculo de la energía depositada por el evento.

Por lo tanto, el AC presentado se muestra como un método válido para la calibración de ganancias en detectores de cristales continuos. En este tipo de detectores, la energía detectada para un determinado centelleo depende de forma directa de los ánodos y electrónica cercanos a la posición donde el centelleo se ha producido. Este efecto es el que ha sido explotado para aplicar la filosofía de modelado basada en ACs.

Capítulo 4

Implementación de Superficies Dinámicas en GPUs: Simulación del Autómata Celular Continuo para Procesos de Atacado Anisótropo Húmedo

Como se ha explicado previamente en el capítulo 2, las GPUs han demostrado ser una arquitectura eficiente para la simulación de numerosos algoritmos masivamente paralelos, incluyendo también los AC. En este capítulo se presentan dos aportaciones:

1. Una metodología que permite la simulación eficiente de superficies dinámicas sobre GPUs.
2. Una implementación del ACC sobre la arquitectura de computación CUDA, la cual se presenta junto con un estudio que analiza y justifica las decisiones de diseño tomadas.

El resultado de estas dos aportaciones se resume en la creación de un simulador de atacado anisótropo húmedo: *GPUetch*, el cual incrementa la velocidad de simulación en varios órdenes de magnitud sobre simuladores existentes actualmente (comerciales y educativos) que realizan sus cálculos sobre CPUs.

4.1. Simuladores Secuenciales del Autómata Celular Continuo

La simulación del ACC sobre CPUs ha sido ampliamente estudiada e implementada en el pasado. Las principales características de dichas implementaciones para obtener la mayor eficiencia posible son [149]:

- La utilización de árboles octales para la reducción de memoria utilizada.
- La creación y mantenimiento de una lista enlazada la cual alberga los átomos superficiales.

En estos simuladores, la evolución en cada paso de tiempo consiste en dos tareas: marcado y eliminación. En la tarea de marcado, se evalúan todos los átomos superficiales almacenados en la lista, reduciendo su ocupación acorde a las ecuaciones definidas por el modelo. Los átomos cuya ocupación se ha reducido a un valor menor o igual que cero son marcados para ser eliminados de la lista. La segunda etapa consiste en la eliminación de la lista enlazada de todos los átomos marcados y la adición a la misma de nuevos átomos que aparecen en este paso de tiempo en la superficie.

Pese a que en el capítulo 6 se realiza un análisis detallado del coste computacional de la simulación del atacado anisótropo húmedo basado en ACCs, podemos adelantar aquí que dicho coste computacional está directamente relacionado con el tamaño de la superficie simulada. Asimismo, la alta carga computacional de estos métodos ya ha sido relatada con anterioridad [150] como una limitación importante.

En la actualidad existe el software VisualTAPAS 2.0 [122], y su variante comercial IntelliEtch 1.0 [151], los cuales simulan el proceso de grabado

húmedo mediante ACCs. Pese a ser una implementación satisfactoria del método, la alta carga computacional que supone la simulación de ACCs hace que simulaciones con este software duren desde varios minutos hasta algunas horas.

Habitualmente, el proceso de desarrollo de un MEMS mediante una herramienta CAD de atacado anisótropo consiste en realizar numerosas pruebas, modificando los parámetros como tiempos de atacado, geometría de las máscaras o las características del atacante, por lo que este proceso de diseño se vería beneficiado en gran medida si las simulaciones de dicho proceso pueden ser aceleradas, acortando de esta forma el ciclo de diseño.

Ésta es la motivación que nos ha llevado a intentar implementar este método en nuevas arquitecturas computacionales. La utilización de GPUs como arquitectura objetivo es debido a su éxito previo con algoritmos masivamente paralelos en general y ACs en particular. Asimismo, el hecho de utilizar una arquitectura que previamente se ha extendido a la práctica totalidad de computadores actuales, aumenta la repercusión de cualquier avance realizado en este campo debido a que cualquier diseñador de MEMS puede beneficiarse de estas mejoras sin la necesidad de utilizar ningún hardware especializado.

4.1.1. Árboles Octales para el Almacenamiento del ACC

En una gran cantidad de casos, la interacción entre dos regiones físicas (p.e. materiales) puede modelarse como una superficie dinámica. Un claro ejemplo de este caso es el modelo ACC para grabado húmedo, donde la superficie modelada representa la interacción entre la superficie de silicio y el atacante.

A la hora de simular una superficie dinámica, únicamente la región espacial donde esta teniendo lugar la interacción guarda algún interés, por lo que almacenar todo el volumen tridimensional que engloba a la superficie puede resultar una aproximación poco eficiente al problema.

Una alternativa que ha sido utilizada para no almacenar todo el volumen en la memoria de la computadora es la de almacenar los puntos de la superficie como hojas de un árbol. La idea es representar la región cúbica

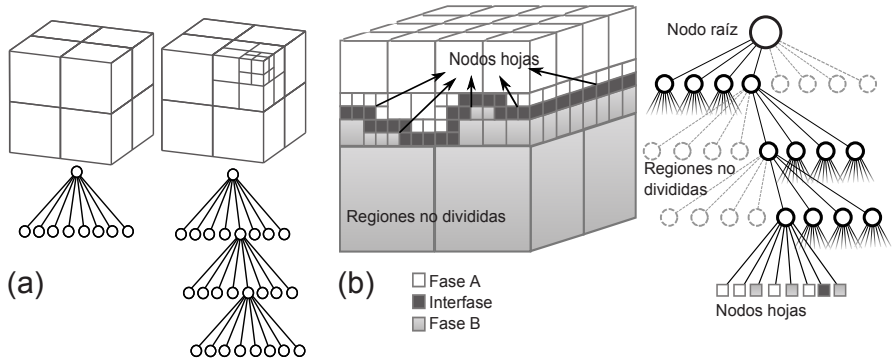

Figura 4.1: (a): División de un espacio mediante un árbol octal. (b): Árbol octal aplicado a un modelo que representa la interacción de dos medios.

que engloba a la superficie como un único nodo, denominado *raíz*. Esta región es subdividida en ocho subregiones, representadas por ocho nodos hijos del raíz (Figura 4.1 (a)). Para cada subregión, es posible incrementar el detalle de la descripción subdividiendo otra vez la región (añadiendo ocho nodos hijos a cierto nodo). Esta subdivisión espacial puede continuar hasta el límite de detalle del modelo, donde en este caso, los nodos (llamados *hojas*) representan la mínima unidad de división del modelo: vóxeles en el caso de un volumen genérico 3D, o células en caso de un AC.

Esta metodología, denominada *árbol octal*, permite almacenar con todo el detalle necesario las regiones espaciales donde la interacción se está llevando a cabo, mientras que las regiones sin interés no ocupan espacio en memoria. Los árboles octales ofrecen un equilibrio entre velocidad de acceso ($O(\log(N))$) y eficiencia de almacenamiento [152]. La figura 4.1 (b) muestra un árbol octal definido para el modelado de una interacción entre dos espacios. Las zonas alejadas de la interacción no son de interés, y por lo tanto el árbol octal no las define en detalle. Por otro lado, todas las porciones de espacio donde está definida la superficie, son refinadas en el detalle requerido por el modelo.

Esta estructura de datos ha sido utilizada para aplicaciones gráficas y simulaciones científicas en muchos contextos: algoritmos de trazado de

rayos para iluminación de escenas (denominados en inglés *Ray tracing*), simulaciones adaptativas, descomposiciones paralelas dinámicas, etc.

En 1991, D. Libes propuso utilizar los árboles octales para la simulación eficiente de superficies dinámicas [153]. La utilización del árbol octal posibilita, además, la fácil implementación de una superficie no delimitada. El espacio definido por el árbol octal puede ser ampliado en cualquier momento simplemente incluyendo el nodo raíz actual como un hijo de un nuevo nodo que abarca un volumen mayor. El ACC es un caso concreto de superficie dinámica donde la implementación del árbol octal supuso un ahorro significativo de memoria utilizada [149, 154].

En el caso de los ACC, la mínima unidad espacial definida por el árbol octal es la denominada *celda unidad ortorrómbica*, la cual contiene una pequeña cantidad de átomos de silicio. La celda unidad se basa en el hecho de que cualquier material cristalino, como el silicio, puede ser construido a partir de repetir la celda unidad a lo largo de los tres ejes. Una celda unidad ortorrómbica es una celda cúbica reescalada en las tres direcciones en la cual todos sus ángulos son rectos. La metodología para obtener una celda ortorrómbica para las simulaciones de ACCs ha sido estudiada previamente [154]. El tamaño y la forma de la celda unidad depende de la orientación de la superficie formada por el cristal de silicio, siendo 4 átomos de silicio la celda más pequeña (superficie tipo (100)), hasta miles de átomos (1432 átomos para superficie (977)).

La simulación de cualquier orientación de silicio más allá de las orientaciones principales ofrecidas en las obleas de silicio comerciales es una funcionalidad importante desde un punto de vista de investigación. El poder simular cualquier orientación nos da la posibilidad de caracterizar fácilmente el modelo teórico para un amplio rango de datos. La utilización del árbol octal permite, de una manera natural, la utilización de las celdas unidad como base de construcción un cristal de silicio.

A lo largo del capítulo veremos que el tamaño de la celda unidad influye en gran proporción en la velocidad de simulación. Para estudiar este hecho necesitamos introducir el concepto de *supercelda*. Una supercelda es un conjunto de una o varias celdas unidad adyacentes que se unen para definir una nueva mínima discretización del espacio (de mayor tamaño). La figura

4.2 muestra la asociación entre los árboles octales y la utilización de celdas unidad o superceldas.

4.1.2. Implementación GPU de un Árbol Octal para la Simulación de Superficies Dinámicas

Como se ha comentado en el capítulo 2, la filosofía de computación de las GPUs se basa en aplicar de forma masivamente paralela el mismo algoritmo, denominado *kernel* sobre distintos elementos.

A priori, la simulación del ACC puede ser implementada de forma óptima sobre una GPU: la misma regla de evolución debe aplicarse sobre todas las células que forman la superficie del substrato de silicio, por lo que separando la superficie en pequeños trozos y asignando un hilo de ejecución para cada uno se podría, en un principio, paralelizar el algoritmo. Habitualmente, estas superficies pueden estar formadas por cientos de miles de átomos, permitiendo la utilización de todos los multiprocesadores existentes en las GPUs actuales.

Sin embargo, una teórica gestión del árbol octal por parte de la GPU sufre de una gran restricción: diversas tareas relacionadas con la modificación del octree son secciones críticas. Esto significa que sólo un hilo de ejecución puede realizar estas tareas al mismo tiempo. La no exclusión de los diversos hilos a la hora de realizar estas tareas puede dar lugar a condiciones de carrera, pudiendo causar corrupción en los datos. Las tareas relacionadas con un árbol octal que son consideradas secciones críticas son las siguientes:

- Asignación o liberación de una región de memoria asociada a un nodo hoja (supercelda).
- Modificación de la estructura del árbol octal, adición o eliminación de nodos.
- La decisión de particionar o simplificar determinada región espacial.

Un ejemplo de este hecho puede ser cuando dos hilos intentan asignar región de memoria a un nuevo nodo hoja. Si el acceso no está controlado,

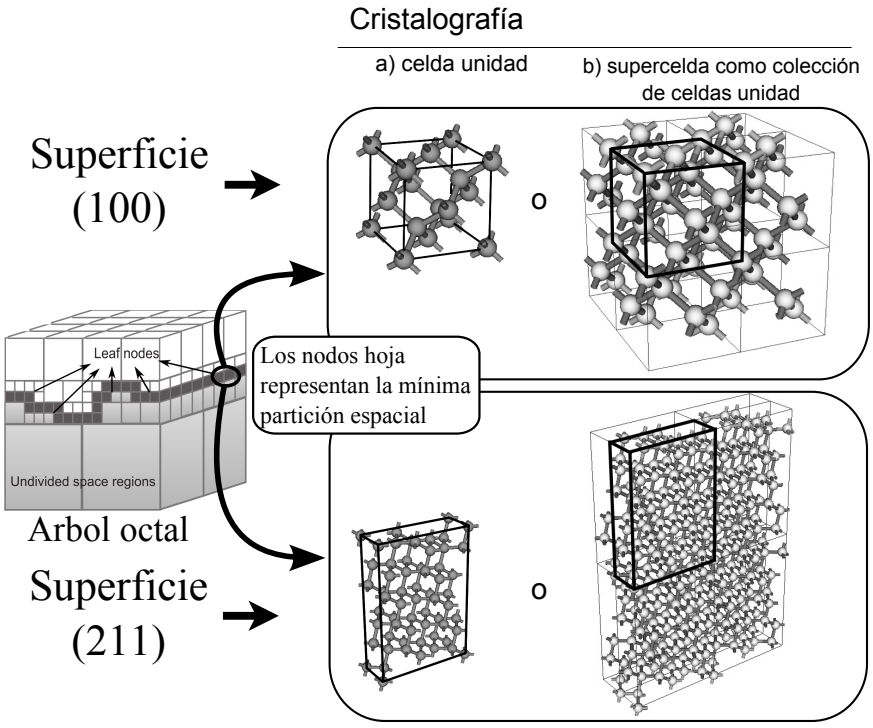

Figura 4.2: Los nodos hojas definen la mínima región espacial. En el ACC esta región depende de la estructura cristalina del silicio y puede ser la correspondiente celda unidad (a), o una agrupación de celdas unidad: superceldas (b).

la misma sección de memoria podría acabar siendo asignada a dos regiones espaciales distintas, causando un funcionamiento erróneo. Por lo tanto, es obligatoria la exclusión de hilos para operar sobre el árbol octal.

Pese a que las GPUs actuales poseen la capacidad de realizar operaciones atómicas sobre la memoria global, la serialización de los hilos está muy distante del paradigma de programación de las GPUs. Asimismo, el patrón de acceso a la memoria global requerido para obtener *coalescing* no se adapta bien a la estructura del árbol octal. Esto hace que la implementación GPU del árbol octal difícilmente obtenga buenos resultados, siendo más adecuada la implementación en alguna arquitectura computacional diseñada para la rápida ejecución de algoritmos secuenciales.

La solución propuesta para poder simular de forma eficiente modelos basados en el ACC en arquitecturas GPU consiste en desacoplar los cálculos relacionados con el modelo y la administración de la superficie dinámica basada en el árbol octal. La superficie dinámica es almacenada en su totalidad en la GPU, encargada de aplicar la operaciones relacionadas con el modelo. Por otro lado, el árbol será almacenado y modificado por la CPU de la computadora (procesador maestro), optimizada para la ejecución secuencial de código.

Para poder implementar esta modalidad, el almacenamiento de datos debe seguir las siguientes directrices:

- La memoria global de la GPU debe ser entendida como un vector de *Clústeres de Memoria* (CMs). Cada CM tiene la capacidad de almacenar todos los datos del modelo para una supercelda.
- La CPU dispone de un repositorio de punteros a CMs de la GPU no utilizados.
- A pesar de que la administración del árbol octal es realizada por la CPU, los nodos hoja del árbol almacenan punteros a CMs que están siendo utilizados para almacenar el ACC.

Esto significa que cada nodo hoja del árbol almacenado en la CPU apunta a una supercelda, la cual contiene M átomos y está almacenada en un CM dentro de la GPU. Con esta implementación, la superficie

está almacenada completamente en la GPU, mientras que la CPU posee la información necesaria sobre los CMs utilizados y libres para tomar de forma adecuada decisiones sobre qué regiones espaciales deben asignarse o eliminarse en la GPU.

El desacoplo propuesto requiere una comunicación continua entre ambas plataformas. Mientras se procesa la superficie dinámica, los algoritmos ejecutados en la GPU pueden decidir que una región espacial en particular debe ser modificada (por ejemplo, una supercelda no se necesita más y puede ser eliminada). Esta información debe ser indicada a la CPU para que haga las operaciones pertinentes sobre el árbol. Para cumplir este objetivo, cada CM posee una variable, como una variable de señal o *flag*, para indicar si es necesario algún tipo de proceso sobre dicha región. Los hilos de ejecución lanzados en la GPU activarán los *flags* cuando sea necesario, indicando el tipo de acción. La CPU recibirá y analizara dichos *flags*. En caso de que alguna modificación sea necesaria, a partir de la numeración del CM es posible obtener la posición espacial en el árbol mediante una simple tabla *look-up*, pudiendo actuar sobre ella.

Por último, la CPU debe informar a la GPU sobre CMs recientemente asignados o liberados. La figura 4.3 muestra un diagrama de la estructura de datos y procedimiento propuesto. Utilizando este método, todos los cálculos masivamente paralelos relacionados con el modelo pueden ser ejecutados por la GPU, mientras que la administración del árbol puede ser realizada por la CPU.

Debido a este desacoplo, las posiciones de las superceldas en el espacio 3D no poseen ninguna correlación con las posiciones de los CMs correspondientes en la memoria global. Debido a que muchos modelos físicos/químicos requieren la inspección de la vecindad de cada átomo superficial con el objetivo de decidir la nueva serie de eventos, es necesario permitir un acceso rápido a las superceldas vecinas dentro de la GPU, especialmente para los átomos en la periferia de las superceldas. Asimismo, mantener el número de lecturas de memoria independiente de la complejidad del modelo es una funcionalidad importante cuando se buscan superceldas vecinas.

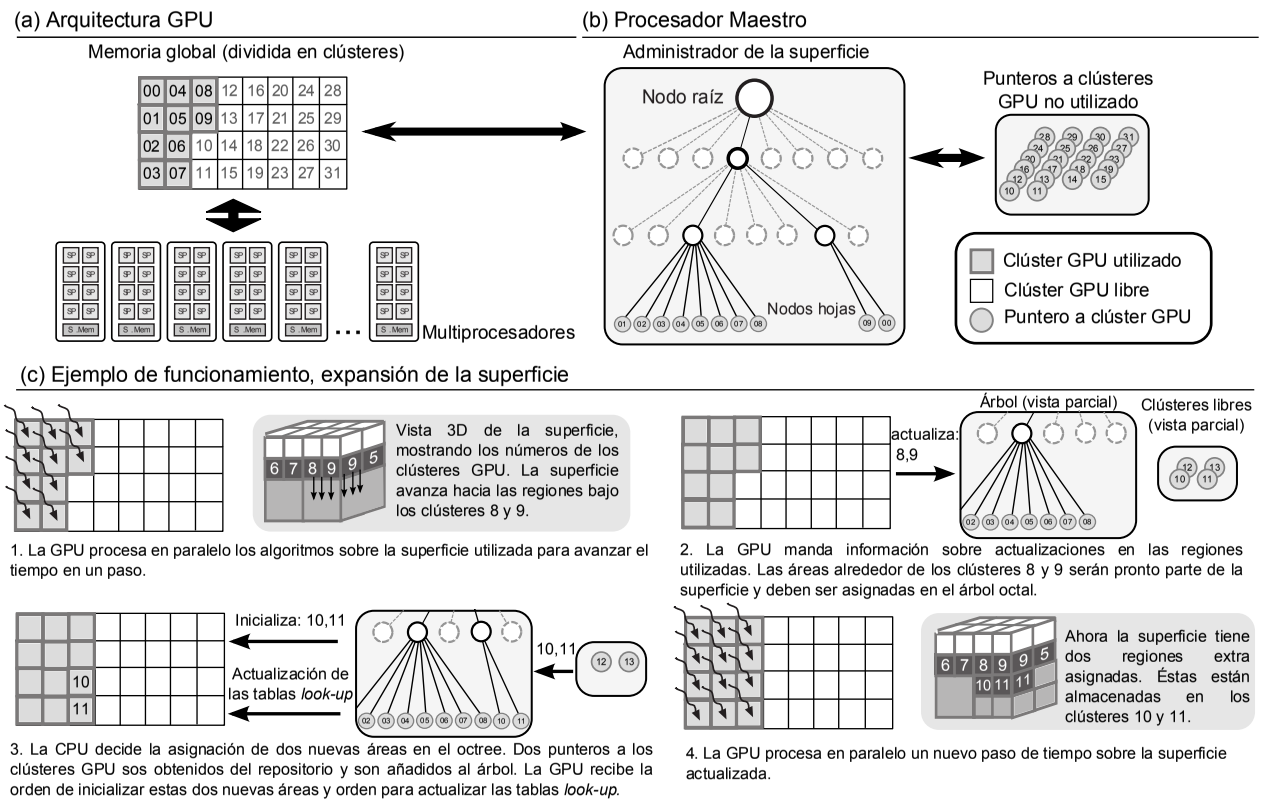

Figura 4.3: Implementación propuesta para la simulación de superficies dinámicas usando árboles octales sobre GPUs. (a) Memoria de la GPU dividida en clústeres. (b) Estructuras de datos en la memoria del procesador maestro. (c) Ejemplo de adición de nuevas regiones a la superficie.

Una estructura de datos que posee ambas funcionalidades es una simple tabla *look-up* la cual almacena punteros a las superceldas vecinas para cada CM. Esta tabla debe ser almacenada en la memoria global de la GPU. Para poder mantener la información de esta tabla actualizada, los cambios en la morfología de la superficie ocurridos a lo largo de la simulación deben verse reflejados inmediatamente. Debido a que es la CPU quien mantiene la información acerca de la organización espacial de los clústeres de la GPU, ella es la responsable de enviar las actualizaciones de esta tabla a la GPU.

Finalmente, sobre el repositorio de CMs libres alojado en la memoria de la CPU, una pila funciona de forma adecuada. Los tiempos de acceso de una pila son $O(1)$, y debido a su comportamiento LIFO (*Last In First Out*), un CM recientemente liberado será el siguiente a ser utilizado. Esta funcionalidad previene la dispersión excesiva de las superceldas a lo largo de la memoria global de la GPU. El mantener los datos compactados en la memoria hace que los algoritmos paralelos sean más eficientes debido a que se puede asumir que la memoria de la GPU está totalmente ocupada hasta cierto límite, de manera que el procesado de datos se tiene que aplicar solamente hasta dicho punto.

4.1.3. Detalles de Implementación del ACC en la GPU

En la presente sección procedemos a mostrar los detalles de implementación del ACC definido por M. A. Gosalvez (sección 2.3.4.1) para la arquitectura Nvidia CUDA, tomando como base la metodología explicada en la sección anterior. Pese a que el modelo ACC parece ofrecer resultados precisos, los simuladores académicos y comerciales que lo implementan tienen una funcionalidad limitada debido a la relativa baja eficiencia de procesado cuando se ejecuta sobre un procesador secuencial.

Los detalles de implementación ofrecidos en esta sección están formados por: una descripción detallada de la utilización de la memoria de la GPU, una descripción en pseudocódigo de los *kernels* implementados en la GPU y una enunciación del procedimiento de administración del árbol octal por parte de la CPU.

4.1.3.1. Variables Principales

Por definición, los ACs son modelos muy intensivos en el acceso a memoria: para cada átomo, el estado de todos los vecinos debe ser leído y el nuevo estado debe ser determinado y guardado en cada paso de tiempo. En estudios previos, se ha llegado a la conclusión de que, para modelos matemáticos con patrones de acceso a memoria similares tal como *Finite-Difference Time-Domain* (FDTD), el límite en el rendimiento para la implementación sobre arquitecturas GPU en estos casos es el ancho de banda de la memoria global [72]. Por lo tanto, la mayoría de optimizaciones realizadas en la implementación GPU del ACC tienen el objetivo de reducir la lectura/escritura de datos en la memoria global.

Nuestra implementación es capaz de simular un amplio rango de distintas superceldas que puede variar en tamaño y estructura interna. Los átomos dentro de la supercelda están numerados de 0 a $M-1$, siendo M el tamaño de la supercelda. Para referirnos a un átomo dentro de la red cristalina utilizamos cuatro coordenadas $(n1, n2, n3, m)$, donde el trío $(n1, n2, n3)$ determina la posición espacial (discreta) de la supercelda y m, identifica al átomo de la supercelda. Debido a que cada supercelda se almacena en un CM, la posición en memoria p para el átomo m del CM K es: $p = m + M \cdot K$.

La tabla 4.1 muestra las variables principales utilizadas en nuestra implementación GPU. Cada una de las variables es almacenada como un vector extenso y continuo donde todos los CMs guardan sus datos en la posición correspondiente. *num_Atoms* y *num_MC* indican la cantidad máxima de átomos y CMs que pueden ser utilizados en la simulación. Estos valores dependerán de la cantidad de memoria global libre en la GPU.

Las variables *Occ* y *Erate* almacenan la ocupación Π_i y la velocidad de extracción actual r_i para todos los átomos superficiales. La razón de almacenar la velocidad de extracción es debido a que en muchos casos, la costosa tarea de calcular r_i en cada paso de tiempo no es necesaria: dicha velocidad se mantiene constante para una gran cantidad de pasos de tiempo. Por ejemplo, para una superficie en el plano (100) cuando se simula un atacante KOH 40wt % a 70°C, la velocidad se calcula la primera vez que emerge el átomo en la superficie, para mantenerse invariante alrededor

de 13 pasos de tiempo, momento en que el átomo es eliminado.

De la misma forma, $TRate$ almacena la tabla de velocidades R y $N1sb$ almacena las variables n^{1s} y n^{1b} juntas utilizando un byte por átomo. Las variables $CellInfo$, $UCInfo$ y $TArea$ se refieren a las tablas look-up que permiten a los hilos de ejecución acceder a la vecindad de cualquier átomo, como se describe en el siguiente párrafo. Finalmente, existe un conjunto de variables adicionales, como Syb, que indica el tipo de átomo (superficial, enterrado, máscara) , $MCstate$ y $TBuf$.

La tabla *look-up* $CellInfo$ puede ser entendida como una matriz 2D con 4 columnas y hasta 4096 filas, donde la columna j de la fila i almacena la coordenada m del vecino j del átomo i. Asimismo, $UCInfo$ es también una matriz 2D de las mismas dimensiones donde la columna j de la fila i almacena las coordenadas $(n1, n2, n3)$ relativas del vecino j del átomo i de cualquier supercelda. Estas tres coordenadas están compactadas en un mismo byte, utilizando un bit por eje espacial y posición relativa (arriba-abajo, izquierda-derecha, delante-detrás), lo que hace un total de 6 bits utilizados. Las tablas $CellInfo$ y $UCInfo$ son complementarias en el sentido de que ambas son necesarias para obtener la localización de un átomo superficial vecino. Ambos vectores están almacenados en la caché de constantes de la GPU. Por comparación, la tabla $TArea$ posee una estructura distinta, que puede verse como una matriz 2D con 6 columnas y num_MC filas, la cual almacena, para cada supercelda, las posiciones de memoria donde están almacenadas las superceldas adyacentes (norte, sur, este, oeste, delante, detrás). Este segundo tipo de tabla, ya mencionada en la sección 4.1.1, es almacenada en la memoria global de la GPU y se utiliza para recuperar la posición 3D de cualquier supercelda, solventando el hecho de que las superceldas vecinas están almacenadas en CMs distantes. Existen, por tanto, dos tipos de tablas *look-up*, una para definir la estructura interna de la supercelda, y otra para definir la superficie a partir de la localización espacial de las superceldas.

El proceso de atacado del silicio puede ser entendido como la eliminación ocasional de átomos de la superficie. En cada paso de tiempo únicamente una pequeña cantidad de átomos es eliminada, y como resultado, las vecindades y las velocidades de extracción se mantienen invariantes la

mayor parte del tiempo, por lo que no es necesario recalcularlos en cada paso. Como ejemplo, ratio de átomos actualizados/visitados puede rondar 1/50 para simulaciones de paso constante y puede bajar hasta valores de 1/1000 o menor para simulaciones de paso variable, considerando velocidades de extracción relativamente grandes y celdas unidad pequeñas en ambos casos.

Manteniendo en memoria información acerca de la vecindad (n^{1s} y n^{1b}), es posible reducir, por tanto, la cantidad de lecturas de memoria global. Asimismo, leer una entrada de la tabla R para un determinado átomo requiere en principio 16 lecturas de vecindad, (4 primeros vecinos y 12 segundos vecinos). Una forma de reducir la cantidad de lecturas es manteniendo n^{1s} y n^{1b} en memoria y obteniendo, para los átomos superficiales, n^{2s} y n^{2b} a partir de los valores n^{1s} y n^{1b} de los primeros vecinos:

$$n_i^{2s} = \left(\sum_{j=0}^{3} n_j^{1s} \right) - 4; \qquad n_i^{2b} = \sum_{j=0}^{3} n_j^{1b}. \tag{4.1}$$

De esta forma, la obtención de la vecindad se realiza en dos pasos: actualización de datos acerca de primeros vecinos (4 lecturas por átomo), y obtención de segunda vecindad a través de la primera vecindad de los primeros vecinos (4 lecturas por átomo). Una vez obtenido la velocidad adecuada, se almacena en la variable *Erate*. A pesar de que n^{1s} y n^{1b} están almacenados en la memoria global (vector $N1sb$ en la Tabla 4.1), R permanece constante durante la simulación, por lo que puede ser almacenada en la caché de constantes, más rápida que la memoria global.

4.1.3.2. División del Algoritmo entre los Hilos de Ejecución

Como se ha descrito en la sección 4.1.1, un CM almacena los datos del modelo correspondientes a una supercelda (M átomos). En este estudio, una colección de G CMs almacenados en memoria de forma contigua es denominado un *Grupo de Memoria* (GM). Por simplicidad, escogemos G para que sea igual al número de hilos por bloque. Un ejemplo de un GM de un tamaño $G = 12$ es mostrado en la figura 4.4, donde cada CM almacena

Tabla 4.1: Variables principales definidas en la GPU. Las variables marcadas con $^{(*)}$ son leídas a través de texturas en aquellos algoritmos en los que el procedimiento realiza accesos frecuentes a la vecindad de los átomos.

Nombre	Tamaño(tipo)	Memoria	Uso de la variable
Occ	$num\_Atoms(float)$	global	Almacenar ocupación del átomo Π.
Erate	$num\_Atoms(float)$	global	Almacena la velocidad de extracción actual r.
Syb	$num\_Atoms(byte)$	global$^{(*)}$	Almacena el tipo de átomo (enterrado, superficie, máscara...). Un bit usado como *flag* para actualización de vecindades.
N1sb	$num\_Atoms(byte)$	global$^{(*)}$	Almacena n^{1s} y n^{1b} (compactados en un byte).
MCstate	$num\_MC(short)$	global	Almacena en número de átomos no eliminados en cada CM.
TArea	$6 \cdot num\_MC(int)$	global$^{(*)}$	Tabla *look-up* de segundo nivel: punteros a las posiciones de memoria de las superceldas vecinas (norte, sur, este, oeste, arriba, abajo).
TBuf	$K \cdot num\_MC(uint)$	global	*Buffer* para trasferencia de datos entre CPU y GPU.
TRate	$4096(float)$	constante	Tabla de velocidades precalculadas R.
CellInfo	$\leq 4 \cdot 4096(short)$	constante	Tabla *look-up* de primer nivel: coordenadas m de los cuatro átomos vecinos.
UCInfo	$\leq 4 \cdot 4096(byte)$	constante	Tabla *look-up* de primer nivel: coordenadas relativas (n_1, n_2, n_3) para los cuatro átomos vecinos.

datos para $M = 9$ átomos. En la práctica, agrupar los CMs en GMs es tan fácil como escoger a nivel de bloque de hilos la región de memoria que debe ser procesada: El bloque 0 procesa desde CM_0 hasta CM_{G-1}; El bloque 1 procesa desde CM_G hasta CM_{2G-1}; etc...

En nuestra implementación, todos los hilos ejecutan un bucle de M iteraciones. Para cada iteración, cada hilo procesa un átomo y cada bloque de hilos procesa G átomos almacenados en posiciones contiguas en el mismo GM, como se describe en la figura 4.4(a). Con en el fin de procesar todos los CMs activos, el número adecuado de bloques de hilos debe ser lanzado en cada ejecución de *kernel*. Este valor es decidido por la CPU,

basado en la cantidad de CMs que están actualmente activos en la GPU. En conjunto, cada hilo procesa una cantidad de átomos equivalente al tamaño de la supercelda (M átomos), y cada bloque de hilos procesa una colección de átomos contiguos en la memoria de la GPU. Por lo tanto, este procedimiento proporciona un patrón de acceso unificado o *coalesced* a la memoria global de la GPU, reduciendo la cantidad de accesos a memoria. Por último, la figura 4.4(b) describe un ejemplo donde cada hilo evalúa únicamente los átomos de un único CM. Pese a que es un comportamiento más sencillo de programar, da lugar a un acceso no eficiente a la memoria global.

4.1.3.3. Bucle de Simulación

La figura 4.5 muestra una vista abstracta del bucle principal de simulación de la implementación propuesta. Basado en la cantidad total de CMs utilizados para almacenar la superficie en la memoria de la GPU, la CPU determina la cantidad requerida de hilos justo antes de iniciar el siguiente paso, para poder lanzar los *kernels* con el número adecuado de bloques de hilos. En los apartados a) y b), cada bloque opera sobre un GM.

La parte a) de la figura 4.5 tiene el objetivo de actualizar las velocidades de extracción de los átomos superficiales. Esto es llevado a cabo ejecutando dos *kernels* similares de forma secuencial, de forma que las tareas de (1) a (4) son realizadas por el segundo kernel después de que el primero las haya realizado. En primer lugar, ambos *kernels* comprueban si es necesario alguna actualización leyendo datos del vector *Syb*, la cual posee para cada elemento un *flag* de 1 bit utilizado para indicar si algún cambio ha sucedido en la vecindad. Si el átomo evaluado tiene el *flag* activado, entonces:

1. El primer kernel lee el estado de los átomos vecinos, calcula n^{1s} y n^{1b}, y lo almacena en *N1sb*.

2. El segundo kernel lee las variables *N1sb* de los propios átomos y de su primera vecindad, calcula n^{2s} y n^{2b} usando la ecuación 4.1, obtiene la velocidad de extracción $R(n^{1s}, n^{1b}, n^{2s}, n^{2b})$ del elemento correspondiente de *TRate* y lo almacena en *Erate*.

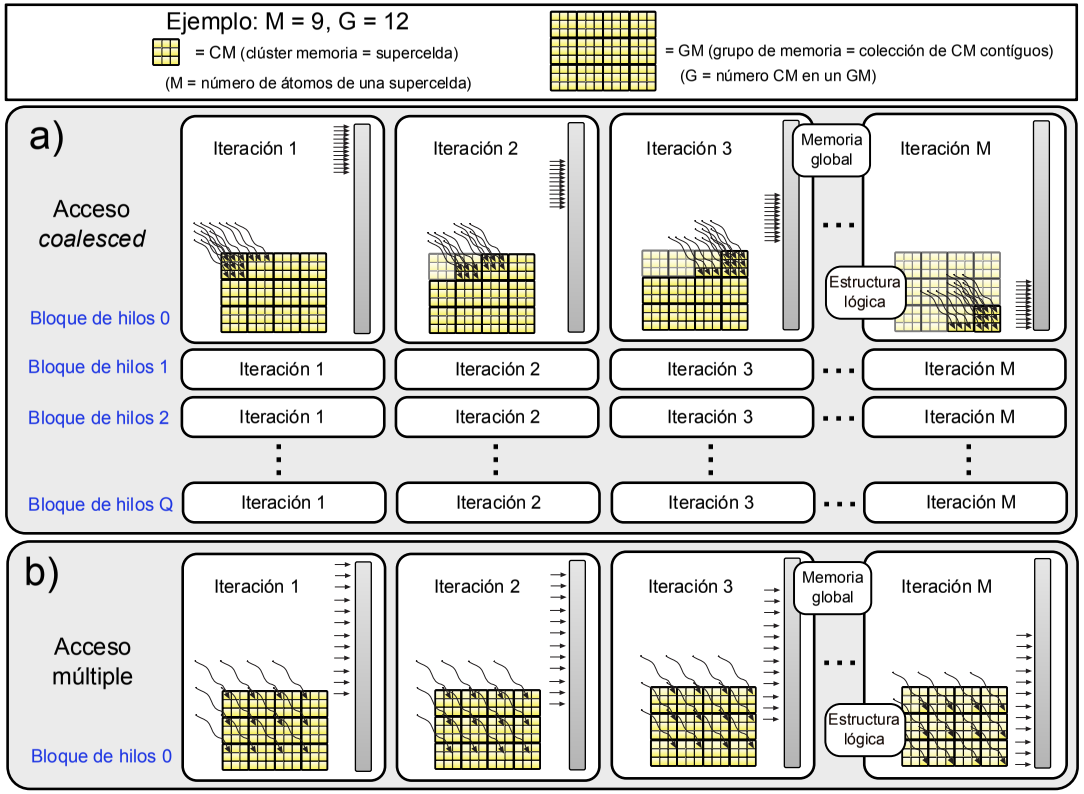

Figura 4.4: Representación gráfica de un Grupo de Memoria (GM): (a) esquema de implementación que permite un acceso *coalesced* a la memoria global de la GPU, (b) un ejemplo de acceso no unificado o *uncoalesced*.

La estructura de computación basada en dos *kernels* es utilizada para reducir la cantidad de accesos a la memoria global, como se ha explicado en el contexto de la ecuación 4.1. El primer *kernel* actualiza los datos relativos a las vecindades y el segundo la velocidad de extracción.

Debido a que cada hilo procesa un átomo superficial en cada iteración del bucle, los accesos a la memoria global pueden ser fácilmente optimizados para obtener un acceso *coalesced*, por ejemplo haciendo que hilos contiguos procesen átomos contiguos. Sin embargo, las lecturas de datos referentes a átomos vecinos son más complejas de optimizar. Esto es debido a dos factores:

1. El conjunto de átomos contenido en las superceldas varía en tamaño y estructura interna dependiendo de la orientación cristalográfica simulada.

2. Las posiciones de los CMs donde están los átomos vecinos no son conocidas *a priori* debido a que son escogidas en tiempo de ejecución (los CMs son liberados y asignados de forma dinámica durante la simulación).

Para átomos localizados en los bordes de las superceldas, es probable que los átomos vecinos (en el espacio 3D) estén almacenados en CMs distantes (en la GPU). Por otro lado, para los átomos localizados en el interior de la supercelda, los vecinos están localizados en el mismo CM. En este contexto, se ha recurrido a la utilización de la unidad de textura para realizar lecturas de las variables *Syb*, *N1sb* y *TArea*. Debido a la localidad espacial en memoria existente entre las células del mismo CM, la utilización de las cachés de textura proporciona una mejora de velocidad sobre los accesos puramente *uncoalesced* a la memoria global.

La parte b) de la figura 4.5 tiene el objetivo de actualizar la ocupación de los átomos superficiales y monitorizar cuáles de ellos llegan a una ocupación de valor cero. La eliminación de estos átomos provee el origen atomístico a la propagación macroscópica de la superficie. Asimismo, los CMs cuyos átomos han sido atacados en su totalidad son marcados para su reutilización y/o liberación de la memoria de la GPU. Los pasos (1)-(3)

siguen un procedimiento similar a la parte a). Para este caso, cada hilo comprueba si el átomo procesado es superficial leyendo la variable *Syb*. Si es el caso, se aplica la ecuación 2.6 para reducir la ocupación (almacenada en *Occ* usando el valor la velocidad de extracción (*Erate*). Si la ocupación se ha reducido a cero, entonces:

1. El *flag modificado* (almacenado en *Syb*) se activa para los primeros y segundos vecinos del átomo en cuestión.
2. El estado de los primeros vecinos se modifica de *enterrado* a *superficial*.
3. El contador *MCstate* se reduce en una unidad. Este contador monitoriza la cantidad de átomos no eliminados en el CM correspondiente.

Cuando $MCstate = 0$, el CM ha sido totalmente procesado y, por tanto, puede ser liberado y reutilizado para almacenar otra supercelda.

A primera vista, la escritura a los *flags modificado* puede dar lugar a accesos *uncoalesced* y por tanto, muy costosos. Sin embargo, la acción de activar los *flags* se realiza sólo para una pequeña fracción de átomos en comparación con la gran cantidad de reducción de ocupaciones necesarias, resultando en un patrón de acceso a memoria de baja intensidad. Por otro lado, varios de los átomos vecinos pueden ser eliminados en el mismo paso de tiempo, escribiendo por lo tanto múltiples veces el flag modificado para cierto átomo. La utilización de memoria compartida como *buffer* para prealmacenar los *flags* de un MG previo a su escritura en memoria global puede prevenir la activación múltiple del mismo *flag*, reduciendo por tanto los accesos globales a memoria.

Finalmente, la parte c) de la figura 4.5 efectúa las computaciones requeridas relacionadas con la gestión de la superficie. Debido a que el estado de la vecindad es utilizado para determinar la velocidad de extracción de los átomos, los átomos localizados en los límites de las superceldas deben poder acceder a sus vecinos. Por lo tanto, el algoritmo debe asegurar que una supercelda que contenga al menos un átomo superficial puede acceder, en cualquier caso, a las superceldas vecinas en el espacio. Debido a que cada supercelda está almacenada en un CM, este acceso a la vecindad se

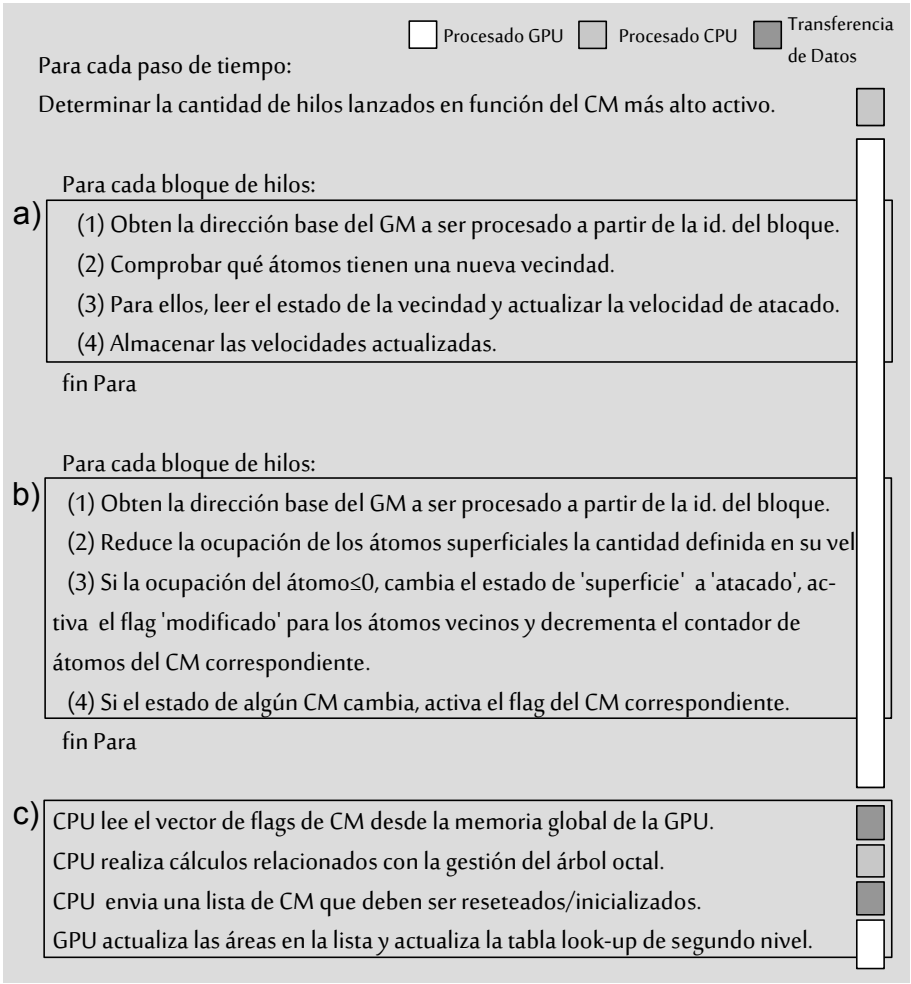

Figura 4.5: Pseudocódigo de la implementación paralela del ACC. (a) Actualización de vecindades y velocidades de extracción. (b) Reducción de ocupaciones y eliminación de átomos. (c) Administración de la superficie.

consigue obligando a que los correspondientes CMs adyacentes en el espacio hayan sido asignados. Para poder cumplir este objetivo, el vector de *flags* utilizado para indicar los estados de los CMs (el cual es leído por la CPU) debe indicar los siguientes eventos:

- No es necesario realizar ninguna acción (00).
- Por primera vez, un átomo enterrado ha pasado a ser un átomo superficial en la supercelda. El atacado en esta supercelda ha comenzado, por lo que las superceldas vecinas en el espacio deben estar asignadas en la GPU (01).
- Todos los átomos en la supercelda han sido eliminados y las superceldas vecinas no poseen ningún átomo superficial, por lo que la presente supercelda puede ser eliminada (10).

Cuando un evento es detectado (casos 01 y 10), el hilo correspondiente activa el *flag* del CM, el cual es almacenado de forma temporal en *TBuf*. La CPU analizará *TBuf* en cada paso de tiempo, realizará las operaciones correspondientes para la gestión del árbol octal e informará a la GPU sobre (i) la necesaria inicialización de ciertos CM, y (ii) actualizaciones en la tabla *look-up* de segundo nivel. Este proceso requiere que un total de dos bits de información por CM deban ser enviados de la GPU a la CPU en cada paso de tiempo. Pese a que el tamaño de esta trasferencia parece muy pequeño, la cantidad total de datos transferidos depende directamente del tamaño de la supercelda. Por lo tanto, la definición de la supercelda debe ser un equilibrio entre (i) reducir el número de átomos procesados y almacenados de forma innecesaria, conseguido usando superceldas más pequeñas, y (ii) prevenir excesivas trasferencias GPU-CPU, así como la creación de un árbol demasiado complejo que resulte en tiempos de gestión demasiado altos. El efecto del tamaño de la supercelda en la eficiencia de la implementación es analizado en el siguiente apartado.

El último objetivo de la parte c) de la figura 4.5 es el de actualizar la tabla *look-up* de segundo nivel almacenada en la GPU e inicializar los CMs asignados en el presente paso de simulación. El *kernel* de la GPU que efectúa estas funciones recibe a través de *TBuf* (i) una lista de CMs eliminados y asignados y (ii) el nuevo conjunto de punteros vecinos para cada CM actualizado. Los hilos que ejecutan este *kernel* leen datos de esta

lista y actualizan las posiciones correspondientes de la tabla *look-up*, una entrada por hilo. Asimismo, si el CM en cuestión debe ser inicializado, el propio hilo es el que realizará esta acción, fijando el tipo de todos los átomos a *enterrado* y su ocupación a 1.0. El patrón de acceso a memoria para la actualización de la tabla *look-up* y la inicialización de los CMs no está especialmente optimizado en nuestra implementación. Sin embargo, debido a que la cantidad de CMs actualizados por paso de tiempo es pequeño, no afecta al rendimiento final del algoritmo.

4.1.4. Validación del Algoritmo

En la presente sección se realiza una serie de pruebas para validar la implementación propuesta del ACC sobre una arquitectura GPU. Nuestro objetivo es caraterizar el efecto de la aplicación del árbol octal a la simulación del ACC en la GPU, así como obtener detalles significativos de la simulación, tales como la distribución de tiempos de cómputo o escalabilidad del algoritmo.

4.1.4.1. Descripción de las Pruebas

Los principales objetivos de los tests realizados son:

1. Estudiar el impacto de la utilización del árbol octal por parte del algoritmo en el tiempo global de simulación y la ocupación de la memoria global.
2. Analizar en detalle los tiempos de ejecución de los distintos *kernels* utilizados en el bucle de ejecución.
3. Describir la escalabilidad de la implementación propuesta como una función del tamaño de los sistemas simulados.
4. Presentar tiempos de ejecución típicos para un amplio rango de superficies.

Para la comparación de resultados computacionales, se ha simulado 2.000 pasos de tiempo de atacado químico de paso constante para una oblea

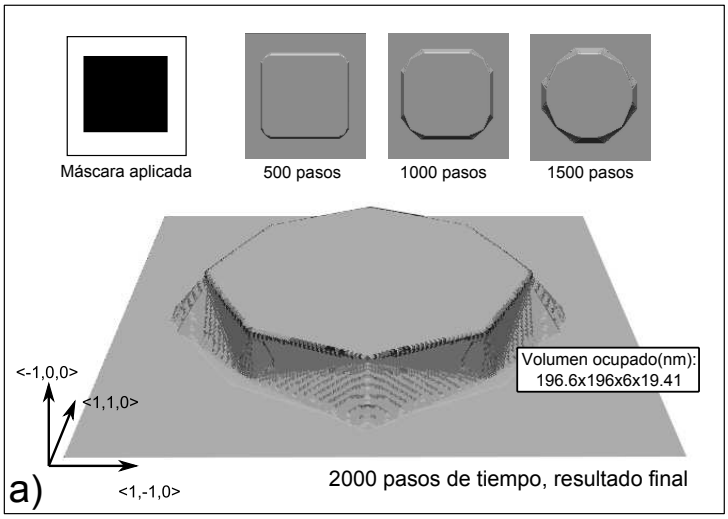

Figura 4.6: Estructura simulada para el estudio de la dependencia de la eficiencia de la implementación con el tamaño de la supercelda. 2.000 pasos de tiempo aplicados. Modelo calibrado utilizando KOH 40wt % 70°C.

de silicio de orientación <100> usando un patrón rectangular que cubre aproximadamente el 45 % de la superficie en una solución de KOH con una concentración del 40wt % a una temperatura de 70°C. La calibración del ACC ha sido obtenida a partir de [104]. En este experimento, el efecto de *underetching* aparece en las cuatro esquinas de la máscara. La figura 4.6 muestra la evolución de la superficie a lo largo de la simulación.

Con el objetivo de estudiar el impacto que el tamaño de la supercelda (y, por ende, de la utilización más o menos intensiva del árbol octal) en el tiempo computacional y la memoria utilizada (objetivo 1), utilizamos el hecho de que la partición mas pequeña en los nodos hoja del árbol corresponde con la celda unidad ortorrómbica del cristal de silicio. Utilizando múltiplos de esta celda unidad mínima podemos generar celdas unidad cada vez mayores y analizar el impacto del tamaño de los CMs tiene en la velocidad y ocupación de memoria del algoritmo. Analizaremos parámetros relevantes como el tiempo de ejecución, el porcentaje de uso de la memoria

global y el volumen de transferencia entre la CPU y la GPU, así como la cantidad de instrucciones ejecutadas por la GPU.

Para el caso más óptimo de supercelda, mostraremos un análisis detallado del coste computacional de todos los *kernels* lanzados en la GPU, viendo cualitativamente el coste de cada paso del bucle de simulación (objetivo 2).

Para poder determinar la escalabilidad de la implementación propuesta (objetivo 3), se han simulado tamaños de superficies en el rango de $2^{16} \approx 6.6 \times 10^4$ hasta $2^{24} \approx 1.7 \times 10^7$ átomos de silicio, doblando el tamaño de la superficie en cada simulación. Para poder mantener la forma de la estructura 3D resultante (figura 4.6), la profundidad de atacado necesita ser incrementada por un factor de $\sqrt{2}$ por cada duplicación de superficie. Esto resulta en un incremento de volumen ocupado por el sistema de $\sqrt{2} \times \sqrt{2} \times \sqrt{2} = 2\sqrt{2} = 2.83$ por duplicación de superficie. El volumen de silicio eliminado por el proceso está directamente relacionado con la cantidad de átomos eliminados y, por lo tanto, con la cantidad de cálculos que la GPU debe realizar.

Por último, también presentamos tiempos de ejecución típicos para la implementación paralela propuesta del ACC aplicado a un total de seis sistemas diferentes (objetivo 4), comparando los resultados con la implementación secuencial previamente existente del modelo ACC: VisualTAPAS. Los seis sistemas diferentes, atacados con KOH 30wt % a 80 C son descritos y simulados con VisualTAPAS en la referencia [128]. Cada sistema contiene una máscara diferente la cual posee distintas relaciones de aspecto y presentan orientaciones a lo largo de diferentes orientaciones cristalográficas. Esto sirve para comprobar la corrección del algoritmo basado en GPU y determinar qué mejoras importantes de rendimiento pueden obtenerse independientemente de la forma resultante de las superficies.

El hardware utilizado para la simulación consiste en una GPU Nvidia 9800GT con 512MBytes de memoria y una CPU Intel Core i7 a 2.66 GHz con 3 GB de PC1333 DDR3 SDRAM.

4.1.4.2. Resultados

Los datos de la tabla 4.2 muestran la mejora en varios aspectos del rendimiento del algoritmo ejecutado en la GPU cuando el tamaño de la supercelda se reduce. Esto caracteriza el efecto de separar la memoria de la GPU en CMs más pequeños. Una disminución del tamaño desde 4096 átomos (última fila) conduce a una fuerte reducción en la cantidad de memoria ocupada por la superficie (columna 7). Debido a la reducción del tamaño de la supercelda, la región activa del espacio puede ser definida de una forma más precisa por el árbol octal, evitando por tanto el almacenamiento de muchos átomos enterrados o ya atacados, que no son útiles debido a estar lejos de la interfase. Este efecto está también presente en el número total de instrucciones ejecutadas por la GPU (columna 5). Por otro lado, el tamaño de la supercelda no afecta de forma significativa el rendimiento del algoritmo GPU, manteniendo un flujo de instrucciones por ciclo constante, rondando los 0.5 IPC (columna 6). Reduciendo el tamaño de los CMs mientras se mantiene la eficiencia del algoritmo lleva a una reducción en los tiempos totales de simulación de la superficie (columna 3). De acuerdo con los resultados de tiempos, el tamaño de supercelda óptimo para el modelo implementado contiene 64 átomos. Reduciendo el tamaño de la supercelda más allá ralentiza las simulaciones. Esto es debido a tres efectos:

1. La reducción en la cantidad de átomos no útiles procesados es ya poco significativa.

2. La complejidad del árbol y la cantidad de modificaciones relacionadas es lo suficientemente grande para que el tiempo de proceso de la CPU sea relevante, como se muestra en la figura 4.7 (a).

3. Superceldas muy pequeñas requieren un acceso más intensivo a las tablas *look-up* almacenadas en la memoria global de la GPU.

Para superceldas de tamaño 4 y 8, el tercer efecto se observa claramente en la tabla 4.2 como un incremento de instrucciones realizadas por la GPU y una reducción en el flujo de instrucciones por ciclo.

La figura 4.7 (b) muestra, para una supercelda de 16 átomos, la distribución de los tiempos de computación de la GPU en las distintas tareas asignadas. La tarea más costosa es la relacionada con la reducción de ocupación de los átomos y la notificación de nuevas vecindades a los primeros y segundos átomos . Las tareas no relacionadas directamente con el procesado del modelo, tales como las trasferencia de datos o el mantenimiento de la superficie, ocupan únicamente un pequeño porcentaje del tiempo total.

La escalabilidad de la implementación presentada es mostrada en la figura 4.7 (c). Como se ha explicado en el apartado anterior, doblar el tamaño de la superficie simulada representa un incremento teórico de instrucciones a realizar de $2\sqrt{2}$. El factor de incremento del tiempo computacional por doblado de superficie (T_j/T_{j-1}) es mostrado usando marcadores con forma de diamante. Estos valores son obtenidos dividiendo el tiempo computacional obtenido para cierto tamaño (*ej.* $j = 7$ para 4096×1024 átomos) por el tiempo computacional para el tamaño previo (*ej.* $j = 6$ para 2048×1024 átomos). Estos valores deberían alcanzar el valor teórico ($2\sqrt{2}$). Valores inferiores indican que el algoritmo se ejecuta de forma más eficiente cuando se aplica a una superficie más grande. Por otro lado, valores mayores del valor teórico evidenciarían que el rendimiento de algún fragmento del algoritmo se ve afectado por el tamaño de la superficie, suponiendo un problema grave para escalar el simulador a simulaciones con álto nivel de detalle.

En la gráfica mostrada hay una tendencia de las superficies pequeñas a no ser ejecutadas de forma totalmente eficiente en la GPU. La razón de que esto suceda es debido a que es necesario una gran cantidad de paralelismo en el algoritmo para poder mantener todos los procesadores de la GPU en funcionamiento y, por tanto, obtener una potencia de cálculo óptima. En nuestra implementación, el uso total de las capacidades de la GPU es obtenido con volúmenes iniciales de 2048×1024 átomos o más. La GPU está infrautilizada para superficies menores. Una cantidad de átomos tan grande es necesaria debido a que únicamente una pequeña fracción de los átomos es eliminada en cada paso de tiempo.

Tabla 4.2: Rendimiento de la implementación basada en GPU para diferentes tamaños de supercelda. Cada simulación consiste en 2000 pasos de tiempo ejecutados sobre una GPU Nvidia 9800GT con 512 MBytes de memoria.

Tamaño supercelda (ÅxÅxÅ)	Átomos Si por supercelda	Tiempo de ejecución(s)	Flujo de datos GPU-CPU(MB)	GPU: Instrucciones ejecutadas	GPU: flujo de instrucciones (IPC)	GPU: uso de memoria global(%)
(3.84x3.84x5.43)	4	30.27	572.76	2.370×10^9	0.36	18.51
(7.68x3.84x5.43)	8	22.60	296.67	2.159×10^9	0.36	15.49
(7.68x7.68x5.43)	16	15.82	153.13	1.938×10^9	0.43	14.18
(7.68x7.68x10.86)	32	15.19	121.96	2.634×10^9	0.55	22.85
(15.36x7.68x10.86)	64	14.71	63.65	2.549×10^9	0.52	23.54
(15.36x15.36x10.86)	128	14.82	33.09	2.564×10^9	0.54	24.64
(15.36x15.36x21.72)	256	18.34	27.84	3.867×10^9	0.61	42.25
(15.36x30.72x21.72)	512	21.48	14.75	3.888×10^9	0.52	45.27
(30.72x30.72x21.72)	1024	23.19	7.57	3.827×10^9	0.48	47.94
(30.72x30.72x43.44)	2048	30.87	6.59	6.196×10^9	0.57	84.25
(30.72x61.44x43.44)	4096	37.38	3.45	6.701×10^9	0.51	91.33

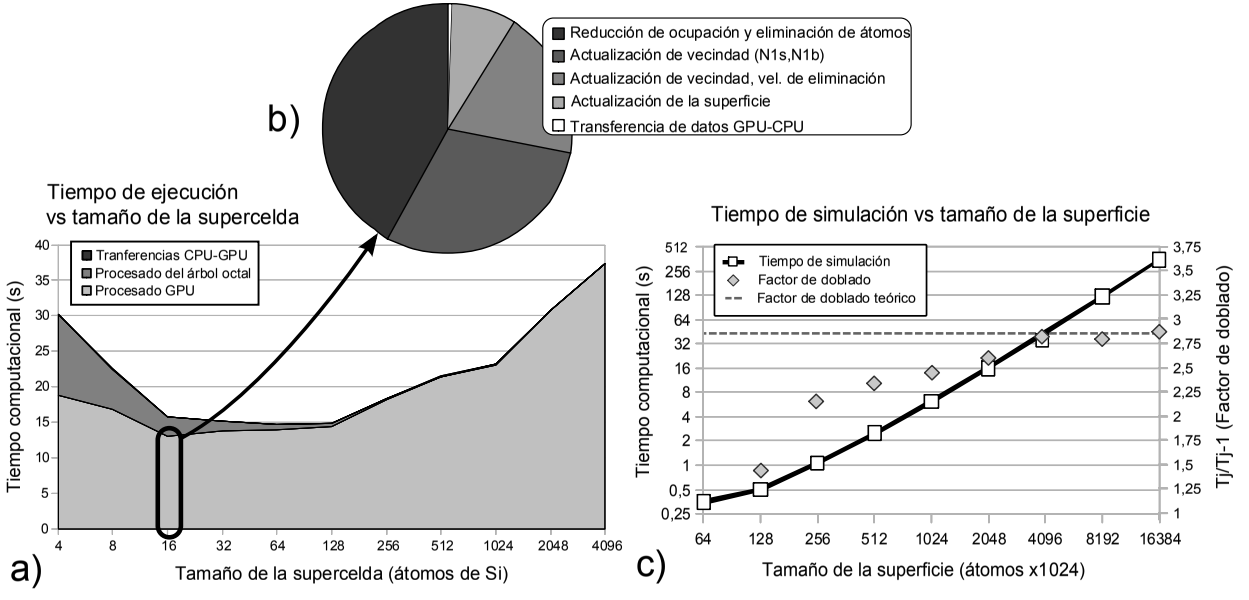

Figura 4.7: (a) Tiempos de simulación del experimento en función del tamaño de la supercelda, desglosados en computación GPU, computación CPU y trasferencia de datos. (b) Desglose del tiempo de ejecución en la GPU de las distintas tareas realizadas en el bucle de simulación. (c) Factor de incremento del tiempo computacional por duplicación del tamaño de la superficie (T_j/T_{j-1}).

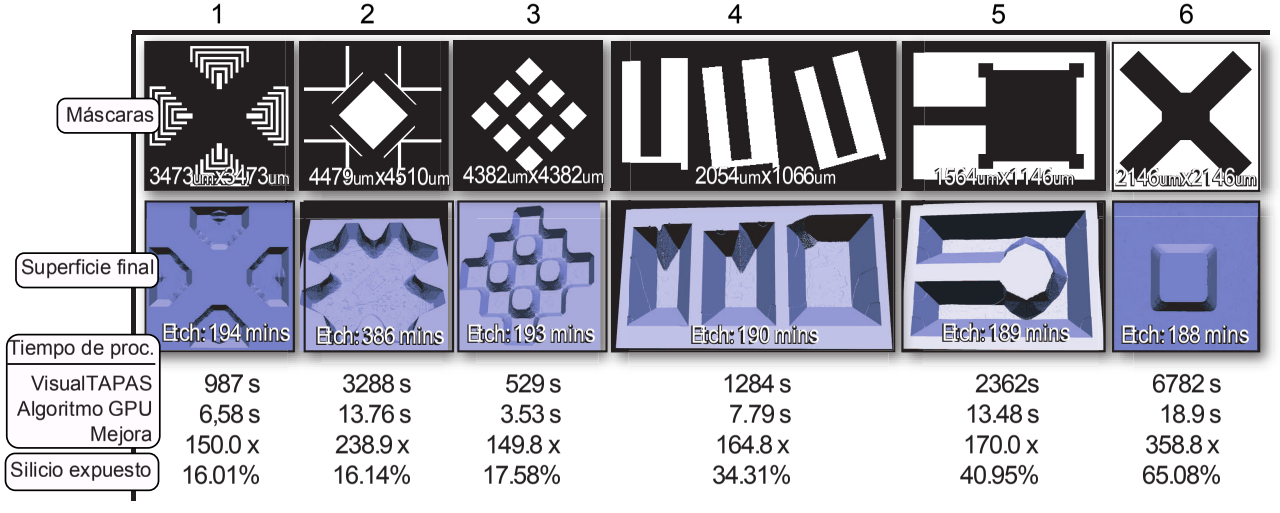

Figura 4.8: Comparación de velocidad entre el simulador VisualTAPAS [122] y el algoritmo propuesto basado en GPU para diferentes superficies. Los diseños de las máscaras obtenidos de [155]. Atacante simulado: KOH 30wt % a 80°C.

Una vez el punto óptimo se ha alcanzado, el incremento en el tiempo de proceso concuerda bien con los incrementos teóricos en el número de cálculos, demostrando de esta manera una buena escalabilidad de la implementación propuesta.

Finalmente, la figura 4.8 demuestra que, para un amplio rango de diferentes diseños de máscaras, las estructuras obtenidas con la implementación del ACC basada en GPU son esencialmente idénticas a las obtenidas con una implementación totalmente secuencial del diseño (VisualTAPAS) [128]. Los resultados evidencian que el algoritmo basado en GPU es al menos dos órdenes de magnitud más rápido que el código puramente secuencial para todos los casos. Las diferencias entre los factores de incremento de velocidad entre ambas implementaciones son debidas al número medio de eliminaciones de átomos por paso de tiempo. Máscaras que dejan más silicio expuesto dan lugar a mayores velocidades de extracción de átomos, resultando en un comportamiento más paralelo del algoritmo y una mejor adaptación a la GPU.

En conclusión, los resultados muestran que la utilización de árboles octales con la implementación propuesta no sólo mejora el rendimiento de simulación de superficies dinámicas en GPUs, sino que también reduce la utilización de memoria. Asimismo, nuestra implementación es varios órdenes de magnitud más rápida que los simuladores secuenciales actualmente disponibles, reduciendo los tiempos de simulación de minutos a segundos. Esta característica es de mucha utilidad debido a que tradicionalmente es necesario simular un procedimiento una gran cantidad de veces hasta obtener los parámetros óptimos.

4.2. Aplicación de la Implementación: Simulador GPUetch

La implementación propuesta y los algoritmos GPU desarrollados en la sección anterior han sido utilizados como núcleo interno para la creación de un nuevo simulador de atacado anisótropo húmedo denominado *GPUetch*.

GPUetch es presentado como una alternativa válida a los simuladores

comerciales y educativos actualmente disponibles, ofreciendo una serie de funciones necesarias para simular la mayoría de procesos de fabricación con MEMS:

- Simulación de una gran cantidad de orientaciones del Silicio.
- Posibilidad de aplicar máscaras de óxido de silicio, nitruro de silicio, así como capas sacrificiales de polisilicio.
- Posibilidad de simular de forma aproximada atacado DRIE, así como simulación de atacado isótropo.
- Simulación de atacado a doble cara y obleas SOI.
- Una librería con una gran cantidad de calibraciones de distintos atacantes, obtenidos por el método explicado en el capítulo 5.
- Un entorno de visualización gráfica.
- Herramientas de medida de distancias en la estructura final
- Caracterización del atacante, obteniendo las velocidades de atacado para un amplio rango de orientaciones.

GPUetch ya ha sido utilizado para la predicción de diversas estructuras, como por ejemplo la fabricación de microagujas mediante TMAH + Triton [156], ofreciendo resultados acorde con el experimento, y por tanto, siendo de utilidad para la obtención de una nueva técnica de microfabricación. Asimismo, GPUetch se ha utilizado como herramienta base para una nueva metodología de calibración basada en *algoritmos genéticos* (capítulo 5), la cual requiere una elevada cantidad de simulaciones para obtener una solución adecuada. Con los simuladores tradicionales, este método era inabarcable computacionalmente.

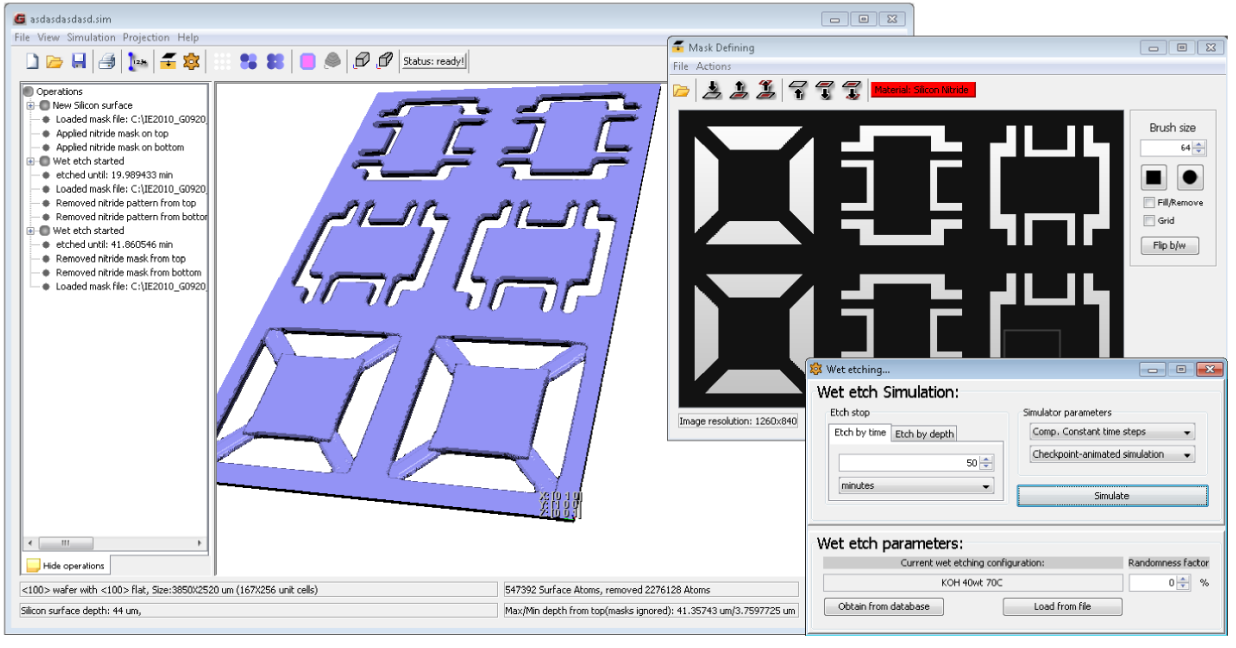

Figura 4.9: Captura de pantalla de GPUetch. (a) Panel principal, donde se presenta una vista 3D de la estructura, el historial de acciones e información acerca de la superficie simulada. (b) Menú de máscara: permite aplicar capas para la protección del silicio, así como capas sacrificiales. (c) Menú de atacado húmedo: controla la simulación del proceso, permitiendo la modificación de parámetros.

La figura 4.9 muestra una imagen del software en el proceso de diseño de un microacelerómetro de tres ejes [93]. Las ventanas que se muestran son el panel principal donde se muestra una vista 3D del substrato de silicio (a), el menú de máscaras, donde se puede simular las transferencia de máscaras al silicio con materiales resistentes al atacado y el menú de simulado, el cual permite manejar opciones del atacado tales como el tiempo, el tipo atacante o la precisión del modelo (c).

4.3. Conclusiones

La disponibilidad y el desarrollo constante de arquitecturas *Many-Core* en los últimos años ha abierto la exploración de nuevas implementaciones para problemas previamente existentes. Un ejemplo de este nuevo tipo de arquitecturas, las GPUs, sobresalen como una opción atractiva para un amplio rango de aplicaciones científicas. En este capítulo demostramos que la superficies dinámicas pueden ser almacenadas y simuladas de forma eficiente usando árboles octales como estructura de soporte, la cual previene el almacenamiento/procesado de datos no útiles por parte de la GPU.

La conclusión de este capítulo es que es posible obtener una simulación eficiente de una superficie dinámica por parte de la GPU realizando un desacoplo de la gestión de la superficie y los cálculos del modelo: (i) almacenando todo los datos del modelo y realizando todas cálculos relacionados con la evolución de la superficie en la GPU, y (ii) almacenando al árbol octal en la CPU, la cual realiza también todos los cálculos relacionados con su gestión.

La aplicación de la implementación propuesta ha sido llevada a cabo para el modelo ACC el cual modela el grabado anisótropo, uno de los procesos de microfabricación de MEMS basados en silicio más utilizado. Las pruebas sobre el algoritmo implementado han mostrando buenos resultados: ajustando el tamaño óptimo de la supercelda es posible reducir en gran medida el tiempo de simulación con respecto a una peor (o inexistente) partición del espacio. Asimismo, para la configuración óptima de supercelda, la mayoría del tiempo de proceso es utilizado en los cálculos propios del modelo, mientras que la administración de la superficie apenas

representa una pequeña fracción del cómputo total.

Como resultado de este estudio, la implementación GPU no sólo permite la realización de simulaciones de grabado anisótropo en tiempos menores, sino que también hace posible la simulación de los mismos sistemas con una mayor resolución.

Pese a que la implementación GPU-CPU ha sido probada únicamente con GPUs de Nvidia en este estudio, las guías de diseño presentadas son muy generales y deberían ser válidas para otras arquitecturas *Many-Core*. La característica más importante que hace útil la presente implementación es la falta de un mecanismo global de exclusión. Las GPUs son un claro ejemplo de esto, debido a que el mecanismo de exclusión está basado en operaciones atómicas sobre la memoria global, lo que es muy lento y contrario a la filosofía de programación GPU. De cualquier manera, en el caso de que futuras generaciones sean capaces de manejar eficientemente una estructura basada en un árbol octal, el programador puede portar dicho fragmento de código de la CPU a la GPU, dejando el resto del algoritmo intacto. En este caso, nuestro algoritmo propuesto para el manejo de la memoria GPU para una superficie dinámica sigue siendo válido.

Finalmente, creemos que la implementación propuesta no está limitada únicamente a autómatas celulares, sino que puede ser válido para optimizar otras técnicas computacionales, en especial los métodos *level-set* los cuales, al igual que el ACC, son utilizados para definir un frente que avanza de acuerdo a unos parámetros fijados. Los métodos *level-set*, que actualmente han sido acelerados en GPUs utilizando distintas técnicas [157, 158], podrían hacer uso fácilmente de los árboles octales como estructura de datos de soporte para reducir la ocupación en memoria del modelo, así como mejorar la velocidad de simulación.

Capítulo 5

Calibración del Autómata Celular Continuo Mediante Algoritmos Evolutivos

Con la reciente aparición de nuevos atacantes para el proceso de atacado anisótropo húmedo, se ha abierto un nuevo abanico de microestructuras posibles, incrementando así la utilidad de este procedimiento. Las nuevas disoluciones más relevantes, las cuales ofrecen características muy interesantes tales como la reducción del *underetching*, son la adición de alcohol isopropílico a una solución basada en KOH o la adición del surfactante Triton-X 100 a una solución basada en TMAH (sección 2.3.3). La calibración del ACC para estos nuevos atacantes supone un desafío debido a que los métodos tradicionales de calibración no ofrecen un resultado aceptable.

En el presente capítulo mostramos una nueva metodología que pretende superar dicha limitación, posibilitando calibraciones del ACC más precisas, así como la posibilidad de poder simular procesos basados en nuevas soluciones.

5.1. Introducción, Problemática de la Calibración del ACC

La calibración actualmente propuesta para el ACC consiste en dos partes:

- Para los átomos cuya configuración de vecindad está clasificada, y por tanto es conocida su participación en el proceso de *step-flow* de las orientaciones estudiadas por M. A. Gosalvez, se resuelve el sistema de ecuaciones introduciendo como parámetros datos de la velocidad del atacante para distintas orientaciones (sección 2.3.4.1).
- Para los átomos cuya vecindad no está clasificada de aplica la Función de Probabilidad de Extracción (RPF, del inglés *Removal Probability Function*)

La RPF es una función utilizada para fijar la velocidad de atacado en función de la cantidad de primeros y segundos vecinos del átomo, la cual pretende modelar la relación entre las velocidades de atacado y la cobertura de primeros y segundos vecinos por grupos OH [159]:

$$p(n_1, n_2) = p_0 \frac{1}{1 + e^{\beta\epsilon_1(n_1 - n_1^0)}} \frac{1}{1 + e^{\beta\epsilon_2(n_2 - n_2^0)}} \tag{5.1}$$

donde β, ϵ_1, ϵ_2, n_1^0 y n_2^0 son parámetros de la ecuación que son fijados en función de las características del atacante simulado.

Pese a que este procedimiento es capaz de calibrar el ACC satisfactoriamente para atacantes tradicionales como los basados en KOH, no es capaz de modelar de forma adecuada los efectos de la incorporación de aditivos en la solución. La figura 5.1 muestra este hecho comparando resultados experimentales con la simulación mediante el ACC de los atacantes KOH 40wt % 70ºC y TMAH 25wt % +Triton 0.1 %vv a 80ºC. Los datos experimentales han sido obtenidos de [160], mientras que los datos relativos a las simulaciones se han realizado mediante el atacado de una esfera por GPUetch, utilizando una calibración obtenida mediante el método expli-

cado en este apartado. El simulador GPUetch es descrito en la sección 4.

En conclusión, el obtener un nuevo procedimiento de calibración para poder simular nuevos atacantes con el ACC, así como poder obtener simulaciones más precisas, es un objetivo de gran importancia. En las siguientes secciones enunciamos una nueva metodología que permite obtener calibraciones más precisas para cualquier tipo de solución utilizada en procesos de atacado anisótropo húmedo.

5.2. Algoritmos Genéticos y su Aplicación a ACs

Los *algoritmos genéticos* (AGs), desarrollados por J. Holland en los años 70 [161], son una metodología de búsqueda heurística la cual imita el proceso de evolución natural. Los AGs son utilizados actualmente como un método para obtener soluciones a problemas de optimización y búsqueda. La idea subyacente detrás de este algoritmo es la de que, dada una población de individuos, la presión del entorno ejerce una selección natural (supervivencia de los más aptos). Esto causa una adaptación de la población a las exigencias del entorno.

A lo largo de la ejecución de un AG, se mantiene una lista de elementos llamados *individuos*. Cada individuo representa una posible solución al problema que se desea optimizar, el cual está formado por un conjunto de variables, denominadas *genes*. Dichos genes son el objetivo de optimización por parte del AG. El conjunto de individuos es llamado *población*. El AG puede ser comprendido como un bucle donde en cada iteración se aplican ciertas operaciones sobre la población con el objetivo de optimizar los genes en función de ciertos parámetros fijados por el diseñador. Tradicionalmente, los pasos aplicados en cada iteración son los siguientes:

1. *Selección* de una cantidad de individuos de la población acorde a una función objetivo.

2. *Recombinación* de dichos individuos (*padres*) para la creación de nuevos individuos (*hijos*).

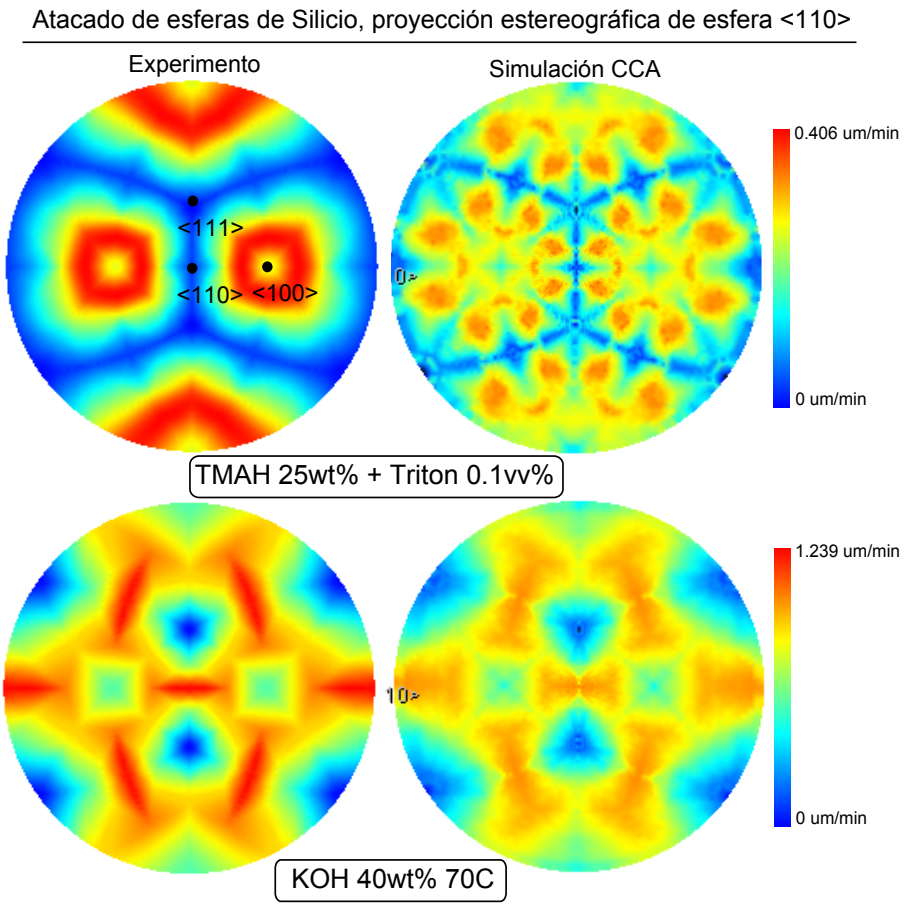

Figura 5.1: Resultados de calibración del ACC mediante la resolución del sistema de ecuaciones y la RPF. Datos experimentales obtenidos de [160]. Simulaciones realizadas sobre una esfera de silicio mediante GPUetch.

3. Aplicación de *mutaciones* en los genes de los hijos.

4. Generación de una población nueva a partir de la población existente y la *reinserción* de los hijos generados en la presente generación.

A medida que nuevas generaciones se suceden, aquellos individuos que se acercan más a la función objetivo tienen más posibilidades de tener descendencia, y por lo tanto, extender las bondades de sus genes. Por otro lado, hijos cuyos genes no sean adecuados, tienen menos probabilidades de que sus genes (y por lo tanto dicha solución al problema) se mantengan a lo largo de las generaciones. De forma independiente, las mutaciones aplicadas a los nuevos individuos buscan encontrar nuevas combinaciones de genes las cuales mejoren las características del individuo.

Los AGs son, por tanto, un algoritmo con un fuerte componente estocástico, de manera que ejecutar varias veces el mismo algoritmo con los mismos datos iniciales no garantiza llegar a la misma solución.

Es fácil ver que este tipo de algoritmos caen en la categoría de *generar y probar*. La función objetivo representa una estimación heurística de la calidad de la solución y el proceso de búsqueda es dirigido por las operaciones de variación y selección.

Los AGs han sido aplicados de forma satisfactoria a una gran cantidad de problemas, totalmente distintos en naturaleza, tales como diseño automatizado relacionado con el automovil [162], diseño de circuitos electrónicos [163] u optimización de rutas [164].

5.2.1. Aplicación de AGs sobre ACs

En muchos casos, la tarea definida por los ACs de encontrar las interrelaciones microscópicas entre células con el fin de obtener el comportamiento macroscópico adecuado es una tarea muy compleja. Por ejemplo, en el caso de la definición actual de ACC, pese a que la relación entre el mundo macroscópico y microscópico ha sido definida para un ámplio rango de orientaciones, existe todavía un conjunto de orientaciones más complejas, de las cuales se desconocen los detalles de la cinética química

relacionada con su atacado. Es en estos casos, donde la RPF se ha utilizado como aproximación.

Es en este punto donde la computación basada en algoritmos evolutivos aplicados sobre ACs puede resultar útil: mapear el comportamiento mascroscópico del ACC a reglas de evolución locales.

La utilización conjunta de AGs y ACs es una técnica introducida recientemente por N. H. Packard et al. en 1990 [165, 166]. La primera publicación relevante, responsable de hacer los AGs una herramienta popular para obtener las reglas de evolución de los ACs es realizada por M. Mitchell *et al.* en el año 1993 [167]. El artículo sostenía la viabilidad de utilizar el modelo llamado *Autómata Celular Evolutivo* para resolver la tarea de clasificación por densidad.

Finalmente, la utilización de esta técnica para la obtención de reglas de evolución no excluye la filosofía tradicional de modelado de ACs, sino que puede utilizarse de forma conjunta con la definición de reglas microscópicas del AC por parte del diseñador. Añadir restricciones en forma de reglas microscópicas predefinidas tiende a reducir de forma significativa el espacio de búsqueda del AG, acelerando su velocidad de convergencia.

Reducir el campo de búsqueda es habitualmente de gran ayuda a la hora de aplicar AGs. Esto es debido a que los procesos de evaluación de las funciones objetivo pueden ser, en muchos casos, costosos computacionalmente. Este hecho unido a la necesidad de realizar una gran cantidad de iteraciones hasta obtener una solución óptima, puede hacer que la utilización de AGs sea inabarcable desde un punto de vista computacional.

5.3. Algoritmo Genético para la Calibración del ACC

Hasta la implementación sobre GPUs de un simulador del ACC para el atacado anisótropo húmedo (capítulo 4), la aplicación de AGs sobre el ACC era una tarea inabarcable desde el punto de vista computacional. La posibilidad de realizar simulaciones en un muy corto espacio de tiempo ha permitido incluir los resultados de simulación dentro de las funciones

objetivo del AG con el fin de evaluar la calidad de una posible combinación de velocidades de reducción de ocupación.

En la presente sección explicamos todos los detalles del AG implementado el cual, con una población de 150 individuos, permite calibrar el ACC a partir de datos experimentales. Estos detalles comprenden la definición de la función objetivo, así como la metodología para la selección, recombinación, mutación y reinserción.

5.3.1. Dominio de la Búsqueda y Estado Inicial

El AG propuesto actuará sobre el ACC existente, por lo que tanto la forma del retículo como la vecindad son fijadas por definición. Las variables a optimizar (y por tanto, los genes de cada individuo) serán las velocidades de reducción de ocupación para cada una de las posibles configuraciones de vecindad. Pese a que teóricamente el ACC posee 4096 combinaciones distintas de vecindad, la mayoría carecen de sentido físico. Para una simulación real, tan sólo de 150 a 350 configuraciones aparecen en las simulaciones.

Con el objetivo de reducir el dominio de la búsqueda, ciertas restricciones son añadidas al algoritmo, las cuales tienen que ver con el cumplimiento de las ecuaciones definidas en el modelo analítico del ACC y con el correcto funcionamiento del *step-flow*.

Estas restricciones son las siguientes:

- Las velocidades de extracción de los átomos con cuatro enlaces de hidrógeno, así como las configuraciones $(n^{1s}, n^{1b}, n^{2s}, n^{2b})$ con sólo un enlace covalente (*lollies*) $(0, 1, x, x)$ y $(1, 0, x, x)$ se mantienen fijas con valor 1.0, acorde al método de calibración tradicional [127].

- Las velocidades de extracción de las configuraciones $(n^{1s}, n^{1b}, n^{2s}, n^{2b})$: $(0, 2, 4, 4)$, $(2, 1, 1, 5)$ y $(0, 3, 6, 3)$ permanecen ancladas en ciertos valores de acuerdo con las velocidades experimentales de los planos (100), (110) y (111) respectivamente. Esto es posible debido a la simplicidad de las ecuaciones que unen el comportamiento macroscópico y microscópico para estas tres orientaciones [127]:

$$R_{100} = \frac{a}{4} r_{(0,2,4,4)} \tag{5.2}$$

$$R_{110} = \frac{a}{2\sqrt{2}} r_{(2,1,1,5)} \tag{5.3}$$

$$R_{111} = \frac{a}{\sqrt{3}} \frac{r_{(0,3,6,3)} r_{(0,1,6,3)}}{r_{(0,3,6,3)} + r_{(0,1,6,3)}} \tag{5.4}$$

donde $r_{(a,b,c,d)}$ es la velocidad de reducción de la ocupación para el átomo con vecindad (a, b, c, d) y a representa una distancia de $5.43\mathring{A}$

- Las configuraciones que forman parte de los genes de los individuos son escogidos mediante el uso de un histograma. Dicho histograma almacena las configuraciones observadas en las simulaciones durante las últimas 10 generaciones, las configuraciones que no aparecen se eliminan de los genes de los individuos, reduciendo así las variables a optimizar.
- Las funciones objetivo poseen factores para favorecer el *step-flow* (sección 5.3.2)

En cuanto al estado inicial de la población, teniendo en cuenta las restricciones anteriormente enunciadas, los genes de inicializan de forma aleatoria mediante la siguiente ecuación:

$$r_{(n^{1s},n^{1b},n^{2s},n^{2b})} = r_{(0,2,4,4)} + Rnd \tag{5.5}$$

Rnd es una variable aleatoria que puede tomar valores en el rango $[-0.125, 0.125]$. Asimismo, $r_{(0,2,4,4)}$ posee habitualmente una velocidad de extracción media con respecto el resto de configuraciones, lo que sirve como centro de la distribución aleatoria.

5.3.2. Funciones Objetivo

Las funciones objetivos sirven de guía al AG para obtener los genes óptimos que hacen a la población poseer el comportamiento deseado. La

función objetivo aplicada a un individuo debe indicar de forma adecuada cómo de bueno o malo es ese individuo en función a las características exigidas. En muchos casos, obtener una función objetivo que evalúe de forma adecuada la idoneidad de un individuo es una de las tareas más complicadas a la hora de definir un AG.

Para el caso del ACC, con el propósito de la obtención de las velocidades de extracción adecuadas para simular un determinado atacante, nuestra función objetivo es ponderada por cuatro efectos distintos:

$$\begin{aligned} E^k &= c_1 E_1'^k + c_2 E_2'^k + c_3 E_3'^k + c_4 E_4'^k, \\ &\text{donde} \quad E_i'^k = E_i^k / E_i^{min}. \end{aligned} \tag{5.6}$$

Las variables E_i^k representan el resultado obtenido de aplicar cierta función que evalúa en qué intensidad aparece el efecto i sobre el individuo k. Estos efectos son:

1. Diferencias en la morfología resultante tras el atacado de una esfera entre el experimento y la simulación realizada con el individuo.

2. Defectos característicos de la estructura al realizar un atacado sobre una superficie (100) aplicando una máscara circular.

3. Similitudes en las velocidades de atacado entre configuraciones con primeras vecindades $(n1s, n1b)$ distintas.

4. Variación en la velocidad de atacado entre configuraciones con primeras vecindades similares.

Estas cuatro características, que miden los posibles defectos que posee el individuo en cuestión, deberán ser minimizadas por el AG, por lo que valores de E^k menores representan genes más óptimos.

Para poder ponderar los cuatro parámetros E_i^k cuya naturaleza es completamente distinta, sus valores son normalizados con respecto al mejor (menor) valor obtenido para toda la población E_i^{min} por lo que, para cualquier individuo, el resultado normalizado para cada efecto puede tener un valor mínimo de 1 (mejor caso) y tender a infinito (peor caso).

Finalmente, el conjunto de coeficientes c_i es utilizado para ponderar con distintos pesos los cuatro coeficientes. Un valor mayor para uno de los coeficientes otorga mayor importancia a la característica correspondiente y por tanto el AG tiende a optimizar dicho efecto sobre los otros tres. El valor de coeficientes utilizado que mejores resultados ha dado para nuestros experimentos ha sido el siguiente: $\{c_1, c_2, c_3, c_4\} = \{0.55, 0.35, 0.15, 0.05\}$, aunque pequeñas variaciones pueden ayudar a reducir con más intensidad alguno de los efectos.

A continuación describimos al detalle las funciones propuestas para evaluar la existencia de cada uno de los efectos E_i^k.

5.3.2.1. Diferencias en la Morfología en el Atacado de una Esfera

Uno de los parámetros más importantes a minimizar es la variación de la forma resultante tras el atacado a una esfera de silicio, donde las velocidades de atacado de todos los ángulos posibles de silicio son expuestas. Como se explicó en la sección 2.3.3, el atacado de una esfera de silicio es una de las metodologías tradicionales para caracterizar la anisotropía de un atacante, por lo que la minimización de este error forzará que los individuos simulen una anisotropía similar a la del experimento.

Para obtener la caracterización de cierto individuo en relación con el atacado de una esfera, se utiliza la aplicación *GPUetch* (sección 4.2) donde, para la calibración definida para cierto individuo, se realiza un atacado a una semiesfera de 22 mm (figura 5.2 (a)), alcanzando una profundidad máxima de atacado de 400 um. La superficie de la esfera está formada por 259.516 átomos, mientras que el proceso de atacado elimina alrededor de $1 \cdot 10^6$ átomos. Esferas menores aceleran la simulación pero añaden errores significativos en la convergencia del AG debido a la reducción en la precisión de la lectura de la anisotropía de la simulación. Tras el atacado por el simulador (figura 5.2 (b)), se realiza un muestreo sobre la semiesfera, con una separación de dos grados en latitud y longitud entre puntos muestreados. Este muestreo determina esencialmente la media de la distancia al centro de la esfera de todos los átomos de dicha región discreta $D_{post(i,j)}$.

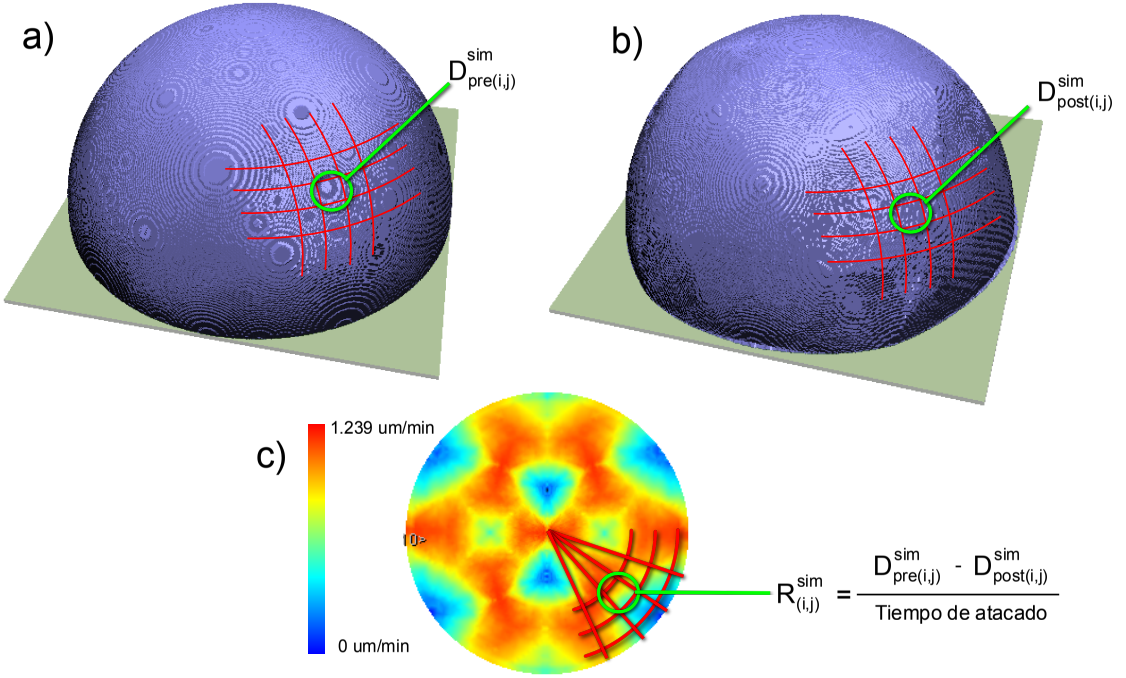

Figura 5.2: (a) Semiesfera de silicio implementada en GPUetch. (b) Atacado de la semiesfera durante 600 minutos con KOH 40wt % a 70°C. (c) Velocidades de atacado obtenidas a partir de la semiesfera muestrada.

La parte inferior de la semiesfera simulada, la cual está en contacto con con un sustrato inerte en *GPUetch* para evitar el atacado por la cara inferior, se ve afectada por esta condición de borde, por lo que las velocidades de atacado de esta zona de la esfera no son fiables. De la misma manera, los muestreos de la parte superior tienden a presentar grandes errores debido a la reducida cantidad de átomos por cada región muestreada. Debido a esto, la esfera se muestrea entre unos rangos de latitud de 10º y 70º. Debido a que todas las orientaciones posibles del silicio aparecen repetidas múltiples veces a lo largo de la esfera, el dejar fuera dichas regiones no elimina información necesaria.

El resultado del muestreo de la esfera es una matriz 2D denominada D^{sim} de dimensiones $(30, 180)$ (figura 5.2 (c)). Asimismo, un muestreo similar puede hacerse sobre los datos experimentales, obteniendo así la matriz D^{exp}.

A partir del tiempo de atacado (t_{etch}) y la profundidad atacada para cada región muestreada de la esfera antes y despues del atacado ($D^{sim}_{pre(i,j)} - D^{sim}_{post(i,j)}$), es posible obtener la velocidad de atacado para cada orientación de silicio $R^{sim}_{i,j}$ como se muestra en la figura 5.2 (c). $R^{sim}_{i,j}$ se compara directamente con la velocidad experimental ($R^{exp}_{i,j} = \frac{D^{exp}_{pre(i,j)} - D^{exp}_{post(i,j)}}{t_{etch}}$). Asimismo, con el objetivo de reducir ruido en la simulación causado por la resolución de la esfera, se aprovecha la simetría existente en las estructura cristalina para realizar una media entre 4 posiciones las cuales corresponden a la misma orientación de la superficie de silicio.

La ecuación 5.7 define matemáticamente la metodología utilizada para el cálculo de las diferencias de morfología en el atacado de la esfera:

$$E_1^k = \sum_{i=0}^{29} \sum_{j=0}^{44} \frac{\left| R^{exp}_{i,j} - R'^{sim}_{i,j} \right|}{R^{exp}_{i,j}}, \qquad (5.7)$$

$$\text{donde} \quad R'^{sim}_{i,j} = \frac{R^{sim}_{i,j} + R^{sim}_{i,(j+45)} + R^{sim}_{i,(j+90)} + R^{sim}_{i,(j+135)}}{4}.$$

i y j representa el valor correspondiente de latitud y longitud mues-

treados respectívamente para las variables R^{sim} y R^{exp}.

5.3.2.2. Deformaciones de la Estructura

Pese a que las velocidades de atacado de la esfera para un cierto individuo concuerden con los valores experimentales, existen ciertas deformaciones en las estructuras que no aparecen en el atacado a una esfera. Las deformaciones observadas que se intentan reducir mediante este coeficiente son las siguientes:

1. Diferencias de simetría en el atacado de bordes, resultando en estructuras no simétricas tras el proceso de *underetching* en una esquina (figura 5.3 (c)).

2. Efectos de *charcos* debido a una mala asignación de parámetros, lo cual hace que los átomos cercanos a una pared vertical sean eliminados más rápido de lo debido (figura 5.3 (d)).

3. Deficiencias en la forma de los bordes (falta de redondez y simetría) para atacantes basados en TMAH + Triton. Los atacantes basados en TMAH +Triton se caracterízan por poseer un efecto de *underetching* reducido y la formación de bordes redondeados para las esquinas convexas de las máscaras (figura 5.3 (e)). Esto es debido a la reducción de la velocidad de atacado de las orientaciones (110) y vecinas, haciéndola similar a la velocidad de atacado del plano (111) [111].

   Este defecto es debido a pequeñas variaciones en las velocidades de extracción en los planos intermedios entre (110) y (111), dando lugar a bordes irregulares o con salientes (figura 5.3 (f)).

Para evaluar estos efectos sobre la solución incluida dentro de un individuo determinado, se realiza el atacado de una máscara circular de 150 um de diámetro sobre una superficie de 200umx200um de tipo (100). La profundidad de atacado es de 25um (figura 5.3 (a)). La cantidad de átomos superficiales inicial es de 94.450, mientras que la cantidad de átomos eliminados en este proceso ronda los $3 \cdot 10^6$ átomos.

Tras el atacado se realiza un muestreo de la superficie, obteniendo como resultado una matriz H de dos dimensiones de tamaño $(200, 200)$ la cual almacena en cada posición la profundidad de atacado de dicho muestreo.

La ecuación aplicada para evaluar estos efectos varía en función del tipo de atacante. Para atacantes basados en KOH y TMAH, se aplica la ecuación 5.8.

$$E_2^k = E_{2(KOH)}^{\prime k} \cdot E_2^{\prime * k} \cdot E_2^{\prime ** k} \tag{5.8}$$

$$E_{2(KOH)}^{\prime k} = \left(\sum_{i=0}^{199} \sum_{j=0}^{199} |H_{i,j} - H_{j,i}| \right) \tag{5.9}$$

$$E_2^{\prime * k} = c \left| \frac{H_{(0,0)}}{H_{max}} - 1 \right| + 1 \tag{5.10}$$

$$E_2^{\prime ** k} = c \left| \frac{R_{(0,2,4,4)} a\, t}{H_{max}} - 1 \right| + 1 \tag{5.11}$$

La variable $E_{2(KOH)}^{\prime k}$ depende directamente del error de simetría, el cual se calcula obteniendo la diferencia entre H y su traspuesta. $E_2^{\prime * k}$ es un factor que detecta el efecto de charco. Para ello se obtiene la diferencia entre la profundidad máxima de atacado (H_{max}) y un punto en el borde de la superficie y alejado de las paredes verticales, donde el efecto de charco es mínimo ($H_{(0,0)}$).

Por último $E_2^{\prime ** k}$ se asegura que la velocidad de atacado vertical es equivalente a la velocidad teórica del plano (100). Aquí $R_{(0,2,4,4)}$ es la velocidad de extracción del átomo tipo *dihydride* (con dos enlaces de hidrógeno) , a equivale a una distancia de 1.35 $\mathring{A}$ y t es el tiempo de atacado aplicado a la superficie. Este tercer parámetro es necesario en ocasiones en las cuales el efecto de charco es corregido de forma inadecuada por el AG acelerando la velocidad de $R_{(0,2,4,4)}$ o acelerando la velocidad de expansión del mismo.

El coeficiente c es utilizado para potenciar el efecto de los errores calculados por $E_2^{\prime * k}$ y $E_2^{\prime ** k}$ con respecto a $E_{2(KOH)}^{\prime k}$ en E_2^k. El valor utilizado en nuestra implementación es de 4.

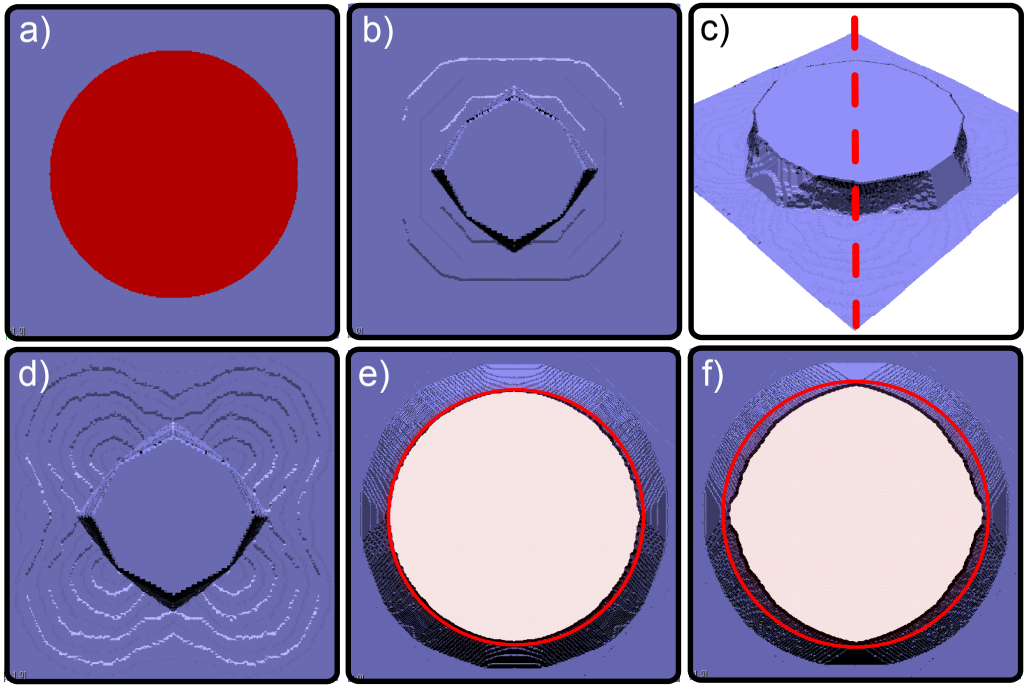

Figura 5.3: Deformaciones típicas medidas por el segundo coeficiente de la función objetivo. (a) Máscara aplicada. (b) KOH 40wt % 70°C, sin defectos. (c) KOH 40wt % 70°C, errores de simetría. (d) KOH 40wt % 70°C, errores de charco. (e) TMAH + Triton 25wt % 80°C, sin defectos. (f) TMAH + Triton 25wt % 80°C, deficiencias en la redondez de la estructura.

Cuando el atacante está basado en TMAH + Triton, el término $E'^{k}_{2(KOH)}$ de la ecuación 5.8 es modificado por otra función la cual mide al detalle errores en la forma redondeada resultante, característica de este tipo de atacantes. En este caso los errores se detectan a partir de la medida de variaciones en la profundidad de atacado para puntos que están a la misma distancia del centro. Defectos en la forma ideal se traducen en diferencias de esta profundidad. El algoritmo implementado se aplica, para un rango de distancias d_i del centro de la superficie que varía entre 0 y 95 muestras en pasos de 0.25 muestras. Los pasos menores de una muestra se consiguen interpolando los puntos de la matriz matriz H, obteniendo así H_{in}. La toma de medidas para cada distancia se realiza en 64 puntos equidistantes los cuales se obtienen incrementando un ángulo α_j. Para este nuevo término las medidas para la distancia d_i y el ángulo α_j son denominadas $H_{in(i,j)}$. La figura 5.4 muestra varios puntos de medidas para una misma distancia d_i.

Las variaciones de profundidad para cada distancia son medidas calculando la desviación estándar con el fin de evitar formas ovaladas ($\mathcal{E}_i^{(1)}$). Asimismo se pondera variación media de la profundidad en función del incremento de ángulo ($\mathcal{E}_i^{(2)}$), con el fin de reducir grandes variaciones en pequeños incrementos de ángulos, que pueden resultar en un borde ruidoso. Las ecuaciones 5.12-5.14 definen las medidas expuestas en los dos últimos párrafos.

$$E'^{k}_{2(TMAH+Triton)} = \sum_{i=1}^{N} \left(\mathcal{E}_i^{(1)} + \mathcal{E}_i^{(2)} \right) \tag{5.12}$$

$$\mathcal{E}_i^{(1)} = \sqrt{\frac{\sum_{j=1}^{M} \left(H_{in(i,j)} - \bar{H}_{in(i)} \right)^2}{M}} \tag{5.13}$$

$$\mathcal{E}_i^{(2)} = \sum_{j=2}^{M} \left| H_{in(i,j)} - H_{in(i,j-1)} \right| \tag{5.14}$$

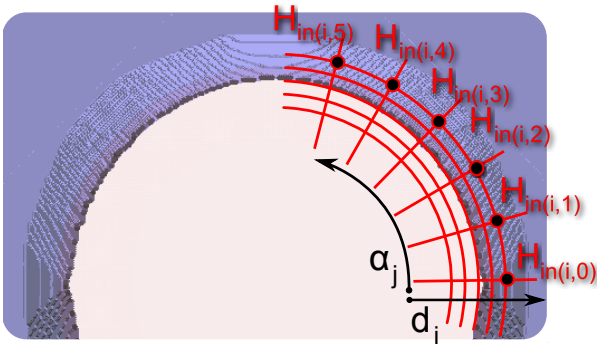

Figura 5.4: Medida de la profundidad de atacado $H_{in(i,j)}$ en función de la distancia d y el ángulo α para atacantes basados en TMAH + Triton.

En este caso, i indexa cada una de las distancias al centro utilizada para el cálculo de un total de N, j indexa cada uno de los puntos utilizados a distancia i del centro (de un total de M, 64 en nuestra aplicación), $\mathcal{E}_i^{(1)}$ mide la variación de la profundidad de atacado a puntos de la misma distancia y $\mathcal{E}_i^{(2)}$ puede entenderse como la derivada de la profundidad en función del incremento de ángulo.

5.3.2.3. Velocidades de Extracción entre Configuraciones con Primeras Vecindades Distintas

El objetivo de esta variable es preservar el correcto funcionamiento del *step-flow* favoreciendo que configuraciones atomísticas con mayor cobertura de terminaciones OH posean velocidades de extracción mayores. La ecuación aplicada es la siguiente:

$$E_3^k = \sum_{n1s=1}^{3} \sum_{n1b=0}^{3} \left(\frac{\bar{R}_{n1s,n1b}}{\bar{R}_{(n1s-1),n1b}} \right) + \tag{5.15}$$

$$+ \sum_{n1s=0}^{3} \sum_{n1b=1}^{3} \left(\frac{\bar{R}_{n1s,n1b}}{\bar{R}_{(n1s-1),n1b}} \right) + \tag{5.16}$$

$$+ \sum_{n1s=1}^{3} \sum_{n1b=1}^{3} \left(\frac{\bar{R}_{n1s,(n1b-1)}}{\bar{R}_{(n1s-1),n1b}} \right); \tag{5.17}$$

Las variables $\bar{R}_{n1s,n1b}$ representan la velocidad media de extracción para los átomos que poseen una primera vecindad de n^{1s} y n^{1b} átomos superficiales y enterrados respectivamente. El minimizar este coeficiente fuerza que las velocidades de extracción para familias con mayor cobertura de OH sean mayores que familias con una menor cobertura. Así mismo, se optimiza el hecho de que familias con misma cantidad de vecinos pero de carácter *bulk* en vez de superficial posean menor velocidad de atacado, como se ha observado en diversos casos de calibraciones del modelo [128].

5.3.2.4. Variación en la Velocidad de Extracción entre Configuraciones con Primeras Vecindades Similares

Asimismo, para poder respetar la correcta implementación del *step-flow*, la siguiente variable está pensada para favorecer que familias de átomos con configuración similar posean velocidades de extracción similar. La forma de representar esto es midiendo la desviación estándar de las configuraciones cuya primera vecindad es idéntica:

$$E_4^k = \sum_{n1s=0}^{3} \sum_{n1b=0}^{3} \sqrt{\frac{\sum_{n2s=0}^{11} \sum_{n2b=0}^{11} R_{(n1s,n1b,n2s,n2b)} - \bar{R}_{(n1s,n1b)}}{12 \cdot 12}} \tag{5.18}$$

5.3.3. Selección de Individuos

En el proceso de selección, se escoge para cada iteración los individuos que serán utilizados como padres para la generación de los nuevos individuos. Cada individuo en este proceso recibe una probabilidad de reproducción dependiendo del valor obtenido mediante la función objetivo y dicho valor de los otros individuos de la población.

El método de selección escogido está basado en la generación de un ranking de individuos en función de la puntuación obtenida por el valor objetivo. Dicho ranking introduce un escalado uniforme a lo largo de la población y permite una forma simple y efectiva de controlar la *presión selectiva*, la cual determina la probabilidad de que los individuos con mejores características sean escogidos. La clasificación basada en ranking es un método más robusto que definir las probabilidades de selección de forma proporcional al resultado de la función objetivo [168].

En nuestra implementación, hemos escogido la implementación de un ranking lineal, donde la probabilidad de selección en función de la posición *pos* en el ranking es:

$$Prob(pos) = 2 - SP + 2 \cdot (SP - 1) \cdot \frac{(Pos - 1)}{(Nind - 1)} \tag{5.19}$$

Donde SP puede tomar valores en el rango $[1.0, 2.0]$ y se utiliza para variar la presión selectiva. En nuestro algoritmo, hemos escogido $SP = 1.5$, el cual provee una buena relación entre presión selectiva y pérdida de diversidad en la población.

Una vez creado el ranking, se escogen individuos por parejas de forma aleatoria, donde las posibilidades de selección del individuo i vienen dadas por $Prob(i)$, nunca escogiendo el mismo individuo dos veces para la misma pareja. El método para escoger los individuos implementado es denominado *rueda de ruleta* [169].

5.3.4. Recombinación

La recombinación produce nuevos individuos a partir de la combinación de los genes de los padres. En nuestro caso la recombinación es aplicada a valores reales: las velocidades de extracción para las posibles configuraciones atomísticas. La metodología aplicada en nuestro algoritmo es la *recombinación intermedia* [170], utilizada únicamente con variables reales. En este método, cada gen del hijo es generado a partir de la combinación de los respectivos genes del padre ponderados por un factor aleatorio. La regla utilizada para la recombinación es la siguiente:

$$Var_i^{hijo} = Var_i^{padre1} \cdot a_i + Var_i^{padre2} \cdot (1 - a_i) \tag{5.20}$$

Donde Var_i^{hijo} representa el gen i del hijo producido. a es un valor aleatorio entre el rango $[-d, 1 + d]$, donde d representa el área de expansión del valor del gen del hijo. Un valor de $d = 0.25$ es recomendado para evitar la compresión del rango de datos a lo largo de las generaciones [171].

En nuestro algoritmo, la aplicación de esta estrategia de recombinación da lugar a 24 hijos a partir de los padres seleccionados mediante la técnica explicada en 5.3.3.

5.3.5. Mutación

En el proceso de mutación, los individuos creados a partir de la recombinación son alterados de forma aleatoria con el fin de encontrar variaciones que den mejores resultados en la función objetivo. Estas variaciones tienden a ser pequeñas y son aplicadas a los genes de los individuos con una pequeña probabilidad. El proceso de mutación es utilizado para poder salir de óptimos locales y prevenir que los genes entre los distintos individuos sean demasiado similares.

En nuestro algoritmo, el porcentaje de variables mutadas que produce resultados óptimos es del 5 %. Cuando se aplica una mutación, para escoger el tamaño de dicha mutación aplicamos el operador de mutación del *Breeder Genetic Algorithm* (ecuación 5.21), recomendado por [170], el cual

aplica pequeñas mutaciones con una gran probabilidad y grandes mutaciones con una pequeña probabilidad. Pequeños cambios en los genes suelen dar buenos resultados. Sin embargo mutaciones grandes pueden dar lugar a una convergencia más rápida a la soluciόm óptima.

$$Var_i^{mut} = Var_i \cdot s_i \cdot r_i \cdot a_i \tag{5.21}$$

En la ecuación 5.21 s_i es un valor aleatorio uniforme que puede tomar los valores $\{-1, +1\}$, r_i define el rango de mutación, asignado en nuestro algoritmo al valor actual del gen que está siendo modificado y $a_i = 2^{-u \cdot k}$ es un valor de escalado que potencia la probabilidad de mutaciones pequeñas sobre las grandes. Dentro de la definición de a_i, u es una variable aleatoria dentro del rango $[0, 1]$ y k define la precisión de la mutación, es decir, define el mínimo paso de mutación posible. En nuestra implementación, $k = 8$ ofrece buenos resultados de convergencia.

5.3.6. Reinserción

Una vez los hijos de la nueva generación han sido producidos a partir de los métodos de selección, recombinación y mutación, la función objetivo debe ser evaluada sobre ellos. El método recomendado de reinserción es la reinserción elitista basada en una clasificación previa [171]. En este método, en cada generación se crea una cantidad de hijos menor a la población. Tanto los padres como los hijos son clasificados de mejor a peor mediante la función objetivo. Para una cantidad de individuos concreta, los peores individuos de la población son substituidos por los mejores hijos.

En nuestro AG implementado, de los 24 hijos generados, los 10 cuya función objetivo da mejores resultados substituyen a los 10 peores individuos de la población.

5.4. Resultados

El AG definido en la sección anterior ha sido implementado y aplicado para la calibración de un total de 34 atacantes los cuales pueden dividirse

en los siguientes grupos:

- Atacantes basados en KOH con concentraciones 24wt %, 30wt %, 35wt %, 40wt % y 50wt % a 60°C, 70°C y 80°C para todos los casos.
- Atacantes basados en TMAH 10wt % 20wt % y 25wt % a 60°C, 70°C y 80°C para todos los casos.
- Atacantes basados en KOH + IPA 24wt % a 60°C, 70°C y 80°C.
- Atacantes basados en TMAH + Triton 25wt % a 60°C, 70°C y 80°C.
- Atacantes isótropos.

El AG ha sido capaz de calibrar correctamente todos ellos. Dichas calibraciones pertenecen actualmente a la base de datos del simulador comercial IntelliEtch [151]. Sin embargo, el tiempo de convergencia no es similar, siendo los atacantes basados en TMAH + Triton los más costosos. Esto es debido a que pequeñas variaciones en las velocidades de extracción entre los planos (100) y (111) dan lugar a deformaciones significativas en las estructuras creadas.

El sistema utilizado para realizar las calibraciones consiste en siete ordenadores conectados entre sí mediante una red *ethernet*. Seis de ellos estaban equipados con tarjetas Nvidia (5 Nvidia Geforce GTX260 + 1 Nvidia Tesla C1060) para la evaluación de las funciones objetivo sobre GPUetch. El séptimo ordenador se encargaba de ejecutar el AG y distribuir las simulaciones requeridas a través del resto de ordenadores. El algoritmo encargado de ejecutar el AG ha sido desarrollado en Java y en la actualidad forma parte de *GPUetch* como utilidad interna disponible para el usuario a la hora de calibrar nuevos atacantes.

Con este método, el tiempo de calibración ronda las 5 horas para los atacantes basados en KOH, los cuales requieren varios cientos de generaciones para evolucionar hasta la solución óptima y alrededor de 24-48 horas para los atacantes basados en TMAH + Triton, los cuales requieren varios miles de generaciones.

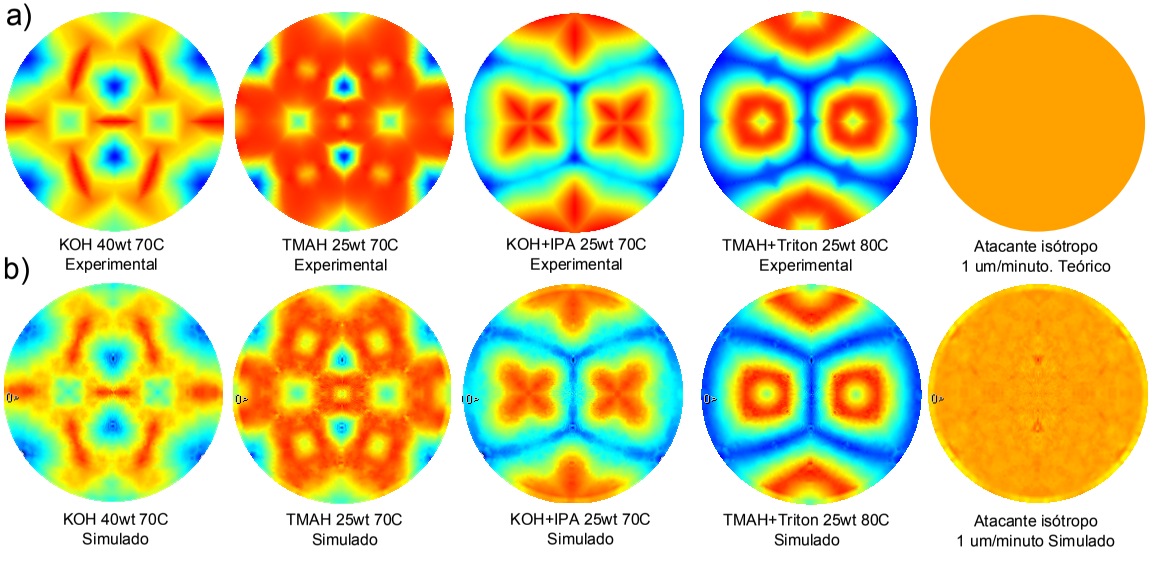

Figura 5.5: Resultados de la anisotropía de las calibraciones convergidas mediante el AG. (a) Velocidades de atacado experimentales obtenidas de [160]. (b) Velocidades de atacado obtenidas mediante *GPUetch* con las calibraciones realizadas con el AG.

A continuación mostramos diversos resultados obtenidos de la aplicación del AG, incluyendo: Una vista previa del resultado de convergencia de la anisotropía de un atacante de cada grupo, el proceso de calibración del atacante KOH 40wt % 70ºC así como microestructuras simuladas a partir de la calibración obtenida. Asimismo mostramos resultados de la convergencia de la calibración de un atacante basado en TMAH + Triton 25wt % a 80ºC.

5.4.1. Resultado de Anisotropía para las Configuraciones Convergidas

En la figura 5.5 mostramos el resultado de los atacados a la semiesfera de silicio para un atacante de cada familia. Los resultados obtenidos del simulador (apartado b) muestran que la población ha conseguido adaptar sus genes para emular la anisotropía del experimento en todos los casos.

5.4.2. Proceso de Calibración para el Atacante KOH 40wt % 70ºC

La figura 5.6 (a-g) muestra la morfología resultante de la esfera al atacarla mediante la configuración para el mejor individuo de la población en distintas etapas del proceso de calibración. Dichas imágenes evidencian que la anisotropía del atacante converge a la del experimento (figura 5.6 (h)). En 200 iteraciones la anisotropía está prácticamente convergida.

Mirando en detalle los resultados de los distintos elementos de las funciones objetivo E_i en la figura 5.7, se aprecia que tanto la anisotropía de la esfera como los defectos del atacado se han reducido hasta un mínimo. La relación entre las distintas familias según sea su primer vecindario converge aunque de forma más lenta, el cual llega a un mínimo tras las 600 iteraciones. El hecho de hacer converger el error E_3 pese a que no afecta a la anisotropía general del atacante, da lugar a unas calibraciones menos ruidosas donde el *step-flow* se aprecia con mayor claridad (figura 5.8 (a) vs (b)). Por último, la dispersión de las configuraciones similares se mantiene constante.

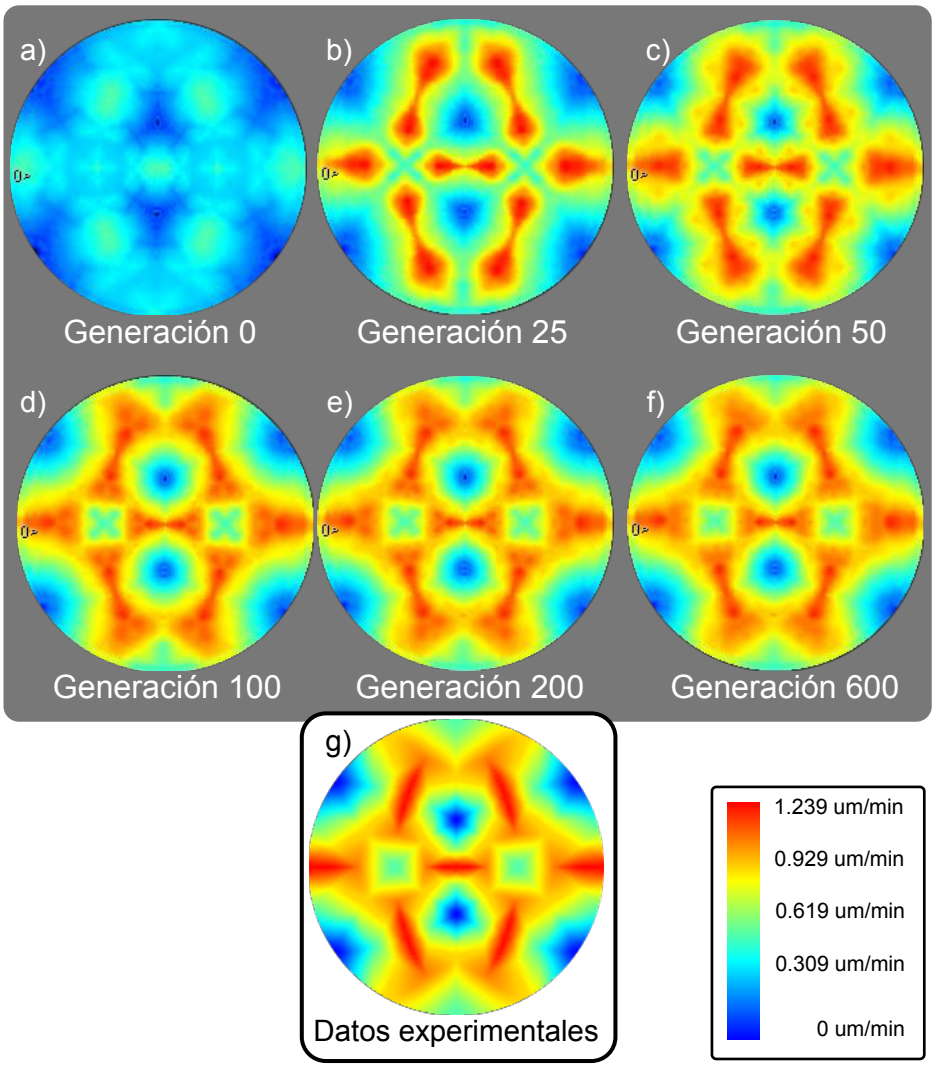

Figura 5.6: (a-g) Evolución de la anisotropía del mejor individuo del AG. (h) Datos experimentales usados para la convergencia.

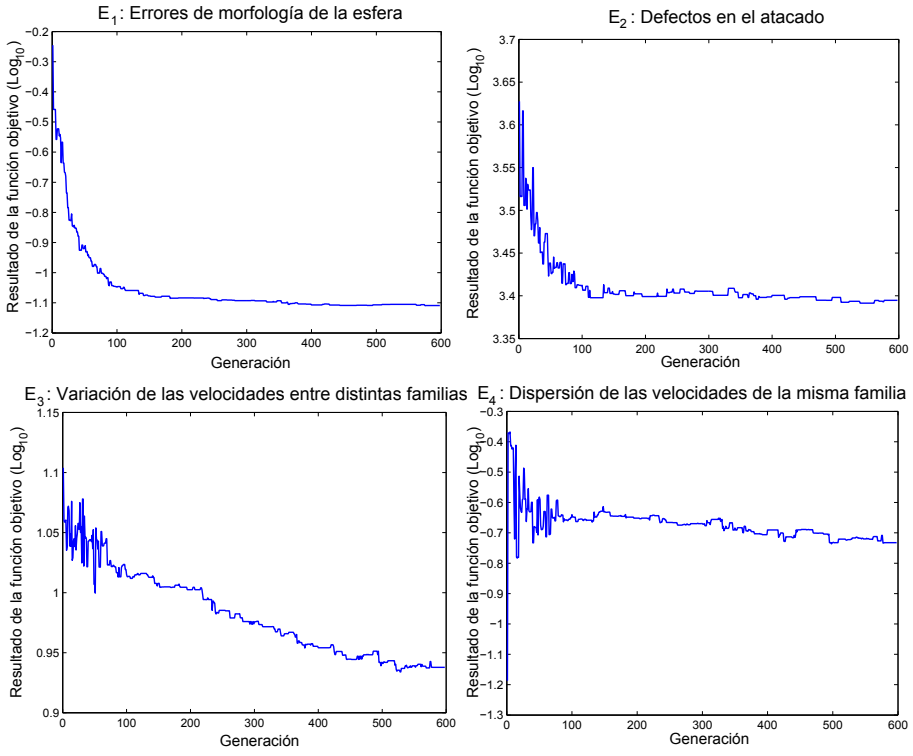

Figura 5.7: Evolución de los valores obtenidos de los distintos componentes de la función objetivo para el mejor individuo de la población en función de la generación.

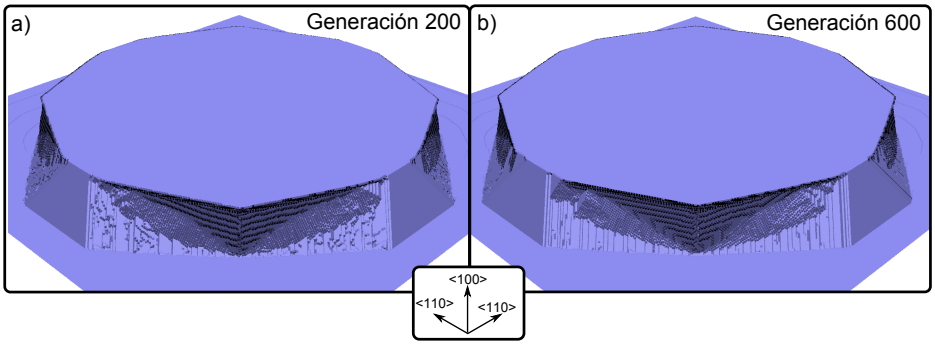

Figura 5.8: Atacado mediante el ACC de una superficie de silicio cuadrada con orientación (100) en la que se ha aplicado una máscara cuadrada con las aristas paralelas a (110). (a) Calibración de KOH 40wt % 70°C convergido con el AG tras 200 iteraciones. Calibración de KOH 40wt % 70°C convergido con el AG tras 600 iteraciones.

La gráfica 5.9 muestra las velocidades de reducción de ocupación asignadas a las distintas configuraciones para el mejor individuo de la generación 600. Dichas configuraciones aparecen agrupadas por elementos con primeras vecindades (n^{1s}, n^{1b}) idénticas. Las distintas líneas trazadas entre los grupos muestran las relaciones entre grupos que se definen en la ecuación de E_3. Una mayor convergencia en E_3 resulta en mayores diferencias en las velocidades entre las distintas agrupaciones. En la gráfica se aprecia que las distintas familias poseen velocidades de atacado muy diferenciados.

Por último, hemos utilizado la calibración obtenida mediante el AG definido para simular el atacado de distintas microestructuras:

- Microsonda de dos ejes (figura 5.10 (a)) [95]
- Accelerómetro de tres ejes (figura 5.10 (b)) [93].
- Matriz de microagujas (figura 5.10 (c)) [172].
- Punta para un microscopio de fuerza atómica (figura 5.10 (d)) [173].

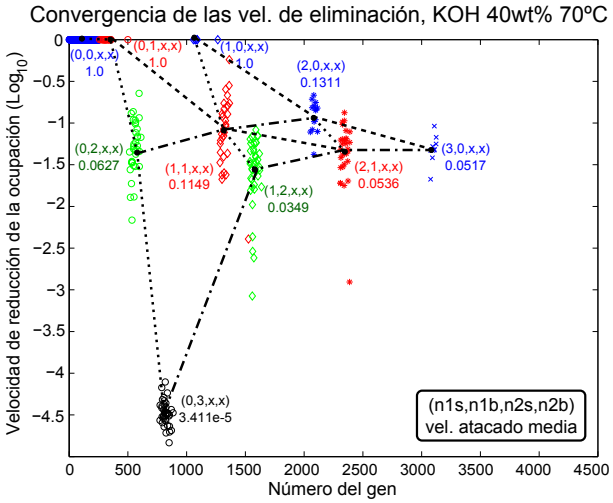

Figura 5.9: Valores de las velocidades de reducción de la ocupación del mejor individuo tras 600 generaciones.

Los resultados obtenidos se asemejan en gran medida a las microestructuras obtenidas de forma experimental, demostrando de esta manera el buen comportamiento de la calibración obtenida.

5.4.3. Resultados para TMAH + Triton 25wt % 80ºC

La buena calibración del ACC para atacantes basados en TMAH + Triton, irrealizable hasta ahora, es uno de los objetivos principales de la búsqueda de nuevos métodos de calibración. El AG necesita varios miles de generaciones (habitualmente menos de 2000) para la correcta calibración de este tipo de atacantes. La figura 5.5 (b) muestra la anisotropía obtenida de la calibración mediante el AG. Cualitativamente el resultado es prácticamente idéntico, lo que significa que *a priori* el ACC es capaz de simular adecuadamente este tipo de atacantes.

Para comprobar la validez de dicha calibración hemos simulado diversos procesos de micromecanizado basados en este atacante y lo hemos

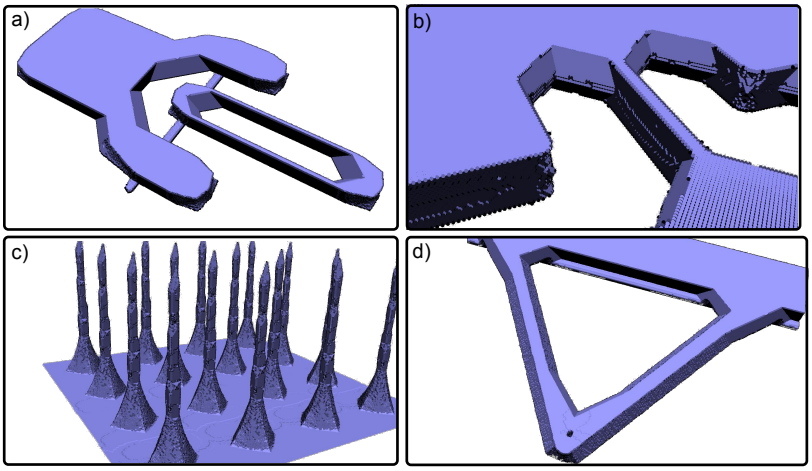

Figura 5.10: Microestructuras creadas a partir del atacante KOH 40wt % 70ºC convergido mediante el AG. (a) Microsonda de dos ejes [95]. (b) Plano ampliado de acelerómetro de tres ejes [93]. (c) Matriz de microagujas [172]. (d) Punta de microscópio [173].

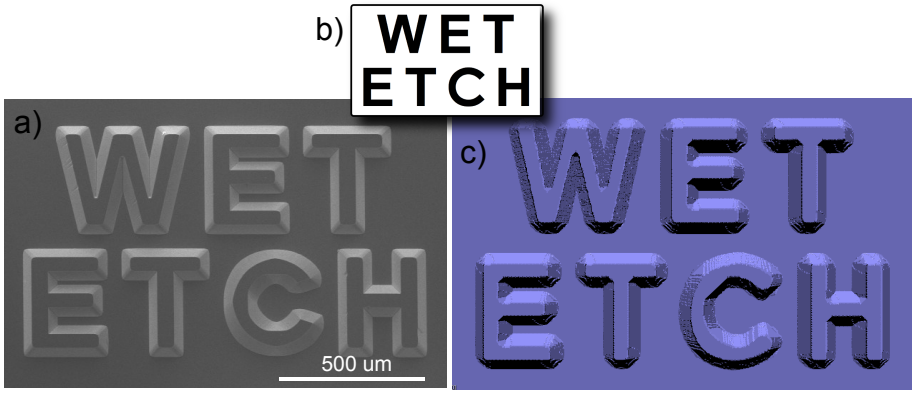

Figura 5.11: Grabado de letras en silicio mediante atacado húmedo con TMAH + Triton 25wt % a 80ºC. (a) Experimento [174]. (b) Máscara. (c) Simulación con GPUetch usando el atacante calibrado con el AG.

comparado con resultados experimentales para evaluar la adecuación de la calibración obtenida. Las microfotografías utilizadas en las figuras 5.11, 5.12 y 5.13 han sido obtenidas gracias a [174].

La figura 5.11 (a) muestra el atacado sobre una máscara la cual graba la inscripción *wet etch* sobre el silicio (figura 5.11 (b)). El propósito de dicha máscara es mostrar de forma cualitativa el reducido efecto de *underetching* que posee este tipo de atacantes, lo que permite grabar símbolos del alfabeto sobre el silicio, algo imposible con atacantes tradicionales basados en KOH o TMAH. El tamaño de la superficie de silicio es 1500umx500um, mientras que la profundidad de atacado es de 40um. La figura 5.11 (c) muestra el resultado de una simulación basada en *GPUetch* la cual emula los parámetros del experimento. El efecto del *underetching* es correctamente modelado por el atacante convergido mediante el AG, resultando en una simulación fiel al experimento.

La figura 5.12 muestra el resultado para un proceso más complejo de fabricación. La estructura fabricada es denominada matriz de cavidades cuadradas o *square ashtray array* [111]. En este caso, la superficie posee una dimensión de 600x600um. El proceso utiliza las máscaras mostradas en el apartado (b) y consiste en los siguientes pasos:

1. Aplicar la máscara (b.1) como óxido de silicio.
2. Aplicar la máscara (b.2) como nitruro de silicio.
3. Atacar con TMAH + Triton hasta 40um de profundidad.
4. Permitir la oxidación local del silicio. En *GPUetch* se consigue aplicando la máscara (b.3) como óxido de silicio.
5. Eliminar la capa de nitruro de silicio y atacar hasta 40um de profundidad otra vez.

Una vez realizados estos pasos, tanto en el experimento como en el simulador, los resultados muestran otra vez una gran similitud entre ambos, demostrando la corrección del simulador para procesos de fabricación mas complejos.

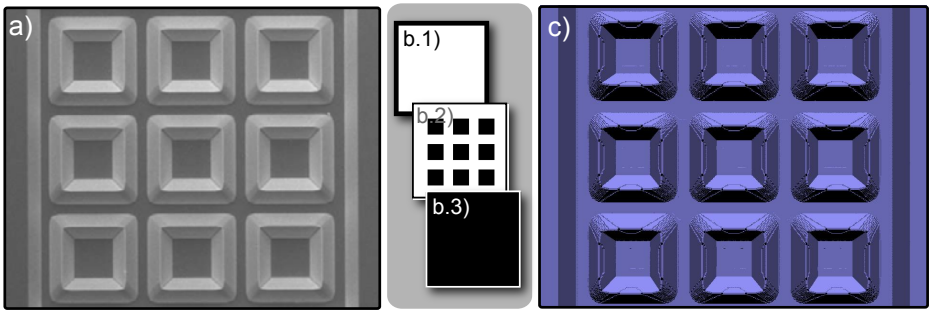

Figura 5.12: Fabricación de la estructura *square ashtray array*. (a) Resultado experimental [174]. (b) Máscaras utilizadas. (c) Simulación con GPUetch usando el atacante calibrado con el AG.

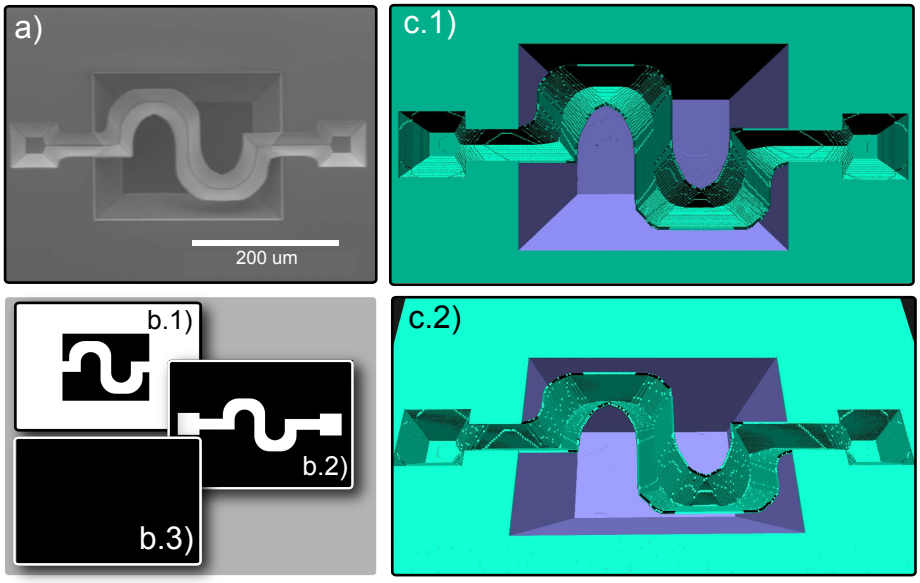

Figura 5.13: Fabricación de un microcanal suspendido en serpentín. (a) Resultado experimental [174]. (b) Máscaras utilizadas. (c) Simulación con GPUetch usando el atacante calibrado con el AG.

Por último, en la figura 5.13 mostramos el proceso de fabricación de un microcanal suspendido en serpentín [111]. Los microcanales en MEMS pueden ser utilizados en aplicaciones tales como transporte o mezcla de fluidos a escala microscópica. El proceso de fabricación de esta microestructura es similar al explicado para la figura 5.12, utilizando las máscaras mostradas en el apartado (b). Para este caso, el tamaño de la superficie de silicio es de 550umx380um, y el atacado se realiza por tiempo, atacando 60 minutos en el paso 3 y 160 minutos adicionales en el paso 5. Los resultados obtenidos en la simulación muestran de nuevo la validez de la calibración obtenida con el AG.

5.5. Conclusiones

El proceso de calibración actualmente existente del Autómata Celular Contínuo (ACC) ha sido un procedimiento que no garantizaba el correcto funcionamiento para todos los atacantes debido a la mala adecuación de la *Función de Probabilidad de Extracción* (RPF, del inglés *Removal Probability Function*) a los nuevos atacantes tales como los basados en TMAH + Triton.

En este capítulo hemos demostrado la viabilidad de utilizar de *algoritmos genéticos* (AGs) para la correcta calibración del ACC para la simulación de nuevos atacantes. El AG diseñado ha sido aplicado de forma satisfactoria a una gran cantidad de atacantes los cuales pueden agruparse en: KOH, TMAH, TMAH + Triton, KOH+IPA y atacantes isótropos.

La aplicación del AG ha obtenido buenos resultados, los cuales se han validado mediante la comparación con resultados experimentales, no sólo con caracterizaciones experimentales de los atacantes sino también con procesos reales de micromecanizado.

La implementación previa de *GPUetch* (captítulo 4) ha hecho posible la investigación y aplicación de esta metodología, algo imposible anteriormente debido al alto coste computacional que suponía el aplicar AGs sobre el ACC. Esto es debido a que diversas de las operaciones necesarias para el funcionamiento del AG implican la realización de dos simulaciones de atacado por cada nuevo individuo generado.

Pese a que el AG definido en este capítulo puede no ser el más óptimo para la resolución del problema propuesto, ha sido obtenido tras muchas iteraciones y obtiene resultados satisfactorios para todos los atacantes probados, por lo que lo presuponemos una solución adecuada y suficientemente óptima.

En conclusión, la calibración del ACC mediante AGs ha demostrado ser un método efectivo el cual supera las limitaciones actualmente existentes de la metodología tradicional basada en la RPF. Esto ha permitido extender la utilidad del ACC para nuevos atacantes, así como mejorar la precisión en la simulación de atacantes tradicionales.

Capítulo 6

Reformulación del Autómata Celular Continuo: Aceleración de la Implementación Exacta.

Pese al éxito del ACC para simular el grabado anisótropo húmedo, el coste computacional de las implementaciones exactas existentes del ACC se ha mantenido prohibitivamente alto para aplicaciones de ingeniería. Este hecho ha forzado el uso de implementaciones aproximadas para realizar simulaciones con tiempos computacionales razonables [154]. El presente capítulo intenta corregir esta situación presentando una reformulación del ACC completamente equivalente del modelo exacto que reduce de forma significativa el coste computacional, permitiendo realizar simulaciones con el modelo ACC sin reducción de precisión en tiempos razonables.

Esto es conseguido mediante la introducción de un nuevo concepto teórico: el *Tiempo Anticipado de Extracción* (PRT, del ingles *Predicted Removal Time*) el cual, junto a una estructura de datos denominada *árbol binario de búsqueda autoequilibrado* (SB-BST, del inglés *Self-Balanced Binary Search Tree*), permite una forma eficiente de determinar cual es el

siguiente átomo o grupo de átomos a ser extraído en cada paso de tiempo.

En el presente capítulo se realiza un análisis del coste computacional de las implementaciones actualmente existentes el ACC, se introduce la nueva formulación, realizando una comparación de coste y precisión, y por último se muestra un caso novedoso de simulación, donde se aprecia cualitativamente las diferencias en los resultados debido a usar distintas implementaciones.

6.1. Implementaciónes Exacta y Aproximada del ACC

Una de las características más importantes de cualquier implementación del ACC es cómo evoluciona el tiempo a lo largo de la simulación. Igualando a cero la parte izquierda de la ecuación 2.6, se puede obtener la magnitud del incremento de tiempo necesario para extraer por completo el átomo i:

$$\Delta t_i(k) = \Pi_i(k)/r_i(k). \tag{6.1}$$

Para una implementación exacta del modelo, el tiempo de simulación debe avanzar de acuerdo al incremento mínimo encontrado a partir de la evaluación de la ecuación 6.1 para todos los átomos superficiales. El átomo o grupo de átomos que comparten el incremento de tiempo mínimo será extraído y las velocidades de extracción serán recalculadas para los átomos cercanos cuya vecindad ha sido modificada. Debido a que la magnitud del incremento mínimo de tiempo variará a lo largo de la simulación, la implementación del ACC que sigue esta filosofía es llamada *ACC de Paso de Tiempo Variable* (VTS-CCA, del ingles *Variable Time Stepping Continuous Cellular Automata*). Como se explicará en la sección 6.2.2, en la práctica el coste computacional del método VTS-CCA es demasiado alto para realizar simulaciones reales para aplicaciones de ingeniería, por lo que sería deseable poseer un método de simulación alternativo más rápido.

En la actualidad, la solución aceptada ha sido mantener el paso de tiempo constante y escoger un paso de tiempo pequeño para evitar la eli-

minación de demasiados átomos cuyo incremento de tiempo es mayor al mínimo. Este método es denominado *ACC de Paso de Tiempo Costante* (CTS-CCA, del ingles *Constant Time Step Continuous Cellular Automata*). A pesar de que esta implementación reduce el coste computacional de la simulación, también introduce errores debido a dos efectos [154]:

1. La ocupación de los átomos superficiales cuyo incremento de tiempo para ser extraído es menor que el paso de tiempo es menor al paso de tiempo constante se vuelve negativa antes de que dichos átomos sean extraídos, por lo tanto, la parte del incremento de tiempo asociado a la ocupación negativa se pierde, pues debería resultar en una reducción de ocupación de los átomos vecinos.

2. El método permite la extracción simultánea de diferentes grupos de átomos que debería ocurrir de forma secuencial de acuerdo a la implementación exacta.

A pesar de la existencia de métodos de compensación de tiempos los cuales se han introducido con el objetivo de reducir estos errores, ninguna de las modificaciones resuelve completamente las deficiencias intrínsecas del método CTS-CCA [154].

Estos errores introducen diferencias en las velocidades de atacado de los planos de silicio. Como ejemplo, la figura 6.1 muestra una comparación de las velocidades de atacado simuladas y experimentales para un amplio rango de planos cristalográfios desde (100) a (111) y desde (110) a (111) para cuatro atacantes distintos: KOH y KOH+IPA a dos concentraciones diferentes (11wt % y 37wt %) a 70°C [106]. Para las cuatro instancias, los resultados muestran que las velocidades de extracción del VTS-CCA se mantienen más cerca del experimento que las de CTS-CCA. Los errores son más acusados en planos de eliminación rápida y los planos de las famílias $(h11)$ y $(hh1)$. Debido a que los errores son dependientes de la orientación, esto lleva a una modificación en la anisotropía del atacante simulado. A pesar de que la implementación CTS-CCA ha sido utilizada tradicionalmente como una alternativa válida al VTS-CCA, la figura 6.1 indica de forma inequívoca que la utilización del método exacto VTS-CCA mejoraría de forma substancial los resultados globales obtenidos en un amplio

rango de simulaciones de atacado en caso de que fuera posible hacer dicha implementación lo suficientemente rápida. Con el objetivo de obtener la figura 6.1, el ACC ha sido calibrado de acuerdo a los datos experimentales (linea contínua), usando el método presentado en [127].

6.2. Coste Computacional de la Simulación del Modelo ACC

Tradicionalmente, los simuladores basados en ACCs mantienen en memoria una lista de átomos superficiales [149]. Dichos átomos superficiales reaccionan con el atacante y su ocupación se reduce gradualmente. Una implementación directa de este comportamiento consiste en que la ocupación de todos los átomos de dicha lista debe reducirse en cada paso de tiempo. Asimismo, cuando la ocupación de un átomo superficial es totalmente reducida a cero, el átomo en cuestión debe ser eliminado de la lista, el estado de los átomos vecinos debe ser actualizado y, si nuevos átomos aparecen en la superficie, deben ser añadidos a la lista.

En este estudio utilizamos T para referirnos al coste computacional, es decir, la cantidad teórica de instrucciones necesarias para realizar cierta tarea. Reservamos el uso de t para definir la variable *tiempo* en la simulación. El coste computacional (T) en los métodos ACC posee dos contribuciones:

1. El coste de reducir la ocupación de todos los átomos superficiales, (T_{oc}).
2. El coste relacionado con tareas de mantenimiento (*bookkeeping*) que actualizan la vecindad de cada átomo extraído, (T_{bk}).

La ecuación 6.2 enuncia de forma matemática dicho coste:

$$T = \sum_{k=0}^{K-1} (T_{oc} \cdot N^{(k)} + T_{bk} \cdot N^{(k)}_{removed}) \,, \tag{6.2}$$

$$T \approx T_{oc} \cdot K \cdot N + T_{bk} \cdot K \cdot N_{removed} \,, \tag{6.3}$$

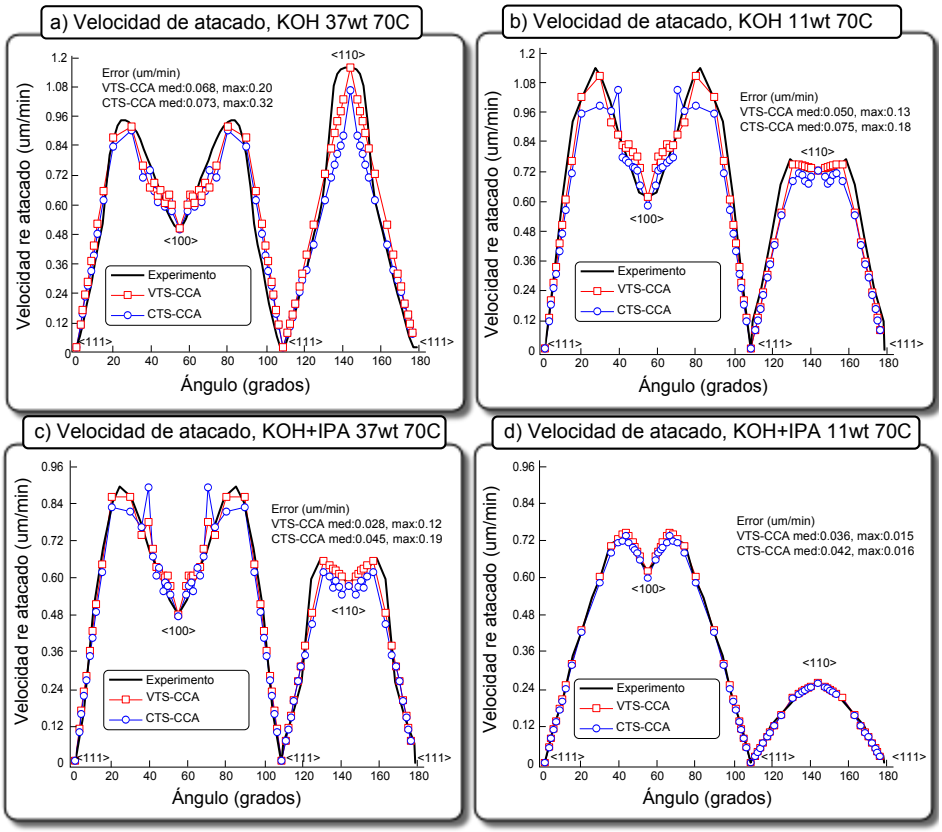

Figura 6.1: Velocidades de atacado en función de la orientación para experimentos [106] y simulaciones CTS-CCA y VTS-CCA para KOH a 11wt % y 37 wt % y KOH+IPA a11wt % y 37 wt %. Todos los experimentos están realizados a 70°C.

donde K es el número total de pasos de tiempo de la simulación, $N^{(k)}$ es el numero total de átomos superficiales en el paso de tiempo k y $N^{(k)}_{removed}$ es el número de átomos extraídos por paso de tiempo k. Con el objetivo de simplificar los cálculos posteriores, podemos suponer que las cantidades de átomos superficiales y extracciones es constante a lo largo de la simulación. La ecuación 6.3 realiza esta simplificación, donde $N = \sum_{k=0}^{K-1} N^{(k)}/K$ es la media de átomos superficiales a lo largo del tiempo y $N_{removed} = \sum_{k=0}^{K-1} N^{(k)}_{removed}/K$ es la media de átomos extraídos por paso de tiempo. Por último, las tareas de *bookkepping* dependen del tamaño medio de la vecindad del átomo. Por tanto, su coste $T_b k$ puede ser estimado como un valor constante de un orden de magnitud más grande que el coste de reducir la ocupación por átomo (T_{oc}).

Definamos

$$T^{(RA)} = T/N^{total}_{removed} \tag{6.4}$$

como el coste computacional medio por átomo extraído, donde

$$N^{total}_{removed} = K \cdot N_{removed} \tag{6.5}$$

es el número total de átomos extraídos. Además, definamos

$$g = (N^{total}_{removed})^{1/3}/N^{1/2} \tag{6.6}$$

como la relación de aspecto entre una profundidad característica asociada al volumen eliminado de átomos y un tamaño de superficie característico asociado a la cantidad de átomos superficiales.

Esta definición está basada en el hecho de que dos sistemas escalados $N_2 = e \cdot N_1$ alcanzarán el mismo estado propagado cuando se cumpla

$$N^{total}_{removed,2} = e^{3/2} \cdot N^{total}_{removed,1} = (N_2/N_1)^{3/2} \cdot N^{total}_{removed,1}. \tag{6.7}$$

De esta igualdad se obtiene que

$$g_2 = (N^{total}_{removed,2})^{1/3}/N_2^{1/2} = (N^{total}_{removed,1})^{1/3}/N_1^{1/2} = g_1 = constante. \quad (6.8)$$

En otras palabras, g es una constante que depende de la relación de aspecto entre volumen eliminado y tamaño de la superficie, cuyo valor es similar para estructuras de forma similar (escaladas entre sí). La constante g puede ser utilizada para expresar otras variables con el objetivo de comparar el coste computacional de sistemas equivalentes escalados.

Aplicando la igualdad $K \cdot N_{removed} = N^{total}_{removed} = g^3 N^{3/2}$, la ecuación 6.3 da:

$$T \approx \left(T_{oc} \frac{N}{N_{removed}} + T_{bk} \right) \cdot g^3 \cdot N^{3/2} , \quad (6.9)$$

$$T^{(RA)} \approx T_{oc} \frac{N}{N_{removed}} + T_{bk} . \quad (6.10)$$

De acuerdo con la ecuación 6.10, si el número de átomos por paso de tiempo $N_{removed}$ es similar al número de átomos visitados N, el término de *bookkeeping* $T_{bk} \approx 10 T_{oc}$ dominará el coste por átomo extraído. Sin embargo, si $N_{removed}$ es significativamente menor que N, el costé se verá dominado por las tareas de reducción de ocupación $T_{oc} \cdot N/N_{removed}$.

Como se analiza más adelante en la sección 6.4, queremos remarcar el coste computacional mostrado en las ecuaciones 6.9–6.10 para el método ACC es directamente proporcional a la *ineficiencia* de la simulación, la cual definimos como $q = N/N_{removed}$, es decir la cantidad de visitas a átomos necesarias para poder eliminar un único átomo:

$$T \approx (T_{oc} q + T_{bk}) \cdot g^3 \cdot N^{3/2} , \quad (6.11)$$

$$T^{(RA)} \approx T_{oc} q + T_{bk} . \quad (6.12)$$

6.2.1. ACC de Paso de Tiempo Constante

Definamos la velocidad de extracción total del sistema $R^{(k)}$ como la suma de todas las velocidades de extracción de los átomos superficiales:

$$R^{(k)} = \sum_{0}^{N^{(k)}-1} r_i^{(k)}, \tag{6.13}$$

y la velocidad media por átomo r como:

$$r^{(k)} = R^{(k)}/N^{(k)}. \tag{6.14}$$

Habitualmente, R varía de un paso de tiempo a otro debido a que varios átomos superficiales son añadidos/eliminados durante el paso de tiempo anterior.

En el método CTS-CCA, el número de átomos extraídos $N_{removed}^{(k)}$, el cual aparece en las ecuaciones 6.9 – 6.10, puede ser expresado como la reducción total en la ocupación de todos los átomos superficiales:

$$N_{removed}^{(k)} = \sum_{i=0}^{N^{(k)}-1} \left(\Pi_i^{(k)} - \Pi_i^{(k+1)}\right) , \tag{6.15}$$

Aplicando la ecuación 2.6 sobre 6.15 obtenemos 6.16.

$$N_{removed}^{(k)} = \sum_{i=0}^{N^{(k)}-1} r_i^{(k)} \Delta t , \tag{6.16}$$

Aplicando la ecuación 6.14 obtenemos

$$N_{removed}^{(k)} = N^{(k)} r^{(k)} \Delta t , \tag{6.17}$$

Usando la ec. 6.17, las ecuaciones 6.9 – 6.10 pasan a ser:

$$T_{CTS} \approx \left(\frac{T_{oc}}{r\Delta t} + T_{bk}\right) \cdot g^3 \cdot N^{3/2} \approx \mathcal{O}(N^{3/2}) \,, \tag{6.18}$$

$$T_{CTS}^{(RA)} \approx \frac{T_{oc}}{r\Delta t} + T_{bk} \,. \tag{6.19}$$

La ecuación 6.18 muestra que el coste computacional total es $\mathcal{O}(N^{3/2})$, mientras que la ecuación 6.19 muestra que el coste por átomo extraído es independiente de N. Velocidades de extracción r mayores conllevan un menor coste computacional por átomo extraídos, debido a las mayores reducciones de ocupación inducidas (ecuación 2.6). Esto está relacionado con el hecho de que velocidades de atacado r más altas requieren menos pasos de tiempo K para eliminar cierta cantidad de átomos. Asimismo, escoger un paso de tiempo mayor Δt conlleva reducciones de ocupación mayores (ecuación 2.6), menos pasos de tiempo K y un coste computacional menor por átomo extraído (ecuación 6.19). A pesar de que las simulaciones pueden ser rápidas computacionalmente para valores grandes de r y/o de Δt, será a expensas de introducir mayores errores respecto al modelo teórico.

6.2.2. ACC de Paso de Tiempo Variable

El método VTS-CCA hace avanzar el tiempo usando el mínimo incremento de tiempo obtenido tras evaluar la ecuación 6.1 para todos los átomos superficiales. En la simulación de un proceso tradicional de micromecanizado basado en grabado húmedo, miles de átomos superficiales cohabitan en el frente de atacado, teniendo cada uno velocidades de extracción y ocupación distintos. Esta heterogeneidad es especialmente pronunciada cuando planos de atacado rápidos y lentos aparecen de forma simultánea en el frente, como es el caso de sistemas con esquinas convexas. En este caso, los átomos superficiales localizados en las esquinas aparecerán y desaparecerán muy rápidamente, mientras que los átomos situados en las paredes laterales tienen tiempos de vida mucho más largos. Una implementación exacta del modelo requiere avanzar desde el tiempo de extracción de un átomo en una esquina hasta el siguiente momento de eliminación en

otra (o la misma) esquina, con la peculiaridad adicional de que la ocupación de todos los átomos superficiales debe ser reducida en cada paso de tiempo, aunque sólo uno o unos pocos átomos sean eliminados.

Como resultado, el número de átomos $N_{removed}^{(k)}$ que aparecen en las ecuaciones 6.9 – 6.10 puede ser estimado aproximadamente en la implementación VTS-CCA como el número de átomos que tiene la máxima velocidad de extracción $n_{r_{max}}$. Por lo tanto, las ecuaciones 6.9 – 6.10 pasan a ser:

$$T_{VTS} \approx \left(T_{oc} \frac{N}{n_{r_{max}}} + T_{bk} \right) g^3 \cdot N^{3/2} \approx \mathcal{O}(N^{5/2}) \,, \tag{6.20}$$

$$T_{VTS}^{(RA)} \approx T_{oc} \frac{N}{n_{r_{max}}} + T_{bk} \,. \tag{6.21}$$

La ecuación 6.21 muestra que N aparece ahora en el coste computacional por átomo extraído. De acuerdo con la ecuación 6.20, esto significa que el coste computacional total de una simulación basada en el método VTS-CCA es $\mathcal{O}(N^{2.5})$: simulaciones de estructuras mayores requieren, no sólo mayores superficies, sino también más pasos de tiempo ($K = g^3 N^{3/2}/n_{r_{max}}$). A pesar de que la implementación VTS-CCA no se ve afectada por la velocidad de extracción global r del sistema, el coste computacional para obtener una implementación exacta es una potencia dependiente del tamaño de la superficie. Como ya se sabía de forma empírica, una simulación basada en el VTS-CCA puede tardar muchas horas o incluso varios días.

En conclusión, en esta sección hemos analizado la complejidad de las implementaciones existentes de los ACC, obteniendo que los costes computacionales son $\mathcal{O}(N^{3/2})$ para CTS-CCA y $\mathcal{O}(N^{5/2})$ para VTS-CCA. La dependencia de CTS-CCA con respecto a la velocidad media de atacado conlleva que la simulación de atacantes con bajas temperaturas o concentraciones (donde las velocidades de extracción son bajas) sea mucho más ineficiente que las simulaciones de atacantes muy reactivos. Finalmente, a pesar que el método VTS-CCA es independiente de r, la dependencia del coste computacional como una potencia elevada del tamaño de la superficie hace de este método poco apropiado para simulaciones reales. Para

solventar estas deficiencias, en la siguiente sección se presenta una implementación del ACC nueva y exacta.

6.3. Enunciado de un ACC Eficiente

Un ACC eficiente es aquel cuyo coste computacional es lo más pequeño posible. En esta sección presentamos una nueva metodología para incrementar la eficiencia del modelo ACC sin introducir errores en la evolución temporal.

6.3.1. Tiempo Anticipado de Extracción

En primer lugar, para definir esta nueva implementación debemos presentar un nuevo concepto: el *Tiempo Anticipado de Extracción* (PRT, del inglés *Predicted Removal Time*). El PRT de un determinado átomo i es el instante calculado (o tiempo anticipado), cuando la ocupación pasará a ser cero y, entonces, el átomo será eliminado de la superficie:

$$PRT_i^{(k)} = t^{(k)} + \Delta t_i^{(k)} , \tag{6.22}$$

$$PRT_i^{(k)} = t^{(k)} + \frac{\Pi_i^{(k)}}{r_i^{(k)}} . \tag{6.23}$$

En esta definición $t^{(k)}$ es el valor actual del tiempo en la simulación y $\Delta t_i^{(k)}$ el incremento de tiempo requerido para el átomo i para ser completamente extraído (ecuación 6.1).

Las simulaciones basadas en VTS-CCA pueden ser realizadas de forma eficiente y precisa usando el PRT como la variable de estado principal en lugar de la ocupación. De hecho, únicamente tres simples modificaciones son necesarias para cambiar la implementación VTS-CCA en una implementación PRT-CCA:

1. Abandonar completamente el procedimiento que actualiza los valores de ocupación de todos los átomos superficiales.

2. Substituir el procedimiento que decide qué átomos necesitan ser eliminados, seleccionando ahora los átomos que tienen el mínimo PRT (en vez del criterio anterior basado en los átomos que habían alcanzado una ocupación cero).

3. Actualizar los valores PRT de los átomos superficiales cuya velocidades de extracción han sido cambiados, simplemente utilizando la siguiente ecuación:

$$PRT_i^{(k+1)} = t^{(k+1)} + \left(PRT_i^{(k)} - t^{(k+1)}\right) \frac{r_i^{(k)}}{r_i^{(k+1)}} , \tag{6.24}$$

donde $r_i^{(k)}$ y $r_i^{(k+1)}$ son las velocidades de extracción viejo y nuevo respectivamente, $PRT_i^{(k)}$ y $PRT_i^{(k+1)}$ son los PRT viejo y nuevo respectivamente y $t^{(k+1)}$ es el nuevo valor del tiempo en la simulación. La ecuación 6.24 se obtiene fácilmente particularizando 6.23 para $k+1$ y usando la ecuación 2.6 con $\Delta t = t^{(k+1)} - t^{(k)}$ con el objetivo de obtener

$$\Pi_i^{(k+1)} = \Pi_i^{(k)} - \left(t^{(k+1)} - t^{(k)}\right) r_i^{(k)}. \tag{6.25}$$

La figura 6.2 muestra una comparación gráfica de las simulaciones basadas en VTS-CCA y PRT-CCA sobre tres pasos de tiempo. Pese a que la variable interna en ambos métodos es distinta, ambos llegan al mismo resultado: por definición, el nuevo método propuesto selecciona la eliminación de los mismos átomos que el VTS-CCA.

6.3.2. Árboles Binarios de Búsqueda Autoequilibrados

El alto coste computacional de las implementaciones tradicionales de los ACC es debido a la necesidad de visitar todos los átomos superficiales en cada paso de tiempo con el fin de reducir su ocupación, aunque simplemente unos pocos átomos fueran eliminados. A pesar de que todos los átomos superficiales están marcados con su valor correspondiente de PRT,

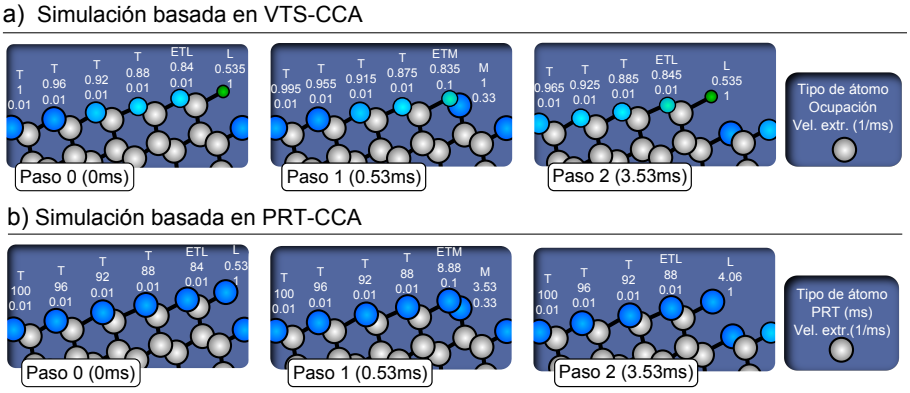

Figura 6.2: Evolución de los parámetros internos de los átomos durante el atacado de una superficie de orientación $(h+2, h+2, h)$. Los campos mostrados son: clasificación del átomo [127], ocupación (a)/PRT (b) y la velocidad de reducción de la ocupación.

la filosofía detrás de esta nueva implementación es la completa eliminación de la necesidad de actualizar parámetros de forma regular. De hecho, durante una secuencia de pasos de tiempo, no es necesario realizar cambios en el PRT de ningún átomo, a no ser que su vecindad haya cambiado. Esto debería reducir de forma considerable el coste computacional. Sin embargo, este objetivo es únicamente posible si una lista de átomos ordenada por su PRT puede ser mantenida de forma eficiente de manera que el próximo átomo a eliminarse será siempre el primero de la lista (en vez de buscarlo a lo largo de toda la lista de átomos superficiales).

El problema en este punto es el de encontrar la estructura de datos más eficiente que pueda manejar una lista ordenada dinamicamente. Las listas ordenadas basadas en vectores clásicos poseen un coste de acceso de $\mathcal{O}(1)$ pero añadir o eliminar un elemento tiene un coste de $\mathcal{O}(N)$. Las listas ordenadas basadas en listas enlazadas poseen costes de adición/eliminación de $\mathcal{O}(1)$, pero el acceso a un elemento posee un coste de $\mathcal{O}(N)$. Como alternativa más eficiente, proponemos el uso de los llamados *Árboles de Búsqueda Binarios Autoequilibrados* (SB-BST, del inglés *Self-Balanced Binary*

Search Tree), los cuales poseen un coste de acceso/adición/eliminación de $\mathcal{O}(log(N))$.

Los *Árboles Binarios* (BTs, del inglés *Binary Tree*) son estructuras de datos donde cada dato es almacenado en un nodo. Cada nodo de un BT puede tener hasta dos nodos hijos, formando una estructura que se asemeja a la forma de un árbol, donde cualquier nodo puede ser accedido siguiendo los enlaces entre el primer nodo o *nodo raíz* (figura 6.3(b)). Los últimos nodos, los cuales no poseen hijos, son denominados como *nodos hoja*. Los *Árboles de Búsqueda Binarios* (BST, del inglés *Binary Search Tree*) son BTs donde el hijo izquierdo tiene siempre almacenado un valor menor que el del padre, mientras que el hijo derecho tiene siempre un valor mayor. Como ejemplo, el BT mostrado en la figura 6.3(b) es realmente un BST. La mayor ventaja de los BST es el hecho de que el acceso a un elemento posee un coste de $\mathcal{O}(\lceil Log_2(N) \rceil)$ [175], donde $\lceil x \rceil$ es el siguiente entero mayor o igual a x. La razón de este coste reducido es que atravesar el árbol nodo por nodo se realiza verticalmente: desde el nodo raíz hasta los nodos hojas. En cada nivel, el valor buscado es mayor (subárbol izquierdo) o menor (subárbol derecho) que el nodo actual, por lo tanto el número de nodos visitado es menor o igual a la profundidad del árbol, la cual es $\lceil Log_2(N) \rceil$.

El coste de insertar/eliminar un nodo en un BST es independiente de N, $\mathcal{O}(1)$. Simplemente es necesario romper los enlaces locales de un padre y su hijo izquierdo/derecho y crear nuevos enlaces entre los nuevos/viejos padres/hijos. Sin embargo, mediante la sucesiva adición/eliminación de elementos, la estructura de un BST puede degenerar y convertirse en una lista enlazada si muchas inserciones son realizadas en la misma rama repetidamente, deteriorando por tanto el coste de acceso a $\mathcal{O}(N)$. En estos casos, el BST está desbalanceado, con una rama muy superficial (pocos niveles) y otra rama muy profunda (muchos niveles). Con el objetivo de mantener una buena velocidad de acceso, la forma del árbol debe mantenerse balanceada o, en otras palabras, la profundidad del árbol debe minimizarse.

Los SB-BST, introducidos por Adelson-Velskii *et. al* [176], definen varios procedimientos a realizar tras la inserción o la eliminación de un nodo en un árbol con el objetivo de que permanezca balanceado y que la profun-

didad se mantenga reducida. El precio a pagar por minimizar la profundidad del árbol, (y por tanto los tiempos de acceso), es la necesidad de realizar una serie de operaciones de mantenimiento del árbol (*tree-bookkeeping*). Los SB-BST aseguran costes logarítmicos para el acceso de datos, inserción y eliminación, las cuales son propiedades atractivas para colecciones de datos grandes, donde el coste extra debido al mantenimiento es compensado de forma substancial por los tiempos de acceso minimizados.

La figura 6.3 muestra una comparación del procedimiento de simulación para avanzar un paso de tiempo en una implementación tradicional del VTS-CCA basada en una lista enlazada y la implementación PRT-CCA propuesta basada en el uso de un SB-BST. La figura muestra el coste computacional asociado para cada operación, por lo que explica de forma gráfica la ventaja computacional de PRT-CCA sobre VTS-CCA.

6.3.3. Coste Computacional del ACC basado en PRT

Las dos tareas que consumen la mayor parte de los recursos de proceso en un simulación de un PRT-CCA son los pasos 1 y 3 de la figura 6.3(b), es decir el *Atravesado Vertical del Árbol* (TVT, del inglés *Tree Vertical Traversal*) con el fin de encontrar el próximo átomo a ser eliminado (T_{tvt}) y las *Tareas de Mantenimiento del Árbol* (TBK, del inglés *Tree Book Keeping*) que aseguran la profundidad mínima (T_{tbk}). Debido a que el coste computacional de ambas tareas es $\mathcal{O}(\log_2 N)$ por átomo eliminado y el número de átomos eliminados es $N_{removed}^{total} = g^3 N^{3/2}$, el tiempo computacional para realizar una simulación basada en PRT-CCA puede ser escrito como:

$$T_{PRT} \approx [(T_{tvt} + T_{tbk}) \cdot \log_2 N]\, g^3 \cdot N^{3/2} \approx \mathcal{O}(N^{3/2} \log N)\ , \quad (6.26)$$

$$T_{PRT}^{(RA)} \approx (T_{tvt} + T_{tbk}) \cdot \log_2 N\ . \quad (6.27)$$

Por lo tanto, el coste computacional del PRT-CCA es $\mathcal{O}(N^{3/2} \log N)$. Sin reducir la precisión de la simulación, el uso de PRT-CCA resulta en una reducción dramática del coste con respecto a VTS-CCA, cuyo coste es $\mathcal{O}(N^{5/2})$. En comparación con CTS-CCA, el coste computacional de PRT-CCA es independiente de la reactividad del atacante y del tamaño

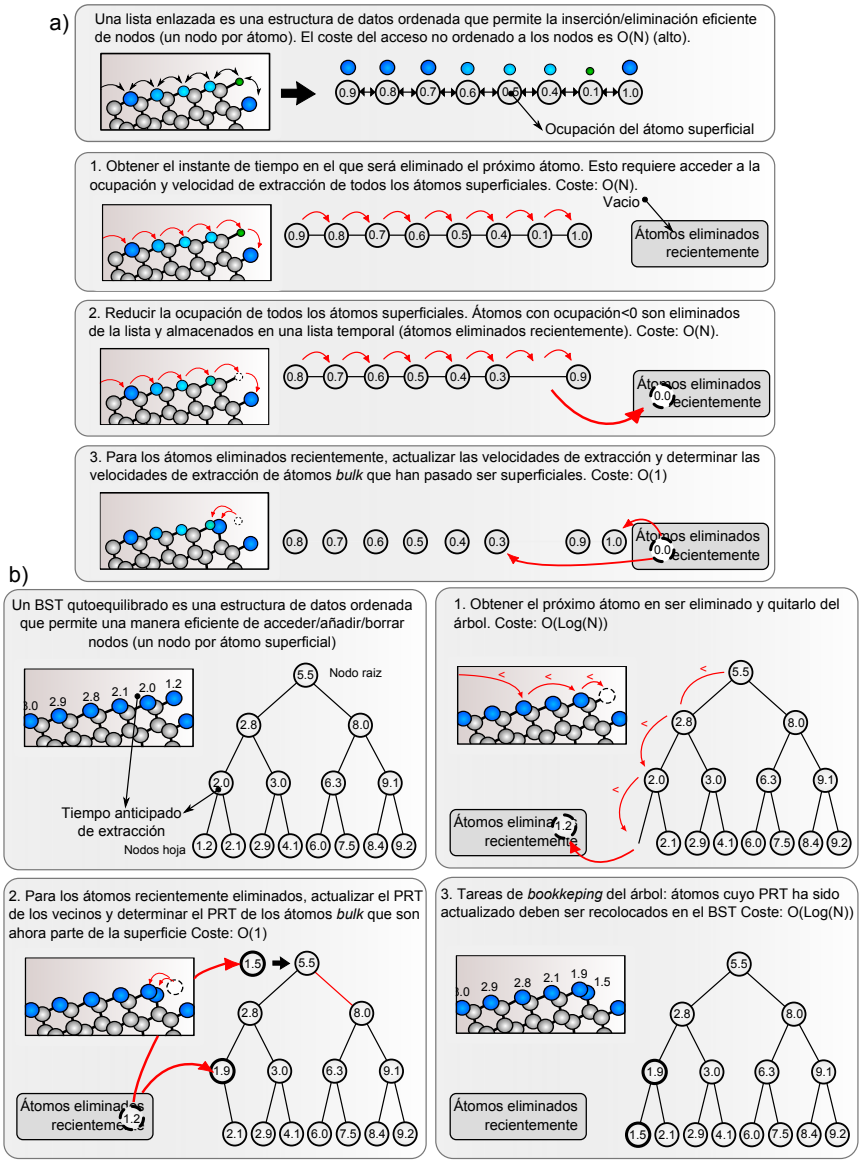

Figura 6.3: Procedimiento para simular un paso de tiempo. (a) VTS-CCA basado en una lista enlazada. (b) PRT-CCA basado en un SB-BST

del paso de tiempo (debido a que Δt no es un parámetro en el método PRT). Debido a que el coste del CTS-CCA es $\mathcal{O}(\frac{N^{3/2}}{r\Delta t})$, la eficiencia relativa de ambos métodos dependerá de los detalles particulares de los sistemas simulados. Por otro lado, PRT-CCA garantiza la obtención de resultados exactos, mientras que la precisión de CTS variará según las características de la simulación.

6.4. Resultados

Con el objetivo de comparar las implementaciones tradicionales y la nueva formulación propuesta en la sección anterior, se ha desarrollado un simulador de grabado húmedo en lenguaje Java el cual implementa el CTS-CCA, el VTS-CCA y el PRT-CCA. Mientras que las dos primeras implementaciones usan una lista enlazada, PRT-CCA se ha implementado usando un SB-BST de tipo *rojo-negro* [177]. Una de las ventajas de la utilización de Java como lenguaje es la existencia en sus librerías de las estructuras de datos necesarias para nuestras implementaciones: (*java.util.LinkedList* para la lista enlazada y *java.util.TreeSet* para el árbol autobalanceado respectivamente). Cada nodo almacenado en dichas estructuras almacena un tipo de datos que representa un átomo del substrato. Dicho tipo de dato almacena internamente campos como la velocidad de atacado, ocupación/PRT y punteros a sus cuatro átomos vecinos con el objetivo de acelerar los cálculos donde la información sobre la vecindad es necesaria.

6.4.1. Sistema Simulado

La superficie simulada consiste en un chip cuadrado el cual puede ser parte de una oblea de Silicio de tipo (100). Sobre esta superficie ha sido aplicada una máscara cuadrada de nitruro de silicio que cubre el 45 % del área y cuyos lados están alineados con la dirección $< 110 >$. Dicha máscara también cubre el perímetro del chip. Un esquema de la máscara puede entontrarse en la figura 6.4 (a)

La estructura obtenida a partir del atacado de la superficie con la

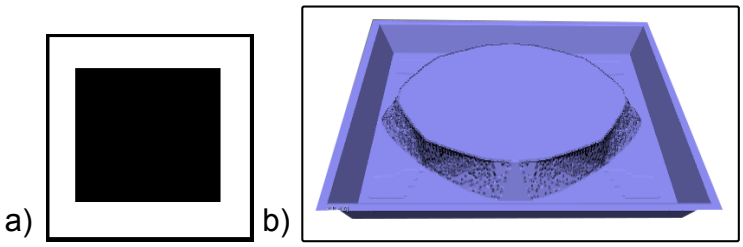

Figura 6.4: Simulación realizada para la comparación de rendimiento entre CTS-CCA, VTS-CCA y PRT-CCA. (a) Diseño de la máscara aplicada. (b) Estado final típico tras el proceso de atacado.

máscara aplicada se caracteriza por un fuerte *underetching* en las cuatro esquinas convexas de la máscara cuadrada y la formación de planos (111) en el perímetro, tal y como muestra la figura 6.4 (b). Las simulaciones se han realizado para el mismo atacante a cuatro temperaturas distintas, dado que esto nos permite el análisis de la eficiencia relativa del método CTS-CCA en función de la reactividad del sistema (mayor temperatura supone mayores r). Los datos experimentales utilizados para calibrar el ACC para las cuatro temperaturas han sido obtenidos de [106]. Para cada temperatura han sido simulados cuatro tamaños de superficie, de manera que la superficie total de cada tamaño dobla la del anterior ($N_{i+1} = e \cdot N_i$, donde $e = 2$). Con el fin de que la relación de aspecto de la estructura tridimensional resultante se mantenga constante (profundidad = 1/15 de la longitud del chip), la profundidad de grabado se incrementa por un factor de $e^{1/2} = \sqrt{2}$ por cada duplicación de la superficie. Debido a esto, el tiempo de atacado se incrementa el mismo factor ($e^{1/2} = \sqrt{2}$) por duplicación. Esto resulta en un incremento total de volumen de las estructuras atacadas por un factor de $e^{3/2} = 2^{3/2} \approx 2.83$ por duplicación de superficie.

La computadora utilizada para realizar las simulaciones está equipada con un procesador Intel core i7 a 2.66GHz con 3 GB de memoria RAM DDR3 PC1066.

Tam. superficie (átomos) / Atacante	92x92			128x128			182x182			256x256		
	CTS	VTS	PRT	CTS	VTS	PRT	CTS	VTS	PRT	CTS	VTS	PRT
KOH 37wt 30C	4277	4343	4346	6092	6013	6065	8722	8641	8688	12409	12292	12348
	4277	48.13e3	21.52e3	6092	139.7e3	94.95e3	8722	391.3e3	224.1e3	12409	1.129e6	475.0e3
	870	**15620**	**420**	**2690**	**96.68e3**	**1360**	**9693**	**758.2e3**	**5010**	**44.69e3**	**8.353e6**	**19.07e3**
	79.52e3	81.40e3	81.36e3	228.7e3	225.9e3	227.8e3	689.8e3	684.9e3	689.6e3	2.024e6	2.013e6	2.021e6
	10.940	*191.87*	*5.134*	*11.760*	*427.96*	*5.010*	*14.051*	*1107.0*	*7.265*	*22.070*	*147.6*	*9.434*
	350	***4096***	***120***	***346***	***8148***	***134***	***339***	***16240***	***147***	***335***	***32752***	***158***
	0.031	*0.047*	*0.043*	*0.034*	*0.053*	*0.044*	*0.041*	*0.068*	*0.049*	*0.066*	*0.127*	*0.06*
	1	1	1	1.09	1.12	1.03	1.33	1.46	1.16	2.11	2.7	1.4
KOH 37wt 50C	713	728	730	1045	1021	1035	1464	1445	1445	2059	2029	2038
	713	37.52e3	28.59e3	1045	119.9e3	111.4e3	1464	389.4e3	340.5e3	2059	1.124e6	874.0e3
	330	**10900**	**380**	**1100**	**82.04e3**	**1270**	**3958**	**662.0e3**	**4720**	**15.99e3**	**7.551e6**	**18.65e3**
	59.58e3	64.12e3	64.27e3	180.9e3	183.1e3	186.6e3	539.9e3	553.2e3	554.5e3	1.584e6	1.618e6	1.626e6
	5.538	*169.98*	*5.878*	*6.077*	*450.44*	*6.881*	*7.331*	*1196.6*	*8.511*	*9.652*	*4664.4*	*11.469*
	101	***3850***	***131***	***100***	***8487***	***145***	***97***	***18709***	***158***	***96***	***37437***	***174***
	0.055	*0.044*	*0.045*	*0.061*	*0.053*	*0.047*	*0.076*	*0.064*	*0.054*	*0.101*	*0.125*	*0.066*
	1	1	1	1.11	1.2	1.06	1.38	1.45	1.2	1.83	2.82	1.47
KOH 37wt 70C	372	364	362	535	523	627	783	773	772	1122	1121	1120
	372	76.35e3	36.95e3	535	217.4e3	205.4e3	783	642.9e3	592.2e3	1122	1.563e6	1.665e3
	420	**20380**	**560**	**1310**	**143.9e3**	**1980**	**4829**	**1.151e6**	**7440**	**15.85e3**	**10.83e6**	**28.34e3**
	90.98e3	93.19e3	92.98e3	263.8e3	271.5e3	274.7e3	806.8e3	841.3e6	839.2e3	2.365e6	2.502e6	2.496e6
	4.616	*217.27*	*6.023*	*4.966*	*528.12*	*7.212*	*5.985*	*1368.1*	*8.865*	*6.700*	*4328.1*	*11.350*
	61	***5517***	***143***	***61***	***10927***	***157***	***60***	***21897***	***169***	***60***	***36743***	***183***
	0.076	*0.039*	*0.042*	*0.081*	0.048	*0.046*	*0.100*	*0.062*	*0.052*	*0.112*	*0.118*	*0.062*
	1	1	1	1.08	1.23	1.1	1.32	1.59	1.25	1.48	2.99	1.47
KOH 37wt 90C	156	138	139	226	199	199	327	289	290	467	410	411
	156	61.84e3	32.61e3	226	196.9e3	183.3e3	327	611.4e3	551.1e3	467	811.4e3	1.117e6
	390	**17500**	**566**	**1200**	**132.5e3**	**1950**	**4202**	**1.233e6**	**8250**	**13.24e3**	**6.049e6**	**25.38e3**
	96.52e3	94.49e3	95.07e3	280.7e3	279.2e3	279.3e3	848.5e3	858.7e3	861.6e3	2.472e6	2.503e6	2.504e6
	4.041	*185.18*	*5.890*	*4.274*	*474.77*	*6.983*	*4.952*	*1436.4*	*8.433*	*5.288*	*2416.4*	*10.133*
	46	***4687***	***143***	***46***	***10137***	***154***	***46***	***21251***	***168***	***45***	***19152***	***181***
	0.088	*0.04*	*0.041*	*0.093*	*0.047*	*0.045*	*0.108*	*0.068*	*0.05*	*0.118*	*0.126*	*0.056*
	1	1	1	1.06	1.19	1.1	1.23	1.71	1.22	1.34	3.19	1.36

t
K
T
$N^{total}_{removed}$
$T/N^{total}_{removed}$
q
$T/N^{total}_{visited}$
p

t:Tiempo de atacado (ms), K: Pasos de tiempo realizados, **T: Tiempo de ejecución (ms),** $N^{total}_{removed}$: Átomos extraidos $T/N^{total}_{removed}$ *:Tiempo por átomo extraido (us)*
q: Ratio átomos Visitados/extraidos, $T/N^{total}_{visited}$ *: Tiempo por átomo visitado (us),* p: Ratio de penaliz. tamaño-rendimiento

Figura 6.5: Comparación de rendimiento entre CTS-CCA, VTS-CCA y PRT-CCA. Tabla de resultados para un amplio rango de características(ver leyenda) de los tres métodos en diversos atacantes y tamaños.

6.4.2. Parámetros de la Simulación

La tabla mostrada en la figura 6.5 muestra los datos más relevantes para todas las simulaciones realizadas. Como se describe brevemente en la leyenda de la esquina inferior-derecha, los resultados de cada sistema simulado están caracterizados por un total de ocho parámetros:

1. t, el tiempo de atacado de la superficie, en ms.
2. K, el número total de pasos de simulación necesarios para obtener la profundidad indicada.
3. T, el tiempo computacional requerido, en ms.
4. $N_{removed}^{total}$, el número total de átomos extraídos del substrato de silicio.
5. $T^{(RA)} = T/N_{removel}^{total}$, el tiempo computacional requerido por cada átomo extraído, en us
6. $q = N_{visited}^{total}/N_{removel}^{total}$, el ratio de la cantidad total de átomos visitado en relación a los átomos extraído (ineficiencia del método).
7. $T^{(VA)} = T/N_{visited}^{total}$, tiempo computacional requerido por átomo visitado, en us.
8. $p = T^{(VA)}/T_{92x92}^{(VA)}$, el ratio de penalización asociado al incremento del tamaño del sistema.

La definición de p se debe al hecho de que en la práctica el tiempo de acceso medio por nodo en una lista enlazada (y en un SB-BST) no es exactamente constante como se espera teóricamente, sino que crece ligeramente con el tamaño de N, como se muestra en la figura 6.6. Esta curva se comporta de una manera no trivial y, de hecho, depende en gran medida de la arquitectura de la computadora sobre la que se esté ejecutando el algoritmo. El incremento en los tiempos de acceso es debido a que, en un algoritmo ejecutado por una computadora, los tiempos de acceso a datos de la memoria dependen en gran medida de los patrones de acceso a estos datos. Este efecto es debido a la existencia de una jerarquía de memorias

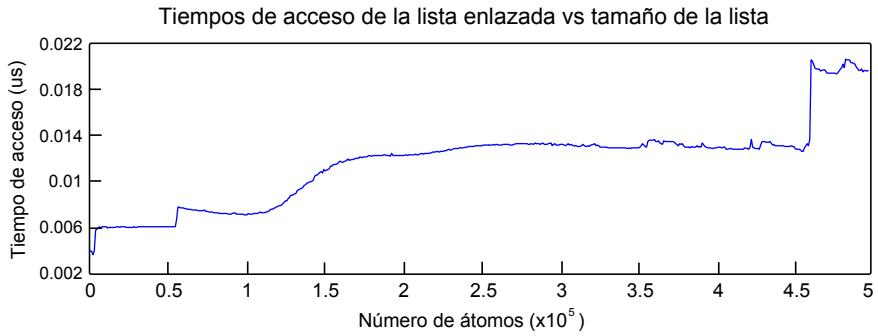

Figura 6.6: Tiempo medio de acceso a un nodo de la lista enlazada (acceso secuencial), en función de la cantidad de nodos en la lista

caché que posee el ordenador con el objetivo de acelerar la lectura de los datos accedidos con mayor frecuencia [178]. A medida que el tamaño de la lista enlazada aumenta, los nodos son leídos desde memorias más lentas, incrementando por tanto los tiempos de acceso. Una forma efectiva de eliminar esta contribución en las medidas de T es la de definir el concepto de *tiempo computacional corregido*: $T_c = T/p$ y el correspondiente *tiempo computacional corregido por átomo extraído*: $T_c^{(RA)} = T^{(RA)}/p$. En los datos mostrados en la tabla de la figura 6.5, p indica los tiempos de acceso como un múltiplo del tiempo para el sistema más pequeño (92x92) para cada temperatura y para cada uno de los tres métodos computacionales.

La manera en que $N_{visited}^{total}$ es determinada en las simulaciones es importante debido a que afecta al valor de la ineficiencia q. Para CTS-CCA, $N_{visited}^{total}$ es el valor tomado como la suma de todos los átomos visitados para reducir las ocupaciones y actualizar las vecindades. Para VTS-CCA, $N_{visited}^{total}$ incluye ademas las visitas necesarias para calcular el próximo paso de tiempo. Para el método PRT-CCA, $N_{visited}^{total}$ contiene las visitas requeridas para obtener el átomo con PRT_{min} y actualizar la vecindad e insertar/eliminar/recolocar nodos en el SB-BST.

6.4.3. Eficiencia Computacional de los Tres Métodos

Para CTS-CCA (columnas blancas de la tabla de la figura 6.5), la eficiencia (ligada directamente a valores pequeños de q) depende fuertemente de la reactividad del atacante aplicado r. Las simulaciones con la temperatura más baja (menor r) requieren visitar cientos de átomos (≈ 350) para poder extraer un único átomo de silicio, mientras que para la temperatura más baja, q se reduce a unas pocas decenas (≈ 45). Esto sucede por el simple hecho de que valores de r mayores dan lugar a una reducción media de ocupación mayor, y por lo tanto menos pasos de tiempo K son necesarios para extraer cierto átomo. Asimismo, los resultados de la tabla demuestran que q es independiente del tamaño de la superficie.

El comportamiento previo de q para el CTS-CCA es confirmado por el comportamiento del tiempo computacional por átomo extraído $T^{(RA)} = T/N^{total}_{removed}$, el cual se representa en la gráfica de la figura 6.7 (a) usando líneas continuas. Tal y como mostraba la ecuación 6.19, $T^{(RA)}$ aumenta a medida que r disminuye por la misma razón que lo hace q. La figura 6.7 (b) muestra que $T^{(RA)}$ aumenta a medida que el tamaño del sistema aumenta, al contrario de lo que podría esperarse en un principio de acuerdo con la ecuación 6.19. Esta divergencia es debida simplemente al incremento del tiempo de acceso práctico a los nodos de la lista enlazada, como se ha explicado en relación a la figura 6.6. Cuando se dibuja el tiempo computacional corregido por átomo extraído $T_c^{(RA)} = T^{(RA)}/p$ (figura 6.8 (a)), se observa que dicho tiempo no varia al cambiar el tamaño de la superficie, como la ecuación 6.19 indica.

Para VTS-CCA (columnas gris claro de la tabla de la figura 6.5), la característica más relevante es la inmensa cantidad de pasos de tiempo K que son necesarios para completar las simulaciones en comparación con CTS-CCA. Esta característica se ve reflejada como una ineficiencia q extremadamente grande, tipicamente entre uno y tres órdenes de magnitud mayor que la ineficiencia de CTS-CCA para el mismo sistema. Asimismo, q parece aumentar a medida que se incrementa el tamaño de la superficie. Como se esperaba de la ecuación 6.21, las líneas punteadas de la figura 6.7 (a) muestran que el coste computacional por átomo para VTS-CCA es independiente de la reactividad del atacante. La figura confirma la in-

mensa diferencia entre los tiempos de proceso de VTS-CCA y los otros dos métodos simulados. Asimismo, la figura 6.7 (b) muestra que $T^{(RA)}$ incrementa al crecer N. De hecho, cuando se representa el tiempo computacional corregido por átomo extraído $T_c^{(RA)} = T^{(RA)}/p$ en la figura 6.8 (a), observamos que $T_c^{(RA)}$ para VTS-CCA se incrementa linealmente con N con una pendiente = 0.922 $\approx$ 1.0), por lo que confirma el coste teórico obtenido en 6.21.

El número de átomos eliminados por paso de tiempo en el método VTS-CCA (obtenido dividiendo los valores de $N_{removed}^{total}$ por los valores de K) se sitúa entre 1.5 y 3 para todo el rango de temperaturas y tamaños. Esto confirma el hecho de que átomos altamente reactivos están continuamente emergiendo a la superficie y siendo eliminados y, como resultado, sólo uno o unos pocos átomos son extraídos en cada paso de tiempo usando este método.

Para el modelo PRT-CCA (columnas gris oscuro en la tabla de la figura 6.5), los valores q son mucho menores que en las simulaciones basadas en VTS-CCA, entre uno y tres órdenes de magnitud. Esto significa que PRT-CCA es habitualmente más de dos ordenes de magnitud más eficiente que VTS-CCA sin afectar la precisión desde un punto de vista teórico. Esto se puede apreciar claramente en la figura 6.7 (a), donde las líneas que representan PRT-CCA (líneas discontinuas) poseen valores mucho menores que las que representan VTS-CCA. Comparando este nuevo método con CTS-CCA, la figura 6.7 (a) evidencia que PRT-CCA es más eficiente para atacantes menos reactivos, en mayor medida por la pérdida de eficiencia experimentada por CTS-CCA cuando r disminuye. Por otro lado, a pesar que CTS-CCA es más eficiente que PRT-CCA para atacantes más reactivos, esto da lugar también a la aparición de mayores errores de simulación, como veremos en detalle en la sección 6.4.4.

Como se muestra en la figura 6.7 (b) y en la tabla de la figura 6.5, los valores de $T^{(RA)}$ y q para el método PRT-CCA se incrementan de forma moderada a medida que aumenta el tamaño de la superficie. Esto es debido al hecho de que el coste de acceso al árbol binario y las tareas de mantenimiento del árbol pasan a ser más costosas a medida que el tamaño de la superficie se incrementa, como se describe en la ecuación 6.27 (dependen-

cia logarítmica). De hecho, cuando analizamos el tiempo computacional corregido por átomo extraído para PRT-CCA (figura 6.8 (b)), obtenemos que $T_c^{(RA)}/\log(N)$ permanece constante al variar el tamaño de la superficie (pendiente = 0.025), lo que significa que el coste de extracción por átomo sigue una dependencia $\log N$, tal y como la ecuación 6.27 indica.

El coste computacional de los tres métodos está resumido en las figuras 6.8 (c)-(d). La figura 6.8 (c) muestras que el tiempo computacional corregido total $T_c = T/p$ crece en función de N con una pendiente de $\mathcal{O}(N^{2.48}) \approx \mathcal{O}(N^{5/2})$ para VTS-CCA y $\mathcal{O}(N^{1.55}) \approx \mathcal{O}(N^{3/2})$ para CTS-CCA, valores muy cercanos a los predichos por las ecuaciones 6.20 y 6.18 respectivamente. De la misma manera, la figura 6.8 (d) muestra que, para PRT-CCA T_c crece con una pendiente $\mathcal{O}(N^{1.58}\log N) \approx \mathcal{O}(N^{3/2}\log N)$, lo cual evidencia también el cumplimiento de la ecuación 6.26.

Por último, los resultados obtenidos, adecuados a la teoría en términos de costes computacionales, demuestran que el número total de átomos extraídos $N_{removed}^{total}$ es proporcional a $N^{3/2}$, como se describe en la sección 6.2.

6.4.4. Errores Relativos con respecto al VTS-CCA

En esta sección analizaremos el resultado final de las simulaciones en los tres métodos, evaluando las similitudes y diferencias existentes para dos de los parámetros más importantes de una simulación: evolución del tiempo y anisotropía del atacante.

El valor final del tiempo de atacado t en una simulación es un parámetro muy importante a la hora de simular un proceso de grabado anisótropo para aplicaciones MEMS. Muy frecuentemente (como es el caso de las simulaciones realizadas en este apartado), el usuario definirá en la herramienta de simulación pertinente la profundidad de atacado que desea y el simulador será responsable de calcular el tiempo de atacado y propagar el sistema hasta el estado final correcto. Errores en t debido a la implementación del ACC se trasladan de forma directa en errores en la geometría final del sistema una vez el diseñador decida llevar los datos obtenidos desde el simulador a la práctica.

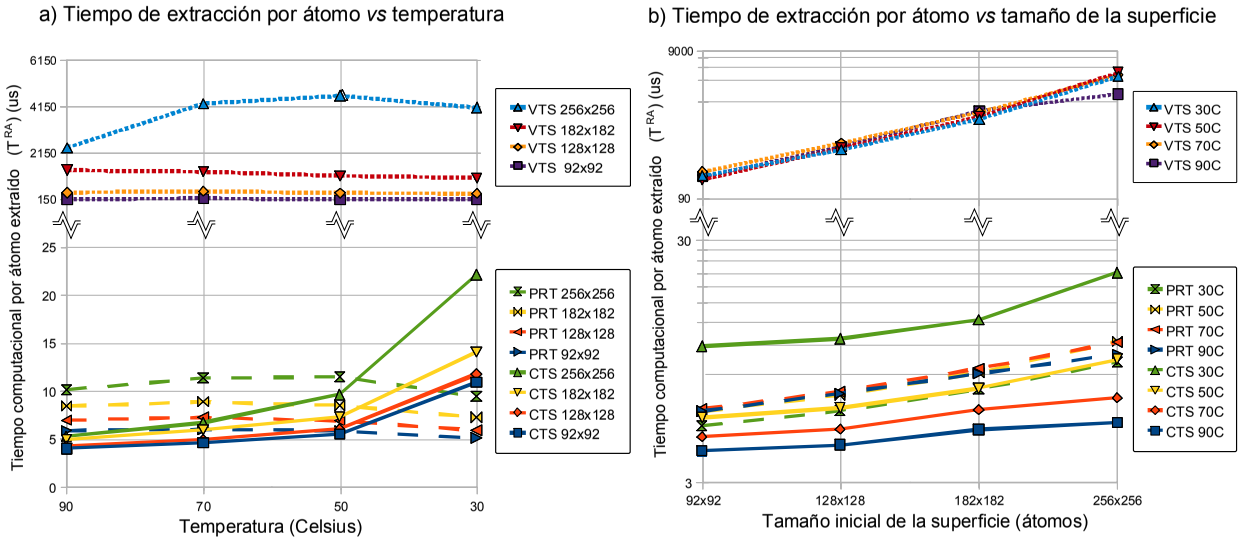

Figura 6.7: Comparación de velocidades de simulación entre CTS-CCA, VTS-CCA y PRT-CCA. (a) Tiempo de extracción de un átomo de silicio en función de la temperatura del atacante. (b) Tiempo de extracción de un átomo de silicio en función del tamaño de la superficie.

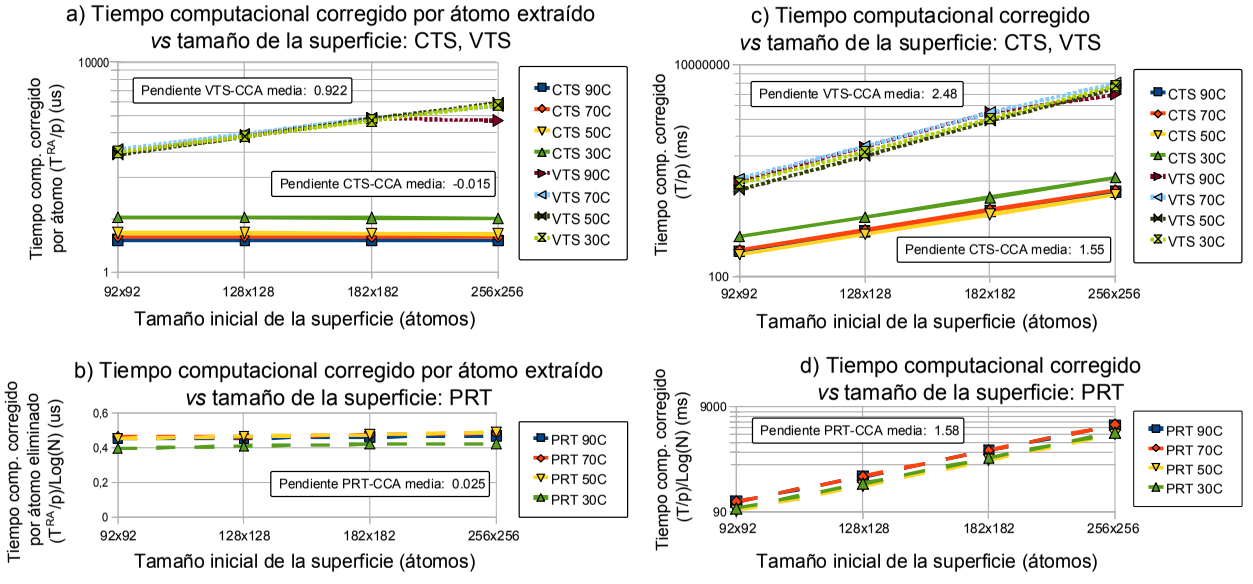

Figura 6.8: Comparación de tiempos de simulación para los métodos CTS-CCA, VTS-CCA y PRT-CCA. (a)-(b) Tiempo de extracción de un átomo de silicio (corregido) en función del atacante. (c)-(d) Tiempo total de simulación corregido en función del tamaño inicial de la superficie.

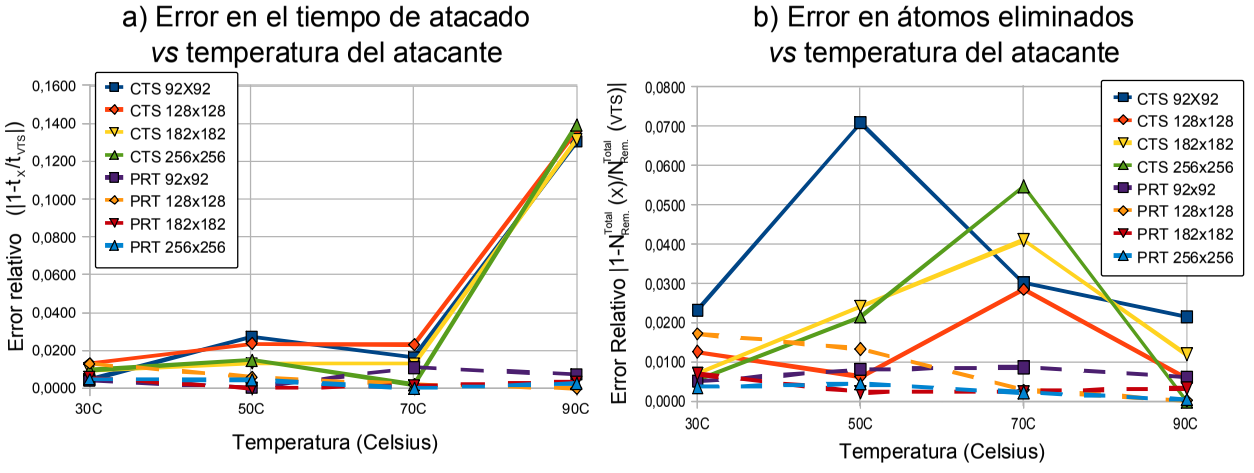

Figura 6.9: Errores relativos de los métodos CTS-CCA y PRT-CCA en función de los resultados de VTS-CCA. (a) Error en el tiempo de simulación para alcanzar la profundidad indicada. (b) Error en la cantidad de átomos extraídos de la superficie.

La figura 6.9 (a) muestra el error relativo en t con respecto al método exacto tradicional VTS-CCA para los sistemas simulados de la figura 6.5. Esta figura demuestra que el error para el método CTS-CCA se incrementa de forma drástica cuando el atacante simulado pasa a ser más reactivo, alcanzando un error máximo del 14 % con respecto a VTS-CCA. Este error es debido al hecho de que velocidades de extracción r mayores resultan en reducciones de ocupación mayores, haciendo más evidentes casos donde la ocupación se vuelve negativa. En principio, este error puede reducirse disminuyendo el paso de tiempo Δt. Por otro lado, esta corrección da lugar a tiempos de simulacióm más elevados.

En el caso de PRT-CCA, la figura 6.9 (a) muestra que los valores de t se mantienen muy cerca de VTS-CCA para todas las temperaturas y tamaños de la superficie. Debido a que PRT-CCA y VTS-CCA son teóricamente equivalentes, estos pequeños errores (típicamente menores del 1 %) tienen un origen numérico. En ambas implementaciones usamos variables de coma flotante de simple precisión para almacenar la ocupación y los valores de PRT respectivamente en la memoria de la computadora. La precisión de este formato es relativa al valor absoluto. Mientras que para el simulador basado en ocupación, las variables almacenan valores entre 0.0 y 1.0, el simulador basado en PRT almacena valores de tiempo t los cuales se incrementan a medida que avanza la simulación, lo que lleva a una pérdida de precisión en la parte decimal en comparación con el uso de la ocupación. Bajo estas circunstancias, las diferencias entre $\mathrm{PRT}^{(k)}$ y $t^{(k+1)}$ dan lugar a ligeras diferencias en el cálculo de $\mathrm{PRT}^{(k+1)}$ cuando se evalúa la ecuación 6.24. Asímismo, hemos comprobado que el uso de variables de coma flotante de doble precisión para PRT y t dan lugar a resultados del PRT-CCA totalmente similares a los de VTS-CCA. A pesar de los errores numéricos asociados al almacenamiento de las variables, la figura 6.9 (a) muestra que los resultados de PRT-CCA son mucho más cercanos a VTS-CCA que los resultados de CTS-CCA.

Una manera de evaluar con un único numero la similitud entre las geometrías obtenidas en las distintas simulaciones es considerar la cantidad de átomos extraídos $N_{removed}^{total}$. A pesar de que la profundidad a lo largo de la dirección $< 100 >$ es similar en todas las simulaciones del mismo

tamaño, diferencias en $N_{removed}^{total}$ indicarán diferencias en las velocidades de atacado de otras direcciones, y por lo tanto desviaciones en la anisotropía del atacante simulado y en la geometría final. La figura 6.9 (b) muestra que el error relativo en $N_{removed}^{total}$ con respecto a VTS-CCA es mayor en CTS-CCA (hasta el 7 %) mientras que se mantiene mucho más bajo para PRT-CCA (por debajo del 2 %). Los orígenes de estos errores para ambos métodos son los mismos que los explicados en relación a la figura 6.9 (a): Errores intrínsecos del método para CTS-CCA y errores numéricos para PRT-CCA.

En conclusión, el método propuesto, PRT-CCA, proporciona tiempos de simulación admisibles (incluso más rápidos que CTS-CCA para sistemas de baja reactividad), mientras que produce resultados mucho mas cercanos a los teóricos obtenidos mediante VTS-CCA, distorsionados únicamente por pequeños errores numéricos de precisión.

6.5. Aplicación Práctica

En esta sección analizamos las diferencias entre las simulaciones realizadas con CTS-CCA, VTS-CCA y PRT-CCA en una aplicación de ingeniería práctica. La estructura MEMS escogida es una sonda micromecánica de dos ejes caracterizada por tener un *beam*[3] de torsión para la detección de fuerzas verticales y un *cantilever*[3] doble para la detección de fuerzas horizontales [95]. La elección de esta microestructura es debida a su novedad, su fabricación basada en grabado anisótropo húmedo y los mínimos grosores del *beam* y el *cantilever*, que evidencian la necesidad de controlar de forma precisa las velocidades de atacado para diversas orientaciones superficiales, así como el tiempo global de atacado en las simulaciones.

Para la fabricación de esta estructura, el modelo ACC ha sido calibrado basándonos en los datos experimentales para KOH 40wt % a 70°C obtenidos por K. Sato *et al.* [104]. Debido a que la mayoría de las orientaciones de la microestructura pertenecen a las orientaciones $< 100 >$ y $< 111 >$

[3]Los *beams* y *cantilevers* son dos estructuras que aparecen con frecuencia en los MEMS. Un *beam* (*viga* en castellano) es una estructura utilizada para soportar cargas. Un *cantilever* es un *beam* con soporte en un único extremo.

en este sistema, los errores introducidos en la implementacion CTS-CCA se espera que aparezcan como una velocidad de atacado incorrecta en las caras (100). Debido a que la calibración del ACC ajusta de forma explícita la velocidad de atacado de la orientación $< 100 >$, las implementaciones exactas del modelo deberían simular la velocidad de atacado de este plano sin errores.

El sistema simulado consiste en un chip de silicio de $5300 \times 3003\ um^2$ con un grosor de 180 um obtenido a partir de una oblea (100) de silicio, con los bordes de dicho chip orientados en la dirección $< 100 >$. La microsonda de dos ejes puede considerarse como una microestructura con una alta relación de aspecto debido a que el *beam* y el *cantilever* poseen un grosor de unas pocas micras, mientras la longitud total del *cantilever* posee varios cientos de micras.

Asimismo, las simulaciones deben conseguir la suficiente precisión en las partes más finas, por lo que la resolución del modelo debe ser incrementada tanto como sea posible.

Las simulaciones se han realizado en una versión modificada de *GPUetch* (sección 4.2), donde el PRT-CCA ha sido implementado modificando las ecuaciones utilizadas para el VTS-CCA, sin implementar el árbol binario. De esta forma, aunque la velocidad de simulación del PRT-CCA es similar al VTS-CCA, este método es suficiente para comprobar los errores de precisión introducidos por CTS-CCA y PRT-CCA. El tamaño inicial de la superficie es de $1.59 \cdot 10^6$ átomos, y la microestructura final está formada por $766 \cdot 10^3$ átomos superficiales aproximadamente.

El proceso de fabricación usado en las simulaciones es descrito de forma gráfica en la figura 6.10 y se resume en los siguientes pasos:

1. Aplicar las máscaras 1 y 2 de nitruro de silicio (figura 6.10 (a)) en las partes superior e inferior de la superficie de silicio respectivamente (figura 6.10 (b)).

2. Aplicar atacado húmedo durante 45 minutos y eliminar las máscaras (figura 6.10 (c)).

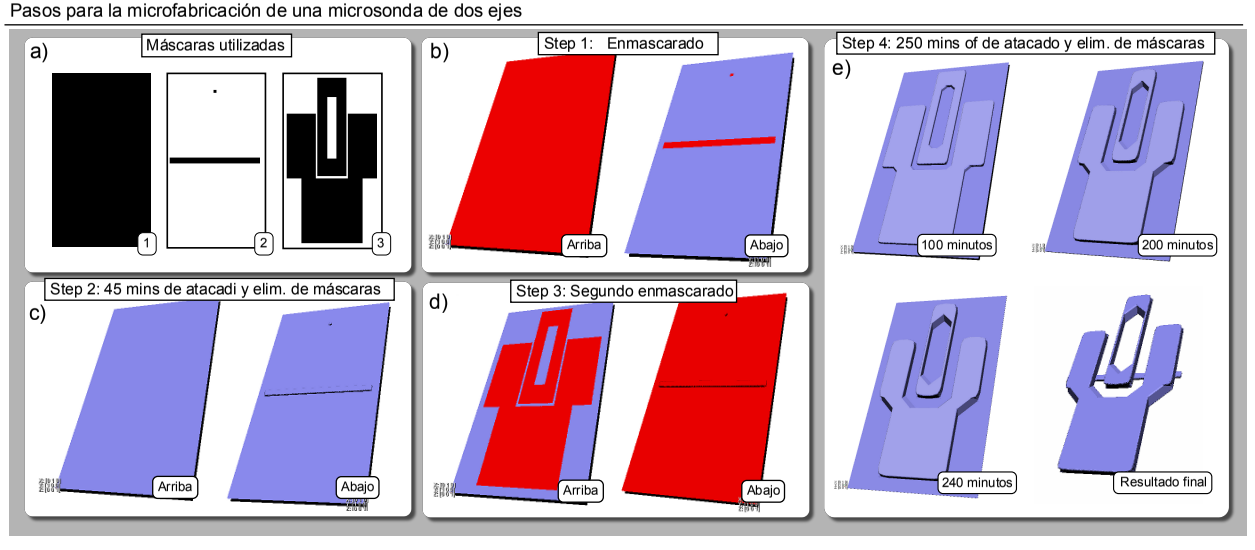

Figura 6.10: Proceso de fabricación propuesto de una microsonda de dos ejes [95]. (a) Máscaras utilizadas. (b)-(e) Pasos de fabricación.

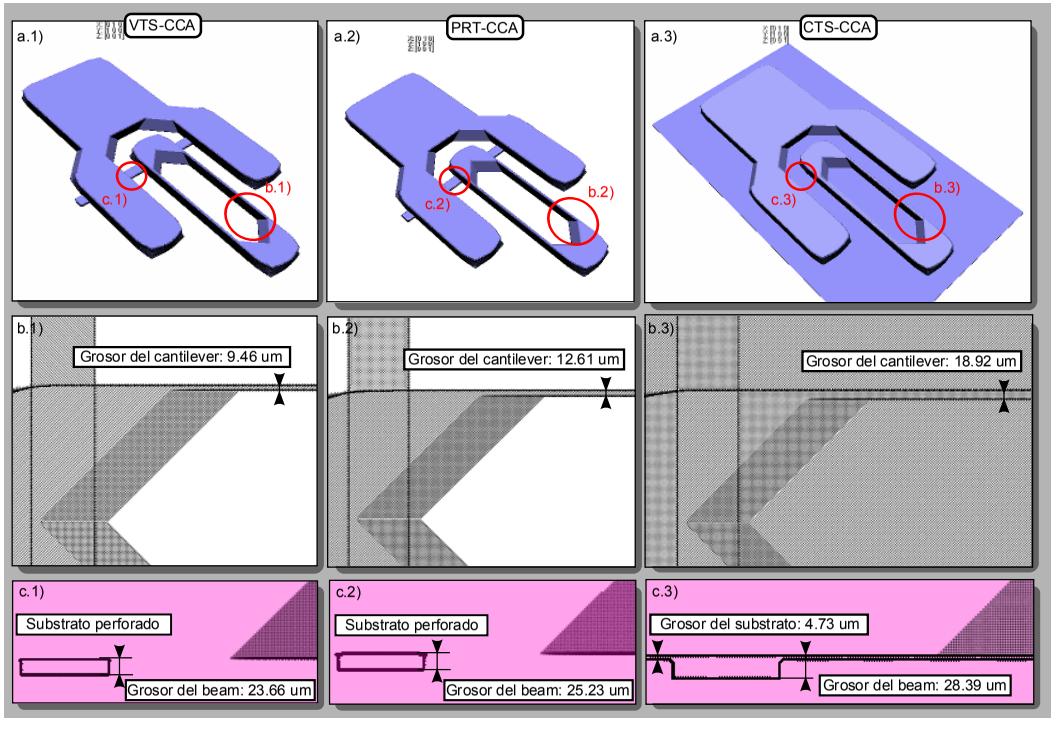

Figura 6.11: Resultado de la simulación de la microfabricación de la microsonda de dos ejes [95] para VTS-CCA, PRT-CCA y CTS-CCA. (a) Vista 3D. (b) Vista detallada del *cantilever* (planta). (c) Vista seccionada del *beam* de torsión (perfil).

3. Aplicar una máscara de nitruro de silicio con la forma de la máscara 3 y 1 en la parte superior e inferior del chip de silicio respectivamente (figura 6.10 (d)).

4. Atacar durante otros 250 minutos y eliminar las máscaras (figura 6.10 (e)).

El tiempo de atacado en el paso 4 debe ser escogido cuidadosamente para que el atacado a la superficie superior e inferior resulte en la perforación del substrato, liberando el *beam* de torsión y los *cantilevers*. El atacado a partir de este punto es poco deseable debido a que resulta en una eliminación total del *cantilever* y los *beams*. Teniendo esto en cuenta, el usuario debe ajustar las dimensiones del orificio alargado interno de la máscara 3 con el objetivo de controlar correctamente el grosor final del *cantilever*.

La figura 6.11 muestra los resultados usando VTS-CCA, PRT-CCA y CTS-CCA. Para CTS-CCA, el substrato de silicio no ha sido perforado (un grosor remanente de $4.73um$ está aún presente). Asimismo, la anchura del *cantilever* es substancialmente mayor que en la simulación realizada para VTS-CCA. Para poder obtener resultados similares con CTS-CCA, es necesario atacar la superficie 10 minutos más. Si este tiempo extra es aplicado en el método VTS-CCA, el *cantilever* queda totalmente atacado, dejando la estructura inutilizable. Los resultados obtenidos mediante el método PRT-CCA son mucho más cercanos a VTS-CCA, con pequeñas diferencias en el *beam* de torsión y el *cantilever* debido a los errores de precisión explicados en la sección 6.4.4.

En conclusión, el uso de CTS-CCA para la simulación del grabado húmedo en escenarios reales pueden dar lugar a errores moderados en los parámetros de fabricación para las dimensiones de la máscara y/o los tiempos de atacado. La implementación propuesta (PRT-CCA) produce resultados más precisos mientras mantiene una complejidad computacional reducida en comparación con VTS-CCA.

6.6. Conclusiones

El estudio realizado en este capítulo se centra en la importancia de la implementación de un modelo lo bastante preciso para las simulaciones del atacado anisótropo húmedo basado en el ACC. A pesar de que la implementación tradicional basada en pasos de tiempo constantes (CTS) permite simulaciones mucho mas rápidas que la implementación exacta basada en pasos de tiempo variables (VTS), existen diferencias relevantes en las velocidades de atacado para diversas orientaciones, así como en los tiempos de atacado simulados y la cantidad de átomos de silicio extraído. Para corregir esta situación, el presente estudio propone una implementación eficiente del modelo exacto basado en la introducción de una nueva variable de estado: el tiempo anticipado de extracción (PRT, del inglés *Predicted Removal Time*), que substituye el concepto tradicional de reducir gradualmente la ocupación de los átomos de silicio y el uso de un árbol binario autoequilibrado, el cual permite acceder y gestionar de una forma eficiente una lista ordenada a partir de los distintos PRT de los átomos. De esta forma es fácil acceder siempre al primer elemento, el cual será el próximo en ser extraído del sistema.

El enunciado de este modelo viene acompañado de un análisis detallado del coste computacional del mismo, así como de los dos métodos tradicionalmente usados. El resultado de este análisis demuestra que CTS-CCA, VTS-CCA y PRT-CCA poseen un coste computacional de $\mathcal{O}(\frac{N^{3/2}}{r\Delta t})$, $\mathcal{O}(N^{3/2+1})$ y $\mathcal{O}(N^{3/2}\log N)$, respectivamente. Estos resultados evidencian que PRT-CCA reduce de forma significativa el coste computacional presentado por VTS-CCA, lo que permite la simulación con este nuevo método, del modelo exacto en tiempos razonables sin la introducción de errores (excepto aquellos introducidos por precisión numérica). Comparado con CTS-CCA, PRT-CCA posee la ventaja de no depender de la reactividad del atacante simulado, dando la posibilidad de ser más eficiente en simulaciones de atacantes poco reactivos o con el predominio de planos lentos.

Finalmente, pese a que el capítulo trata de forma intensiva sobre la inclusión de este nuevo método a las implementaciones tradicionales del ACC, cualquier método atomístico que base el avance de un frente a partir

de la reducción de ocupaciones de los átomos que pueblan la estructura puede beneficiarse de este nuevo punto de vista, por lo que creemos firmemente que este método no está restringido al ACC usado en grabado anisótropo húmedo, sino que su campo de aplicación es significativamente más amplio.

Capítulo 7

Conclusiones y líneas futuras

A lo largo de esta tesis se han cubierto los objetivos propuestos, los cuales se pueden agrupar en tres temáticas:

1. Estudio de los ACs y su aplicación para el modelado de sistemas.

2. Estudio de la implementación de ACs sobre recursos hardware paralelos con el fin de acelerar su ejecución.

3. Aplicar el conocimiento adquirido para superar las limitaciones actualmente existentes en los modelos atomísticos relacionados con el proceso de grabado anisótropo húmedo, usado como método de micromecanizado del silicio.

Estos objetivos están orientados a aumentar la utilidad de los modelos atomísticos en general y del ACC en particular donde, debido a sus altos requisitos computacionales, el poseer un simulador eficiente y preciso es extremadamente útil a la hora de aplicarlo a casos de diseño reales.

El capítulo 2, inicialmente, muestra un estudio amplio de la filosofía de modelado en ACs, su implementación sobre distintos recursos computacionales, y su aplicación para la simulación del atacado anisótropo húmedo, proceso usado en tareas de fabricación de MEMS.

El capítulo 3 ahonda en solventar la problemática de la implementación de ACs sobre FPGAs. Asimismo, aplica la filosofía de modelado de los ACs sobre un caso práctico de ecualización de ganancias en detectores indirectos usados para aplicaciones PET.

El capítulo 4 cubre dos de los objetivos propuestos. En un primer lugar se presenta una metodología para la simulación eficiente de modelos atomísticos que emulan superficies dinámicas sobre GPUs. A continuación se ha aplicado la metodología definida tomando como objeto el ACC definido por M. A. *Gosalvez et. al.* Tras una implementación satisfactoria, se ha podido llegar a las siguientes conclusiones:

- Las GPUs actuales son capaces de simular de forma eficiente superficies dinámicas, si son lo suficientemente grandes y el patrón de acceso a memoria puede optimizarse.
- La integración de los árboles octales para la simulación de superficies dinámicas sobre GPUs reduce drásticamente la cantidad de memoria ocupada y acelera la velocidad de simulación.
- El simulador implementado posee una velocidad de simulación de alrededor de dos órdenes de magnitud superior a los simuladores actualmente existentes basados en una computación secuencial.

El capítulo 5 aprovecha el simulador obtenido en el anterior capítulo para superar la problemática relacionada con los grandes requisitos computacionales que poseen los algoritmos genéticos. De esta manera, hemos podido aprovechar este nuevo punto de vista para calibrar el ACC para nuevos atacantes. De la implementación propuesta y de los resultados conseguidos hemos podido demostrar que los algoritmos genéticos pueden utilizarse de forma satisfactoria para obtener las velocidades de reducción de ocupación de las distintas configuraciones atomísticas con el fin de calibrar el ACC. Las pruebas obtenidas demuestran la validez de este método de calibración propuesto para una gran cantidad de atacantes distintos: KOH, KOH+IPA, TMAH, atacantes isótropos y muy especialmente TMAH+Triton, un atacante que no había sido posible simular hasta ahora y el cual posee gran importancia debido a sus características de reducido *underetching*.

Por último, el capítulo 6 ahonda en la problemática de la ineficiencia del modelo exacto ACC y los errores intrínsecos que introducen las aproximaciones del modelo actualmente aceptadas. En este capítulo se ha introducido una reformulación basada en cambiar la variable de estado del ACC, la cual representaba la ocupación, por un nuevo término: el *Tiempo Anticipado de Extracción* (PRT, del inglés *Predicted Removal Time*). Gracias a esta modificación y a la utilización de un árbol binario autoequilibrado, el modelo obtiene una mayor eficiencia computacional que la implementación exacta tradicional sin por ello introducir errores. La nueva reformulación se ha probado en un caso de simulación real de una microsonda, donde se demuestra que la reformulación presentada supone una mejora de precisión sobre el modelo aproximado tradicional, siendo más exacta y con resultados muy similares a la implementación exacta tradicional. Pese a que el caso de estudio en este apartado haya sido el ACC, la metodología presentada es válida para cualquier sistema atomístico que base su evolución en la reducción de la ocupación o masa interna de las células.

7.1. Resumen de aportaciones

Como resultado del trabajo realizado en la presente tesis, ha sido posible realizar las siguientes aportaciones científicas:

- Publicación de artículo en el congreso: *International Conference on Advances in Electronics and Micro-electronics ENICS08.*

  Néstor Ferrando Jódar, Joaquín Cerdà Boluda, Rafael Gadea Gironés and Vicente Herrero Bosch, *CSDL & GLIDER: CAD Tools for Hardware Implementation of Cellular Automata* Proceedings of International Conference on Advances in Electronics and Micro-electronics (ENICS 2008) (ISBN 978-0-7695-3370-4) 20-25.

- Publicación de artículo en la revista *Nuclear Instrumentation and Methods A*, listada en SCI(2010) con un índice de impacto de 1.317. En él se describe el proceso de calibración de ganancias para detectores PET basados en cristales contínuos usando ACs.

Néstor Ferrando, V. Herrero, J. Cerdà, C. W. Lerche, R. J. Colom, R. Gadea, J. D. Martinez, J. M. Monzó, F. Mateo, A. Sebastia and J. M. Benlloch, *Cellular automatonnext term-based position sensitive detector equalization*, Nuclear Instrumentation and Methods A, 604, 211-214, 2009.

- Publicación de artículo en la revista *Computer Physics Communications*, listada en SCI(2010) con un índice de impacto 1.958. En él se muestra la metodología propuesta para la simulación de superficies dinámicas sobre GPUs así como los resultados de la aplicación de este método sobre el ACC utilizado para el atacado aniótropo húmedo.

  N. Ferrando, M. A. Gosalvez, J. Cerdá, R Gadea and K. Sato, *Octree-based, GPU implementation of a continuous cellular automaton for the simulation of complex, evolving surfaces* Computer Physics Communications 182(3), 628-640, 2011.

- Contrato de licencia de distribución de software para la venta de *GPUetch* a través de la empresa *IntelliSense* como parte del paquete de simulación de atacado anisótropo húmedo *IntelliEtch*.

- Publicación de artículo en la revista *Journal of Micromechanics and Microengineering*, listada en SCI(2009) con un índice de impacto de 1.997. En él se describe el algoritmo evolutivo presentado en esta tesis para la calibración del ACC, así como los resultados objenidos.

  M. A. Gosalvez, N. Ferrando, Y Xing, Prem Pal, K Sato, J Cerdá and R Gadea, *Simulating anisotropic etching of silicon in any etchant: Evolutionary algorithm for the calibration of the continuous cellular automaton*. Artículo aceptado para su publicación en mayo 2011.

- Publicación de artículo en la revista *Journal of Micromechanics and Microengineering*, anteriormente mencionada. En él mostramos la reformulación del ACC propuesta, así como el análisis del coste computacional de las distintas implementaciones del ACC.

  N. Ferrando, M. A. Gosalvez, J. Cerdá, R Gadea adn K. Sato, *Faster, exact implementation of the continuous cellular automaton for*

anisotropic etching simulations Journal of Micromechanics and Microengineering 21(2), 025021, 2011.

Asimismo, se ha enviado el siguiente artículo a una revista indexada, el cual está pendiente del proceso de arbitraje.

- Envío de artículo a la revista *Sensors and Actuators A*, listada en SCI(2010) con un índice 1.674. En él se explica brevemente la reformulación presentada del ACC, esta reformulación es comparada con el *Fast Marching Method*, un método utilizado para describir el avance de frentes.

  M. A. Gosalvez and N. Ferrando *Reformulation of the Continuous Cellular Automaton for the simulation of propagating fronts.*

7.2. Lineas futuras

El trabajo realizado en la presente tesis, pese a suponer avances significativos en el campo de simulación eficiente de ACs, y en especial del ACC utilizado para modelado el atacado anisótropo húmedo para procesos de microfabricación, dista mucho de ser un trabajo cerrado y finalizado. Diversas líneas de trabajo pueden partir del trabajo presentado en esta tesis:

- Pese a que el PRT-CCA resuelve los problemas de ineficiencia de la implementación exacta, el PRT-CCA propuesto es un método fundamentalmente secuencial. Sería muy interesante proponer un modelo basado en PRT-CCA pero con mejores posibilidades de paralelización con el fin de aprovechar arquitecturas hardware como las GPUs.

- El algoritmo genético utilizado para la calibración de ACCs se muestra como un método para interrelacionar las escalas microscópica y macroscópica del ACC. Una posible tarea futura sería la de utilizar el AG para mejorar la tabla de clasificación de átomos presentada en [127] con el objetivo de mejorar el modelo atomístico.

- Asimismo, en relación con el algoritmo genético, sería interesante depurar el algoritmo con el fin de reducir el tiempo de convergencia y obtener mejores calibraciones.

- Otro proyecto, más ambicioso, es el de adentrarse en el modelado y simulación eficiente del atacado DRIE, el cual goza de mucho éxito actualmente debido a la posibilidad de obtener paredes verticales en el substrato de silicio, algo complicado de conseguir mediante el atacado anisótropo húmedo.

- Por otro lado, otra línea de investigación posible es la de aplicar el modelo atomístico de del ACC a otros materiales, tal como el cuarzo, de gran interés en determinados sectores de la fabricación de MEMS.

- Por último, pese a que la implementación propuesta del ACC implementada sobre la GPU ha sido obtenida tras numerosas iteraciones y evaluaciones de rendimiento, tenemos la creencia de que el algoritmo podría depurarse más, obteniendo velocidades de simulación más altas. Asimismo, las nuevas arquitecturas GPU de Nvidia como Fermi ofrecen nuevas posibilidades de computación las cuales sería interesante investigar su adaptabilidad a la simulación del ACC.

Todas las lineas propuestas suponen un interesante reto las cuales están previstas abordarse durante la etapa postdoctoral del tesitando.

Índice de figuras

Índice de tablas

Bibliografía

[1] B. Chopard and M. Droz, *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, 1998.

[2] C. Appert-Rolland, F. Chevoir, P. Gondret, S. Lassarre, J. P. Lebacque, and M. Schreckenberg, eds., *Traffic and Granular Flow ' 07*. Springer, 1997.

[3] S. M. Ulam, "Random processes and transformations," *Proc. International Congress of Mathematicians*, pp. 264–275, 1952.

[4] J. von Neumann, *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[5] M. Gardner, "The fantastic of john conway's new solitaire game life," *Scientific American*, vol. 223, pp. 120–123, 1970.

[6] M. Gardner, *Wheels, Life, and Other Mathematical Amusements*. W.H.Freeman and Co Ltd, 1983.

[7] T. Toffoli and N. Margolus, *Cellular Automata Machines: a New Environment for Modeling*. Mit Press Series, 1987.

[8] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, pp. 219–253, 1982.

[9] N. Margolus, "Physics-like model of computation," *Physica D*, vol. 10, pp. 81–95, 1984.

[10] S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of Modern Physics*, vol. 55, pp. 601–644, 1983.

[11] J. Hardy, Y. Pomeau, and O. de Pazzis, "Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions," *Physical Review A*, vol. 13, pp. 1949–1960, 1976.

[12] U. Frisch, B. Hasslacher, and Y. Pomeau, "Lattice-gas automata for the navier-stokes equation," *Physical Review Letters*, vol. 56, pp. 1505–1508, 1986.

[13] G. G. McNamara and G. Zanetti, "Use of the boltzmann equation to simulate lattice-gas automata," *Physical Review Letters*, vol. 61, pp. 2332–2335, 1988.

[14] S. Wolfram, "Random sequence generation by cellular automata," *Advances in Applied Mathematics*, vol. 7, pp. 123–169, 1986.

[15] M. Tomassini and M. Perrenoud, "On the generation of high-quality random numbers by two-dimensional cellular automata," *IEEE Transactions on Computers*, vol. 49, no. 10, pp. 1146–1151, 2000.

[16] M. Tomassini and M. Perrenoud, "Cryptography with cellular automata," *Applied Soft Computing*, vol. 1, pp. 151–160, 2001.

[17] F. Seredynski, P. Bouvry, and A. Y. Zomaya, "Cellular automata computations and secret key cryptography," *Parallel Computing*, vol. 30, pp. 753–766, 2004.

[18] S. Wolfram, "Cryptography with cellular automata," *Lecture Notes in Computer Science*, vol. 218, pp. 429–432, 1986.

[19] S. Nandi, B. K. Kar, and P. P. Chaudhuri, "Theory and applications of cellular automata in cryptography," *IEEE Transactions on Computer*, vol. 43, no. 12, pp. 1346–1357, 1994.

[20] A. Kirchner and A. Schadschneider, "Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics," *Physica A*, vol. 312, pp. 260–276, 2002.

[21] C. Burstedde, K. Klauck, A. Schadschneider, and J. Zittartz, "Simulation of pedestrian dynamics using a two-dimensional cellular automaton," *Physica A*, vol. 295, pp. 507–525, 2001.

[22] R. White and G. Engelen, "Cellular automata and fractal urban form: a cellular modelling approach to the evolution of urban land-use patterns," *Environment and Planning A*, vol. 25, no. 8, pp. 1175–1199, 1993.

[23] Y. Wei, S. Ying, Y. Fan, and B. Wang, "The cellular automaton model of investment behavior inthe stock market," *Physica A*, vol. 325, pp. 507–516, 2003.

[24] K. Nagel and M. Schreckenberg, "A cellular automaton model for freeway traffic," *Journal de Physique I*, vol. 2, pp. 2221–2229, 1992.

[25] A. R. Kansal, S. Torquato, G. R. Harsh, E. A. Chiocca, and T. S. Deisboeck, "Simulated brain tumor growth dynamics using a three-dimensional cellular automaton," *Journal of Theoretical Biology*, vol. 203, pp. 367–382, 2000.

[26] T. Alarcón, H. M. Byrne, and P. K. Maini, "A cellular automaton model for tumour growth in inhomogeneous environment," *Journal of Theoretical Biology*, vol. 225, pp. 257–274, 2003.

[27] D. G. Mallet and L. G. D. Pillis, "A cellular automata model of tumor immune system interactions," *Journal of Theoretical Biology*, vol. 239, pp. 334–350, 2006.

[28] G. C. Sirakoulis, I. Karafyllidis, and A. Thanailakis, "A cellular automaton model for the effects of population movement and vaccination on epidemic propagation," *Ecological Modelling*, vol. 133, pp. 209–223, 2000.

[29] D. W. Corne and P. Frisco, "Dynamics of hiv infection studied with cellular automata and conformon-p systems," *BioSystems*, vol. 91, pp. 531–544, 2008.

[30] X. Xiao, S. Shao, and K. Chou, "A probability cellular automaton model for hepatitis b viral infections," *Biochemical and Biophysical Research Communications*, vol. 342, pp. 605–610, 2006.

[31] P. Hogewega, "Cellular automata as a paradigm for ecological modeling," *Applied Mathematics and Computation*, vol. 27, no. 1, pp. 81–100, 1988.

[32] H. Baltzer, P. W. Braun, and W. Köhler, "Cellular automata models for vegetation dynamics," *Ecological Modelling*, vol. 107, pp. 113–125, 1998.

[33] I. Karafyllidis and A. Thanailakis, "A model for predicting forest fire spreading using cellular automata," *Ecological Modelling*, vol. 99, pp. 87–97, 1997.

[34] M. Gerhardt and H. Schuster, "A cellular automaton describing the formation of spatially ordered structures in chemical systems," *Physica D: Nonlinear Phenomena*, vol. 36, no. 3, pp. 209–221, 1988.

[35] B. Chopard and P. Luthi, "Reaction-diffusion cellular automata model for the formation of liesegang patterns," *Physical Review Letters*, vol. 72, no. 9, pp. 1384–1388, 1994.

[36] M. Rappaz and C. A. Gandin, "Probabilistic modelling of microstructure formation in solidification processes," *Acta Metallurgica et Materialia*, vol. 41, no. 2, pp. 345–360, 1993.

[37] H. W. Hesselbartha and I. R. Göbela, "Simulation of recrystallization by cellular automata," *Acta Metallurgica et Materialia*, vol. 39, no. 9, pp. 2135–2143, 1991.

[38] O. Thana and S. Büttgenbacha, "Simulation of anisotropic chemical etching of crystalline silicon using a cellular automata model," *Sensors and Actuators A: Physical*, vol. 45, no. 1, pp. 85–89, 1994.

[39] T. Toffoli, "Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics," *Physica D*, vol. 10, pp. 117–127, 1984.

[40] T. Toffoli, "Cam a high-performance cellular-automaton machine," *Physica D*, vol. 10, pp. 195–204, 1984.

[41] T. Toffoli, "Ejemplos de aplicaciones para cam-8." `http://www.ai.mit.edu/projects/im/broch/`, Nov. 2010.

[42] P. Thomas, "Hardware compilation of cellular automata algorithms," 1992.

[43] C. Hochberger, R. Hoffmann, K. P. Völkmann, and J. Steuerwald, "The cepra-1x cellular processor," *Reconfigurable Architectures, High Performance by Configware*, 1996.

[44] J. Cerdá, *Arquitecturas VLSI de autómatas celulares para modelado físico*. PhD thesis, Universidad Politécnica de Valencia, 2004.

[45] M. Cannaro, S. D. Gregorio, R. Rongo, W. Spartado, G. Spezzano, and D. Talia, "A parallel cellular automata environment on multicomputers for computational science," *Parallel Computing*, vol. 21, pp. 803–823, 1995.

[46] G. Sprezzano, O. Spezzano, and D. Talia, "Carpet: A programming language for parallel cellular processing," *Proc. 2nd European School on Parallel Programming Environments*, pp. 71–74, 1996.

[47] D. Talia, "Cellular processing tools for high-performance simulation," *Computer*, vol. 33, no. 9, pp. 44–52, 1993.

[48] M. Resnik, "Starlogo: an environment for decentralized modeling and decentralized thinking," *Conference companion on Human factors in Computing*, pp. 11–12, 1996.

[49] G. A. Wainer and N. Giambiasi, "Application of the cell-devs paradigm for cell spaces modelling and simulation," *SIMULATION*, vol. 76, pp. 22–39, 2001.

[50] B. P. Zeigler and S. Vahie, "Devs formalism and methodology: unity of conception/diversity of application," *Proceedings of the 25th conference on Winter simulation*, pp. 573–579, 1993.

[51] A. Schoneveld and J. F. de Ronde, "P-cam: a framework for parallel complex systems simulations," *Future Generation Computer Systems*, vol. 16, no. 2, pp. 217–234, 1999.

[52] L. Carotenuto and F. Mele, "Pecans: A parallel environment for cellular automata processing," *Future Generation Computer Systems*, vol. 10, pp. 23–41, 1996.

[53] A. Casas, "Intel core i7-980x extreme edition a 3,33 ghz: primera cpu con 6 núcleos," *PC world profesional*, vol. 274, p. 56, 2010.

[54] A. Klimovitski, "Using sse and sse2: Misconceptions and reality," *Intel Developer UPDATE Magazine*, pp. 1–8, 2001.

[55] T. Chen, R. Raghavan, J. N. Dale, and E. Iwata, "Cell broadband engine architecture and its first implementation. a performance view," *IBM Journal of Research and Development*, vol. 51, no. 5, pp. 559–572, 2007.

[56] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics*, vol. 26, no. 1, pp. 80–113, 2007.

[57] W. Whu, T. G. Mattson, and K. Keutzer, "The concurrency challenge," *Design and Test of Computers, IEEE*, vol. 25, no. 4, pp. 312–320, 2008.

[58] Nvidia-Corp., "Especificaciones técnicas de nvidia gtx460." `http://www.nvidia.com/object/product-geforce-gtx-460-us.html`, Nov. 2010.

[59] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors*. Morgan Kaufmann, 2009.

[60] M. Macedonia, "The gpu enters computing's mainstream," *Computer*, vol. 36, no. 10, pp. 106–108, 2003.

[61] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for gpus: stream computing on graphics hardware," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 777–786, 2004.

[62] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Performance evaluation of gpus using the development platform," *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006.

[63] Nvidia-Corp., "Página oficial de nvidia cuda." `http://www.nvidia.es/object/what_is_cuda_new_es.html`, Nov. 2010.

[64] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computer Science and Engineering*, vol. 12, pp. 66–72, 2010.

[65] C. Boyd, "Directcompute: Capturing the teraflop," *Microsoft Personal Developers Conf.*, 2009.

[66] AMD-Corp., "Página oficial de amd stream sdk." `http://developer.amd.com/gpu/atistreamsdk/pages/default.aspx`, Nov. 2010.

[67] Nvidia-Corp., "Fermi architecture white paper." `http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf`, Nov. 2010.

[68] Nvidia-Corp., "Nvidia gtx200 series technical brief." `http://www.nvidia.com/docs/IO/55506/GeForce_GTX_200_GPU_Technical_Brief.pdf`, Nov. 2010.

[69] Nvidia-Corp., "Nvidia cuda c programming guide 3.0." `http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf`, Nov. 2010.

[70] H. Wong, M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying gpu microarchitecture through microbenchmarking,"

IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 235–246, 2010.

[71] M. Silverstein, A. Schuster, D. Geiger, A. Patney, and J. D. Owens, "Efficient computation of sum-products on gpus through software-managed cache," *Proceedings of the 22nd annual international conference on Supercomputing*, 2008.

[72] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W. W. Whu, "Optimization principles and application performance evaluation of a multithreaded gpu using cuda," *Proceedings of the 13rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2008.

[73] J. E. Crates, A. E. Lefohn, and R. T. Whitaker, "Gist: an interactive, gpu-based level set segmentation tool for 3d medical images," *Medical Image Analysis*, vol. 8, pp. 217–231, 2004.

[74] D. Komatitsch, D. Michéa, and G. Erlebacher, "Porting a high-order finite-element earthquake modeling application to nvidia graphics cards using cuda," *J. Parallel Distrib. Comput.*, vol. 69, pp. 451–460, 2009.

[75] T. Preis, P. Virnau, W. Paul, and J. J. Schneider, "Gpu accelerated monte carlo simulation of the 2d and 3d ising model," *Journal of Computational Physics*, vol. 228, pp. 4468–4477, 2009.

[76] W. Liu, B. Schmidt, G. Voss, and W. Müller-Wittig, "Accelerating molecular dynamics simulations using graphics processing units with cuda," *Computer Physics Communications*, vol. 179, no. 9, pp. 634–641, 2008.

[77] S. Gobron, F. Devillard, and B. Heit, "Retina simulation using cellular automata and gpu programming," *Machine Vision and Applications*, vol. 18, pp. 331–342, 2007.

[78] S. Rybacki and J. Himmelspach, "Experiments with single core, multi core, and gpu-based computation of cellular automata," *First Inter-*

national Conference on Advances in System Simulation, pp. 62–67, 2009.

[79] M. G. B. Johnson, D. P. Playne, and K. A. Hawick, "Experiments with single core, multi core, and gpu-based computation of cellular automata," *The 2010 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2010.

[80] M. G. B. Johnson, D. P. Playne, and K. A. Hawick, "Gpu accelerators for evolvable cellular automata," *COMPUTATIONWORLD '09*, pp. 533–537, 2009.

[81] C. Merveille, "Surface quality of {111} side-walls in koh-etched cavities," *Sensors and Actuators A*, vol. 60, pp. 244–248, 1997.

[82] G. Schröpfer, S. Ballandras, M. de Labachelerie, P. Blind, and Y. Ansel, "Fabrication of a new highly-symmetrical, in-plane accelerometer structure by anisotropic etching of (100) silicon," *Journal of Micromechanics and Microengineering*, vol. 7, pp. 71–78, 1997.

[83] L. Qiu, A. Hein, E. Obermeier, and A. Schubert, "Micro gas-flow sensor with integrated heat sink and flow guide," *Sensors and Actuators A*, vol. 54, pp. 547–551, 1996.

[84] L. S. Fan, T. Hirano, J. Hong, P. R. Webb, W. H. Juan, W. Y. Lee, S. Chan, T. Semba, W. Imaino, T. S. Pan, S. Pattanaik, F. C. Lee, I. McFadyen, S. Arya, and R. Wood, "Electrostatic microactuator and design considerations for hdd applications," *IEEE Transactions on Magnetics*, vol. 35, no. 2, pp. 1000–1005, 1999.

[85] S. Baik, J. P. Blanchard, and M. L. Corradini, "Development of micro-diesel injector nozzles via mems technology and effects on spray characteristics," *Atomization and Sprays*, vol. 13, no. 5, p. 211, 2003.

[86] J. Bryzek, "Impact of mems technology on society," *Sensors and Actuators A*, vol. 56, pp. 1–9, 1996.

[87] Y. Développement, "Status of the mems industry 2010 report," 2010.

[88] M. F. Aimi, M. P. Rao, N. C. MacDonald, A. S. Zuruzi, and D. P. Bothman, "High-aspect-ratio bulk micromachining of titanium," *Nature Materials*, vol. 3, pp. 103–105, 2004.

[89] L. Liew, W. Zhang, L. An, S. Shah, R. Luo, Y. Liu, T. Cross, M. L. Dunn, V. Bright, J. W. Daily, and R. Raj, "Ceramic mems," *American Ceramic Society Bulletin*, vol. 80, no. 5, pp. 25–30, 2001.

[90] S. Franssila, *Introduction to Microfabrication*. Wiley, 2004.

[91] K. E. Petersen, "Silicon as a mechanical material," *Proceedings of the IEEE*, vol. 70, no. 5, pp. 420–457, 1982.

[92] Analog-Devices-Corp., "Espeficifaciones admp401." `http://www.analog.com/static/imported-files/data_sheets/ADMP401.pdf`, Nov. 2010.

[93] G. Schröpfer, M. de Labachelerie, S. Ballandras, and P. Blind, "Collective wet etching of a 3d monolithic silicon seismic mass system," *Journal of Micromechanics and Microengineering*, vol. 8, pp. 77–79, 1998.

[94] Sandia-Corp., "Cortesía de sandia national laboratories, summit(tm) technologies, www.mems.sandia.gov," Nov. 2010.

[95] K. Fukuzawa, S. Terada, M. Shikida, H. Amakawa, H. Zhang, and Y. Mitsuya, "Mechanical design and force calibration of dual-axis micromechanical probe for friction force microscopy," *Journal of Applied Physics*, vol. 101, p. 034308, 2007.

[96] L. E. Katz, *VLSI Technology (Chapter 4)*. McGraw-Hill, 1988.

[97] K. R. Williams and R. S. Muller, "Etch rates for micromachining processing," *Journal of Microelectromechanical Systems*, vol. 5, no. 4, pp. 256–269, 1996.

[98] J. K. Bhardwaj and H. Ashraf, "Advanced silicon etching using high density plasmas," *Micromachining and Microfabrication Process Technology*, vol. 2639, pp. 224–233, 1995.

[99] D. B. Lee, "Anisotropic etching of silicon," *Journal of Applied Physics*, vol. 40, no. 11, pp. 4569–4574, 1969.

[100] E. S. Ammar and T. J. Rodgers, "Umos transistors on (100) silicon," *IEEE Transactions on Electron Devices*, vol. 27, no. 5, pp. 907–913, 1980.

[101] V. Lindroos, M. Tilli, A. Lehto, and T. Motooka, *Handbook of Silicon Based MEMS Materials and Technologies, (Chapter 24)*. Elsevier, 2010.

[102] P. Allongue, V. Costa-Kieling, and H. Gerischer, "Etching of silicon in naoh solutions," *IEEE Transactions on Electron Devices*, vol. 140, no. 4, pp. 1018–1026, 1993.

[103] P. Allongue, V. Costa-Kieling, and H. Gerischer, "Etching mechanism and atomic structure of h-si(111) surfaces prepared in nh4f," *Electrochimica Acta*, vol. 40, no. 10, pp. 1353–1360, 1995.

[104] K. Sato, M. Shikida, Y. Matsushima, T. Yamashiro, K. Asaumi, Y. Iriye, and M. Yamamoto, "Characterization of orientation-dependent etching properties of single-crystal silicon: effects of koh concentration," *Sensors and Actuators A*, vol. 64, pp. 87–93, 1998.

[105] K. Sato, M. Shikida, Y. Matsushima, T. Yamashiro, K. Asaumi, Y. Iriye, and M. Yamamoto, "Anisotropic etching rates of single-crystal silicon for tmah water solution as a function of crystallographic orientation," *Sensors and Actuators A*, vol. 73, pp. 131–137, 1999.

[106] R. A. Wind, H. Jones, M. J. Little, and M. A. Hines, "Orientation-resolved chemical kinetics: Using microfabrication to unravel the complicated chemistry of koh/si etching," *Journal of Phyiscal Chemistry*, vol. 106, pp. 1557–1569, 2002.

[107] R. A. Wind, H. Jones, and M. A. Hines, "Macroscopic etch anisotropies and microscopic reaction mechanisms: a micromachined structure for the rapid assay of etchant anisotropy," *Surface Science*, vol. 460, pp. 21–38, 2000.

[108] I. Zubel and M. Kramkowska, "The effect of alcohol additives on etching characteristics in koh solutions," *Sensors and Actuators A*, vol. 101, pp. 255–261, 2002.

[109] I. Zubel and M. Kramkowska, "Etch rates and morphology of silicon (h k l) surfaces etched in koh and koh saturated with isopropanol solutions," *Sensors and Actuators A*, vol. 115, pp. 549–556, 2004.

[110] P. Pal and K. Sato, "Various shapes of silicon freestanding microfluidic channels and microstructures in one-step lithography," *Journal Micromechanincs and Microengineering*, vol. 19, p. 055003, 2004.

[111] P. Pal, M. A. Gosalvez, and K. Sato, "Silicon micromachining based on surfactant-added tetramethyl ammonium hydroxide: Etching mechanism and advanced applications," *Japanese Journal of Applied Physics*, vol. 49, p. 056702, 2010.

[112] M. Shikida, K. Nanbara, T. Koizumi, H. Sasaki, M. Odagaki, K. Sato, M. Ando, S. Furuta, and K. Asaumi, "A model explaining mask-corner undercut phenomena in anisotropic silicon etching: a saddle point in the etching-rate diagram," *Sensors and Actuators A: Physical*, vol. 97, pp. 758–763, 2002.

[113] J. Fruhauf, K. Trautmann, J. Wittig, and D. Zielke, "A simulation tool for orientation dependent etching," *Sensors and Actuators A: Physical*, vol. 3, no. 3, p. 113, 1993.

[114] C. H. Séquin, "A simulation tool for orientation dependent etching," *Sensors and Actuators A: Physical*, vol. 34, no. 3, pp. 225–241, 1992.

[115] G. Li and T. Hubbard, "Segs: On-line www etch simulator," *Proceedings of IEEE MSM'98*, pp. 356–361, 1998.

[116] K. Asaumi, Y. Iriye, and K. Sato, "Anisotropic-etching process simulation system microcad analyzing complete 3d etching profiles of single crystal silicon," *Proceedings of IEEE Int. Conf. MEMS*, pp. 412–417, 1997.

[117] H. Camon, A. M. Gue, J. S. Daniel, and M. Djafari-Rouhani, "Modelling of anisotropic etching in silicon-based sensor application," *Sensors and Actuators A: Physical*, vol. 33, no. 1, pp. 103–105, 1992.

[118] T. Hubbard and E. K. Antonsson, "Cellular automata modeling in mems design," *Sensors and Materials*, vol. 9, 1997.

[119] S. Buttgenbach and O. Than, "Suzana: a 3d cad tool for anisotropically etched silicon microstructures," *ED&TC 96. Proceedings*, vol. 9, pp. 454–458, 1996.

[120] M. A. Gosalvez, R. M. Nieminen, P. Kilpinen, E. Haimi, and V. Lindroos, "Anisotropic wet chemical etching of crystalline silicon: atomistic monte-carlo simulations and experiments," *Applied Surface Science*, vol. 178, no. 1-4, pp. 7–26, 2001.

[121] Z. Zhu and C. Liu, "Micromachining process simulation using a continuous cellular automata method," *Journal of Micromechanical Systems*, vol. 9, no. 2, pp. 252–261, 2000.

[122] M. A. Gosalvez and Y. Xing, "Página web visualtapas." `http://www.fyslab.hut.fi/~mag/VisualTAPAS/Home.html`, Nov. 2010.

[123] Z. Zhou, Q. Huang, W. Li, and W. Deng, "A cellular automaton-based simulator for silicon anisotropic etching processes considering high index planes," *Journal of Micromechanics and Microengineering*, vol. 17, pp. S38–S49, 2007.

[124] J. A. Sethian, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science.* Cambridge University Press, 1999.

[125] B. Radjenovic, M. Radmilovic-Radjenovic, and M. Mitric, "Nonconvex hamiltonians in three dimensional level set simulations of the wet etching of silicon," *Applied Physics Letters*, vol. 89, p. 213102, 2006.

[126] M. A. Gosalvez, K. Sato, A. S. Foster, R. M. Nieminen, and H. Tanaka, "An atomistic introduction to anisotropic etching," *Journal of Micromechanics and Microengineering*, vol. 17, pp. S1–S26, 2007.

[127] M. A. Gosalvez, Y. Xing, and K. Sato, "Analytical solution of the continuous cellular automaton for anisotropic etching," *Journal of Micromechanical Systems*, vol. 17, no. 2, pp. 410–431, 2008.

[128] M. A. Gosalvez, Y. Xing, K. Sato, and R. M. Nieminen, "Discrete and continuous cellular automata for the simulation of propagating surfaces," *Sensors and Actuators A: Physical*, vol. 155, no. 1, pp. 98–112, 2009.

[129] G. C. Sirakoulis, I. Karafyllidis, and A. Thanailakis, "A cad system for the construction and vlsi implementation of cellular automata algorithms using vhdl," *Microprocessors and Microsystems*, vol. 27, pp. 381–396, 2003.

[130] Altera-Corp., "Nios ii processor: The world's most versatile embedded processor." `http://www.altera.com/products/ip/processors/nios2/ni2-index.html`, Ene. 2011.

[131] N. Ferrando, "Desarrollo de Herramientas para el Diseño e Implementación Hardware de Estructuras Celulares Paralelas," Master's thesis, E.T.S.I.T. Universidad Politécnica de Valencia, Camino de Vera S/N CP 46022 Valencia, Spain, 2008.

[132] Grupo-DSD, "Página web del grupo de diseño de sistemas digitales." `http://www.upv.es/dsd/`, Ene. 2011.

[133] Xilinx-Corp., "Memory interfaces for xilinx fpgas." `http://www.xilinx.com/products/design_resources/mem_corner/resource/xaw_dram_ddr.htm`, Ene. 2011.

[134] Altera-Corp., "External memory interface handbook." `http://www.altera.com/literature/hb/external-memory/emi.pdf`, Ene. 2011.

[135] D. Griffiths, *Introduction to Elementary Particles, 2nd ed.* Wiley-VCH, 2008.

[136] C. L. Melcher, "Scyntillation crystals for pet," *Journal Nuclear Medecine*, vol. 41, pp. 1051–1055, 2000.

[137] C. L. Melcher, "Initial characterization of a nonpixelated scintillator detector in a pet prototype demonstrator," *IEEE Transactions on Nuclear Science*, vol. 53, no. 5, pp. 2543–2548, 2006.

[138] Hamamatsu-Corp., "Hamamatsu h800 datasheet." `https://jp.hamamatsu.com/resources/products/etd/pdf/H8500C_H8500D_TPMH1308E01.pdf`, Nov. 2010.

[139] Hamamatsu-Corp., "The photomultiplier handbook." `http://sales.hamamatsu.com/assets/applications/ETD/pmt_handbook_complete.pdf(Chapter4.3.2)`, Nov. 2010.

[140] Hamamatsu-Corp., "The photomultiplier handbook." `http://sales.hamamatsu.com/assets/applications/ETD/pmt_handbook_complete.pdf(Chapter9.1.1)`, Nov. 2010.

[141] H. O. Anger, "Scintillation camera," *The review of Scientific Instruments*, vol. 29, no. 1, pp. 27–33, 1958.

[142] V. Herrero, R. Gadea, R. Colom, A. Sebastiá, J. M. Monzó, R. Esteve, C. W. Lerche, and J. M. Benlloch, "Pesic: An integrated front-end for pet applications," *15th IEEE-NPSS Real-Time Conference*, pp. 1–6, 2007.

[143] S. Siegel, R. W. Silverman, W. Shao, and S. R. Cherry, "Simple charge division readouts for imaging scintillator arrays using a multi-channel pmt," *IEEE Transactions on Nuclear Science*, vol. 43, no. 3, pp. 1634–1641, 1996.

[144] Y. Wang, W. Wong, M. Aykac, J. Uribe, H. Li, H. Baghaei, Y. Liu, and T. Xing, "An iterative energy-centroid method for recalibration of pmt gain in pet or gamma camera," *IEEE Transactions on Nuclear Science*, vol. 49, no. 5, pp. 2047–2050, 2002.

[145] H. Li, Y. Liu, T. Xing, Y. Wang, J. Uribe, H. Baghaei, S. Xie, S. Kim, R. Ramirez, and W. Wong, "An instantaneous photomultiplier gain calibration method for pet or gamma camera detectors using an led network," *IEEE NSS Conference records*, pp. 2447–2451, 2004.

[146] C. W. Lerche, J. M. Benlloch, F. Sánchez, N. Pvón, B. Escat, E. N. Gimenez, M. Fernández, I. Torres, M. Giménez, and J. Martínez, "Depth of gamma-ray interaction within continuous crystals from the width of its scintillation light-distribution," *IEEE Transactions on Nuclear Science*, vol. 52, no. 3, pp. 560–572, 2005.

[147] Sant-Gobain, "Prelude 420 datasheet." `http://www.detectors.saint-gobain.com/uploadedFiles/SGdetectors/Documents/Product_Data_Sheets/PreLude420-Data-Sheet.pdf`, Nov. 2010.

[148] C. W. Lerche, A. Ros, R. Esteve, J. M. Monzó, A. Sebastiá, F. Sánchez, A. Munar, and J. M. Benlloch, "Dependency of energy-, position-, and depth of interaction resolution on scintillation crystal coating and geometry," *IEEE Transactions on Nuclear Science*, vol. 55, no. 3, pp. 1344–1351, 2008.

[149] Y. Xing, M. A. Gosalvez, and K. Sato, "Step flow-based cellular automaton for the simulation of anisotropic etching of complex mems structures," *New Journal of Physics*, vol. 9, p. 436, 2007.

[150] Z. Zhou, Q. Huang, W. Li, and C. Zhu, "A 3-d simulator for silicon anisotropic wet chemical etching process based on cellular automata," *Journal of Physics: Conference Series*, vol. 34, pp. 674–679, 2006.

[151] IntelliSense-Corp., "Página web intellietch." `http://wwwhttp://www.intellisensesoftware.com/modules/IntelliEtch.html`, Dic. 2010.

[152] M. A. Weiss, *Estructuras de Datos en Java (Capítulo 17)*. Addison-Wesley, 2000.

[153] D. Libes, "Modeling dynamic surfaces with octrees," *Computers & Graphics Magazine*, vol. 15, no. 3, pp. 383–387, 1991.

[154] M. A. Gosalvez, Y. Xing, K. Sato, and R. M. Nieminen, "Atomistic methods for the simulation of evolving surfaces," *Journal of Micromechanics and Microengineering*, vol. 18, p. 055029, 2008.

[155] G. für Mikroelektronikanwendung Chemnitz mbH, "Simode, collection of examples," 2001.

[156] B. Tang, K. Sato, H. Tanaka, and M. A. Gosalvez, "Fabrication of shard tips with high aspect ratio by surfactant-modified wet etching for the afm probe," *Paper accepted for IEEE MEMS 2011 conference*, 2011.

[157] M. Roberts, J. Packer, M. Sousa, and J. Mitchell, "A work-efficient gpu algorithm for level set segmementation," *Proceedings of the ACM SIGGRAPH/Eurographics Conference on High Performance Graphics*, pp. 123–132, 2010.

[158] W. Jeong, J. Beyer, M. Hadwiger, A. Vazquez, H. Pfister, and R. T. Whitaker, "Scalable and interactive segmentation and visualization of neural processes in em datasets," *IEEE Transactions on Visualization and Computer Graphics*, pp. 123–132, 2010.

[159] M. A. Gosalvez, R. M. Nieminen, P. Kilpinen, E. Haimi, and V. Lindroos, "Anisotropic wet chemical etching of crystalline silicon: atomistic monte-carlo simulations and experiments," *Applied Surface Science*, vol. 178, pp. 7–26, 2001.

[160] M. A. Gosalvez, P. Pal, J. Ohara, N. Ferrando, H. Hida, and K. Sato, "Acquisition of experimental etch rates for the complete unit sphere based on drie micromachined wagon wheels." Artículo enviado a Journal of Micromechanics and Microengineering en Febrero de 2011.

[161] J. H. Holland, *Adaptation in natural and artificial systems.* The MIT Press, 1992.

[162] A. E. Baumal, J. J. McPhee, and P. H. Calamai, "Application of genetic algorithms to the design optimization of an active vehicle suspension system," *Computer Methods in Applied Mechanics and Engineering*, vol. 163, pp. 87–94, 1998.

[163] G. W. Greenwood and A. M. Tyrrell, *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems.* Wiley-IEEE Press, 2006.

[164] H. Braun, "On solving travelling salesman problems by genetic algorithms," *Lecture Notes in Computer Science*, vol. 496, pp. 129–133, 1991.

[165] N. H. Packard, *Adaptation towards the Edge of Chaos.* Dynamic Patterns in Complex Systems, 1988.

[166] F. C. Richards, T. P. Meyer, and N. H. Packard, "Extracting cellular automaton rules directly from experimental data," *Physica D: Nonlinear Phenomena*, vol. 45, no. 1, pp. 189–202, 1990.

[167] M. Mitchell, P. T. Hraber, and J. P. Crutchfield, "Revisiting the egde of chaos: Evolving cellular automata to perform computations," *Complex Systems*, vol. 7, pp. 89–130, 1993.

[168] G. J. E. Rawlins, *Foundations of Genetic Algorithms Vol 1.* Morgan Kaufmann Publishers, 1991.

[169] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," *Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pp. 14–21, 1987.

[170] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm: I. continuous parameter optimization," *Journal Evolutionary Computation*, pp. 25–49, 1993.

[171] H. Pohlheim, "Geatbx: Introduction evolutionary algorithms: Overview, methods and operators." `http://www.geatbx.com/index.html`, 2006.

[172] M. Shikida, M. Aido, Y. Ishihara, T. Ando, K. Sato, and K. Asaumi, "Non-photolithographic pattern transfer for fabricating pen-shaped microneedle structures," *Journal of Micromechanics and Microengineering*, vol. 14, no. 11, pp. 1462–1467, 2004.

[173] C. Liu, *Foundations of MEMS (Fig. 10-3-1)*. Pearson Prentice Hall, 2005.

[174] "Experimentos realizados en el departamento de micro-nano systems engineering, universidad de nagoya. imágenes reproducidas con permiso de m. a. gosalvez.."

[175] D. knuth, *The art of computer programming vol 3. Sorting and Searching*. Addison-Wesley, 1997.

[176] G. M. Adelson-Velskii and E. M. Landis, "An algorithm for the organization of information," *Soviet Math*, vol. 3, p. 1259–1263, 1962.

[177] R. Bayer, "Symmetric binary b-trees: Data structure and maintenance algorithms," *Acta Informatica*, vol. 1, pp. 290–306, 1972.

[178] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach (Memory Hierarchy Design)*. Morgan Kaufmann, 2007.