UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

PINT, HERRAMIENTA DE SIMULACIÓN BASADA EN TRAZAS PIN

Final Year Project

Computer engineering

Author: Francisco Blas Izquierdo Riera

Director: Julio Sahuquillo Borrás - UPV

Per Stenström - CTH

December 18, 2012

Abstract

In the course of this project we have developed a set of programs to improve the correction and execution time of the gem5 simulator.

For this, we moved the functional simulation step out of gem5 into an independent instrumented process to ensure correction in the functional stage and to provide a good execution speed (since the code will then be natively executed). This instrumentation is done by Pin.

Also, in order to allow efficient communication between the processes despite the limitations imposed by Pin to the available tools, an IPC framework to allow message passing between the processes was developed. This framework uses lockless fifo queues over shared memory so the resulting slowdown is minimal.

Keywords: hardware, simulator, x86, Pin, Pintool, gem5, ipc, fifo, C++

Contents

Chapter 1

Introduction

1.1 Project rationale

Despite the vast amount of hardware simulators that exist nowadays, most of them either lack flexibility on the simulations or are slow since they simulate the code execution instead of instrumenting the natively executed executable. As a related issue, since code is simulated and not executed it is common to find bugs where the simulator will not set the processor state properly which cause corner cases where not acting as the processor causes execution issues with some programs.

Also many simulators lack support for parallel execution and those who do tend to add big overheads when running the simulation in a single machine and will not support instruction level simulation granularity.

Finally current simulators tend to add big overheads to the functional simulation step which makes it unfeasible to run large tests even when simulating simple systems.

Program instrumentation solves all these shortcomings by running the code natively (modified so it will also execute the instrumentation code),

allowing it to run in parallel and, since code is executed natively in the processor, providing a completely native execution.

Given the limitations of the current simulators we consider that the community needs a flexible and fast instrumentation based tracer able to be used with a broad range of programming languages who will handle local simulations in parallel, with small overheads and with instruction level granularity.

1.2 Project objectives

Our main objective is providing an instrumentation based tracer that can be used with other simulators. Given the problems with the size these traces can have we will feed them in a lively fashion.

In order to see whether these objectives are met or not we will measure the slowdown compared to the non instrumented program with a simple trace consumer (to ensure it is not the bottleneck). Our objective is getting at least similar slowdowns to the ones of Graphite [9], but removing the caveats it has at least on single core processors.

Also we intend to design an architecture which can later be expanded to support multiple simultaneous execution threads. This will be done on a later version of the project though due to timing constraints.

1.3 Project strengths

The biggest problem with instrumentation-based systems is that the instrumentation code is limited heavily by the instrumentation API of the instrumentation system (for example the POSIX thread API can not be used with Pin), this also reduces vastly the number of languages that can be used, in

order to overcome these limitations, we use FIFO queues placed in shared memory to extract the data from a process to another, using the operating system process separation to execute the simulation in a different processor. As a result, the simulator overcomes the restrictions caused by the instrumentation framework since these will only apply to the process where the program is being instrumented.

As a side effect of this approach, the resulting simulators will be segmented since the functional simulation can be done on a different processing unit than the one running the simulation itself. As the number of processor cores increases it is likely that hardware simulators will use the segmentation approach more extensively in order to increase performance. Also, when shared the cache accesses caused by the shared memory communication cause slowdowns, hyperthreading processors can be used and proper processor affinity to the processes can be set so the critical simulation parts (i.e those responsible of bottlenecks) will be set along with the previous part on the different threads of a single core so the reads from the FIFO queue are likely to be on the level 1 cache.

To ensure real parallelism each thread of the instrumented program can use a different FIFO queue to extract its traces[1]. This also allows the user to limit instruction granularity by setting an appropriate queue size since the instrumentation will stop execution once the queue is full[2].

As an example of how Pint can be used we also created Gin5 a slightly modified version of the gem5 simulator which uses Pint's instrumentation as the source of the memory access information during the simulation.

Another known problem is that simulators tend to be very good on sim-

[1]The completely multithreaded implementation will be finished with the next release of Pint

[2]For this a simulation started event is added to ensure the first instruction will not be executed until the simulator wants it.

ulating a specific part of the system whilst having issues on others. We consider that in the future this FIFO system may be useful to interconnect simulators so the best of them can be gotten.

1.4 Memory structure

In this introduction we presented the problem we are trying to fix, our objectives and our strengths.

On the next section, we will explain the state of the art at the time of our publication in the topics of Architectural Simulators, Instrumentation systems and Benchmarks.

Afterwards we will analyze the four modules we developed for the project and we will continue later with the design decisions.

We will finally present our benchmarking results and our conclusions.

Annexed you will find a brief user manual in case you want to try our system and the referred bibliography.

On the Annex folder you will find the sources we developed in this project.

Chapter 2

State of the art

Of the many simulators currently available, we have chosen three to explain which is the current state of the art for being the ones on which most work is being done nowadays: gem5, Multi2Sim and Graphite.

Also, as instrumentation tools we will cover gprof based profiling and Pin.

Finally, as benchmarks we will cover the SPEC CPU2006 and the SPLASH-2 benchmarks.

2.1 Architectural simulators

Architectural simulators are tools used to see how a proposed processor design would work without the need of building the processors themselves. Despite these share a some similarities with virtual machines in that they execute programs and that the main focus in both is the correct execution of the program; virtual machines have their main focus in providing a speedy execution of the program, whilst architectural simulators focus on providing good statistics of the program execution and executing the program in the same way the architecture would use.

Architectural simulators tend to be structured in a set of stages, disassembly, functional simulation and cycle by cycle simulation.

During the disassembly stage the machine code to be executed is transformed into a set of structures that can be understood by the simulator, the set of structures used is critical for an efficient simulation.

During the functional simulation the resulting set of structures is interpreted by the simulator to modify the internal state of the processor structures and the representation of the simulated program's memory space.

Finally during the cycle by cycle simulation the represented architecture and system are simulated in a cycle by cycle basis so the timing results are precise.

Simulators may have these stages clearly differentiated or not but all of the do have these stages.

Also some simulators emulating only the memory system (and further processor structures) are based on memory traces. A memory trace is a description of the memory accesses made by a particular program when run which is then replayed on the simulated memory system.

In general trace based simulators tend to be fast since they will not only remove the execution step but also use a more simplified model for the processor. But traces have a few problems: on one side the programs being run need to be run in a way in which they will be generated, for example with a dynamically instrumented program, and when big enough they can take a lot of space, for example a trace of the SPLASH-2 LU with contiguous blocks trace would take around 1.5GiB if each access could be stored in only 32 bits.

Anyway there are some nice works in trace generation with Pin for simulators like Dinero IV [7], an example of which can be found in the dinerotool [1] by Kenneth Barr.

When using traces it is hard to overcome the requirement of using traces, but, it is possible to overcome the space limitation restrictions by feeding them live into our memory simulator. This was the approach chose by us.

2.1.1 Graphite

Graphite [18] [9] is a multicore simulator also written over Pin designed to provide real multithreading both when run locally and when run over a large number of computers. In order to do this, graphite hijacks some syscalls of the syscalls which will then be sent either to the local kernel or to the central kernel or to both. A similar procedure is used to track memory access es and an internal "MMU" tracks which machine has which copy of the memory.

In order to synchronize threads Graphite provides a few different synchronization ways of which the fastest is the lax synchronization method.

Given the popularity of this simulator nowadays Graphite was the simulator chosen as the reference against which we will compare the speed of our system.

Saddly, one of the major caveats with Graphite is that it is very system specific and, as a result, it was impossible for us to run it on our testing equipment.

2.1.2 Multi2Sim

Multi2Sim [20] [3] is a simulator supporting a big set of targets to emulate different architectures, both CPU and GPU.

As a simulator it is split in different components, a disassembler intended to convert the input programs into something the simulator can understand and use, a functional simulator which maintains the CPU and memory state

and runs the code and a cycle by cycle simulator which does the execution. It also provides some visual tools for checking how the simulation is run.

2.1.3 gem5

gem5 [16] [2] is the result of the merge of two powerful simulators: M5 and GEMS. gem5 is a simulator able to emulate some architectures both in Full Mode (this is, running the kernel as part of the simulation) and in Syscall Emulation mode (as the two aforementioned simulators by emulating the kernel for the provided binary).

As a full system simulator it is known for the flexibility it has for emulating different systems, not only by the number of architectures it supports but also by the number of devices it can emulate and the flexibility it provides in doing so.

The main problem it has is that although the modules are written in C++, they are usually run by a python script which complicates the system.

This flexibility gem5 was the reason for choosing this simulator as a target for implementing our system.

2.2 Instrumentation systems

Instrumentation systems provide ways to know how is the code running, either for later statistic generation and performance checking or for other uses like memory trace generation.

Instrumentation can be dynamic if the code that will control how the program is running is added when it is executed or static if this code is interleaved when building the program with the compiler. Normally dynamic instrumentation is preferable since it will allow us to instrument also propri-

etary programs and will not require a modified compiler.

2.2.1 gprof

gprof [17] [6] is a profiling system used along with programs compiled with special flags by gcc [14] [13]. For this gcc will embed the profiling code and mix it with the compiled sources before assembly. This technique is called static instrumentation since it is done in compilation time.

Programs compiled with profiling flags will generate when run binary file, called gmon.out, containing the execution statistics. Afterwards a call to gprof can be used to interpret the generated file.

Although traces could be also generated by using these techniques the requirement of having to compile the programs with a particular compiler is an impediment in some cases thus the ideas provided by this system where discarded.

2.2.2 Pin

Pin [21] [10] [8] on the other side is a dynamic instrumentation framework, this means that instrumentation code is added dynamically. For this Pin hijacks with ptrace the program to be run, as a debugger like gdb [15] [12] would do, and then loads the Pintools' code and the Pin framework into the running program and modifies the process so it will run the code produced by the JIT generator provided by PIN.

For this to work, Pin provides a modified version of the C++ runtime which has some features stripped down in order to prevent incompatibilities with the program being run. Anyway most of the C++ features can still be used by the tools and for those that can not Pin provides alternatives (for example locks).

The main problem with Pin is that running it on hardened systems is complicated since the default method used by Pin to attach to the program via ptrace is considered dangerous by these kernels (since it is not a parent attaching to its child but the other way around), also the JIT compiler provided by Pin causes problems because it tries to have mapPings which are both writable and executable which is another technique restricted by hardened systems.

Despite these issues Pin was the system chosen for providing the instrumentation framework.

2.3 Benchmarks

Benchmarks are programs with standardized inputs that are used to measure and compare the performance of different systems running them. Depending of the component being measured different metrics can be used: power consumption, execution time, number of frames per second generated, etc. Of these in this project we care the most about execution time.

Benchmarks can be synthetic when they emulate the load caused by typical programs of a particular type, examples of which are Dhrystone [25] and Whetstone [5]; or application when they run one or more real world programs like the two we have analyzed. In general real world benchmarks provide more meaningful results since they allow you to see how will real applications behave.

2.3.1 SPEC CPU2006

The SPEC CPU2006 [22] [11] benchmark is a set of programs from the real world which are provided along with some inputs to test the speed of a

system and with a main focus on the CPU execution speed. Despite being there since the 2006 these benchmarks are widely used and understood in the academic and real world.

Most of the programs provided with the benchmark are licensed with GPL style licenses and are well known in the free software world, for example gcc or perl, whilst others come from different research projects. It is because of this that the copyright is held over the input files in this benchmarks.

The main problem with these benchmarks is that they focus on single threaded processes.

2.3.2 SPLASH-2

The SPLASH-2 [23] [26] benchmark was developed by the Flash research group at the Stanford university to provide a set of benchmarks that could be used on shared memory multiprocessor systems. Although the benchmarks are quite old and require modifications to work properly they can still be used and have the advantadge of running in a short time.

The applications provided are related to the scientific world with examples of 3 body gravity simulators or some kernels like the LU decomposition of a matrix.

Since the original tests will not run, we used a modified version of the SPLASH-2 benchmark [19]. Even more modifications were required for the null macro to work properly and for the tests to be able to be run with Pin on hardened systems, these modifications are provided as a patch file in the source distribution.

The main reason for choosing these was that the relative performance results of these tests (although with more than one processor) were provided on [9] so we did not need to run the benchmarks again for Graphite and thus

set up the required Debian environment.

Chapter 3

System description

Our application will be divided in 4 modules: Mead, a framework for providing an efficient message passing interface between different processes; Pint, a Pin based trace generator; Schnapps, a simple consumer of the traces; and Gin5, a gem5 trace player for the memory system.

The traces will be generated by Pint and then fed through Mead to either Schnapps or Gin5 which will process it and provide some simulation statistics.

3.1 Mead: a message passing framework

The pattern of message passing is not new and it conforms the base of some Object Oriented views. Mead will provide a fast and simple way of passing around the traces as messages stating that something has happened (for example the program made an execution memory access of size x at position y). These messages may contain the thread identifier of the thread that caused them and also attached data, for example in the case of a memory write the data available before writing and the data being written.

Although the API provided by mead is quite agnostic of the message

passing system being used we have chosen producer-consumer FIFO queues.

FIFOs are used since they are a known pattern which allows for easy implementation and migration over other interprocedural communication systems, if interprocess shared memory is not an option, like POSIX message queues or datagram sockets.

Our FIFO model differs slightly from the standard model since it allows for two communication types, on one hand you have the event communication system which can queue many events for further handling by the receiving side. On the other you will find a command interface able of holding a single command. The command interface requires acknowledging the sent command and is used to indicate important events which require specific handling by the queue system like the death of the FIFO or the beginning and ending of the simulation procedures.

The main difference between events and commands are that events are unidirectional (from the producer to the consumer) whilst commands can be used bidirectionally (as long as the absence of collisions is guaranteed by the programmer) and are more easily handled with the futex syscall which makes them very useful for events which will require a really heavy processing on the other side by allowing the other thread to preempt the CPU while this is done.

The FIFO architecture is based over a central FIFO (called the main FIFO) which is used to send global events which are supposed to stall the simulation until attended (so the simulator can decide whether it should clear or not the per thread queues before processing the aforementioned event), this queue handles at least the thread creation and deletion events where a new FIFO queue is negotiated between both sides, but it can also be used to process events like the creation and deletion of new mappings amongst

others.

Given the importance of the main FIFO in the architecture it is important that both processes know where to access it beforehand and are able to negotiate its creation independently of who arrived first (since synchronization is impossible before the FIFO creation).

The framework also features a per thread FIFO which can be used to send events which are not of global significance to the listener on the other side. This lets the programmer communicate information fast since the queues can be then lockless and, as a result, as long as there are at least two processors available the current process will not be changed by the kernel preventing expensive context switches. The creation of these FIFOs should ne negotiated over the main FIFO when implementing the multithreaded version.

3.2 Pint: a Pin based trace generator

Pint by itself it is not a simulator but a framework providing efficient ways to extract the data from the instrumented program through Mead. The version presented with this project is single threaded (although designed to be multithreaded and with part of the work for that already done) and relies on Mead for communicating with the simulator itself. As an example the provided instrumentation will study memory accesses made by the program (of any type ranging from prefetches to execution fetches) and sends them out with Mead so the simulator can prevent the issues associated with Pin tracing tools.

The code here is focused heavily on speed and thus the user must have the option to choose the features that should to be used.

The granularity of the execution can be easily tunned by setting an ap-

propriate queue size. For example for instruction by instruction execution the queue must have size one.

Also, a simulation started event must be the first one to be queued so you can discard old elements when you want the execution to be done.

Since all instructions will start (and contain) with a single fetch event it is possible to use this event as the differentiator between instructions. Anyway it is a good idea to integrate at least the number of events the instruction will cause to make tracing easier. This may be done on future versions.

Pint also provides a way to specify the number of instructions that must be executed before switching to the next simulation mode and thus you can provide the number of instructions that must be executed (by the sum of threads) before switching to another simulation mode.

The mode automaton allows for three simulation modes which are switched in the following order, the fast forward, the warm up and the simulation mode, which will then go back to the fast forward.

In the fast forward mode instructions are just accounted and executed but no data is generated which allows for near native speed execution. In the warm up mode and simulation mode instructions will generate events for filling the caches but the entrance and exit of the simulation status are notified to the consumer so it can handle statistics properly.

3.3 Schnapps: a simple consumer of the traces

Schnapps is intended to be used mainly for analyzing the performance of the instrumentation code by consuming the events generated whilst trying to avoid causing any bottlenecks in execution, and also as an example program of how to extract the generated traces.

Schnapps reads the traces generated by Pint and outputs the map changes as they happen (in a diff like format) and some statistics for the current simulation and for the total run, in particular, amount of data read or written by the different memory access types, the number of said accesses that has happened and an execution mark made by xoring the different accesses' addresses together the idea being that different marks imply different traces being generated but the same mark does not necessarily imply the same trace being generated. .

3.4 Gin5: a gem5 trace player

Although previous versions of gem5 came with a trace player supporting different formats, these modules stopped being maintained long time ago and those stopped building, as a result and despite being a good base for starting the work given the big amount of changes the memory system has suffered since then a different base was necessary.

As a result we have set again a generic CPU for playing traces (missing the TLB) which will ask for memory access request to the queues via a clear interface so it can be used also with other types of trace formats including the old ones if the classes containing them are updated.

Chapter 4

System design

4.1 Mead: a message passing framework

Mead has a macro of particular interest: USE_YIELD which will enable the use of the yield system call to let other process use the processor when waiting.

On mead, we have chosen to implement a lockless buffer ring over shared memory for our FIFOs since it is a well known pattern [4] [24].

The other reasons for choosing such an structure was speed and independence. By being lockless we avoid expensive spinlocks that would hinder performance whilst avoiding also having to either use the ones provided by Pin everywhere or building our own. Also having the data in shared memory will prevent us from making expensive system calls to have the messages passed and will allow the usage of cache for that.

It should be taken into account that the lockless queue will only work properly if a single thread acts as reader and a single thread acts as writer. In case of having more threads at either side they require a lock to work properly.

Our implementation is based on templates so it can be used with different classes although it should be taken into mind that the same class (i.e. no inheritance) should be used on the whole queue.

Also one of the current major caveats is that the shared memory address is currently hardcoded and, as a result, only a single instance of the program can be started at the same time. We expect to fix this in future versions by providing a launcher that will allocate an anonymous shared memory segment and pass its identifier to both Pint and the trace reader being used.

The command types are defined by the shmstatus enum. Since newly allocated shared memory is filled with 0s we assume the 0 value as the initial state (NONE). The server will then write a SERVER_STARTED command and wait for a CLIENT_ACK then. Finally when dying the server is expected to send the SERVER_DIED command so the client will not wait forever for data.

Of the many methods provided, those of special relevance for the programmer are the gethead and gettail methods used to be able to access the data we want to insert or extract from the queue, the push and pop methods used for adding or removing an element from the queue and the full and empty methods used to check for these states.

Also some methods for waiting in case the queue is empty/full are provided but these must be used carefully since if the consumer is singlethreaded it could hang waiting forever for the queue to match the condition. In this case special waits monitoring the main queue status too are recommened instead. Also a wait_push method is provided that will wait until a push can be done.

For control handling we provide the send_control, receive_control and ack_control methods. In particular send_control will wait until an ACK is

sent back to state the condition was taken care of.

Finally the wait_start and tell_Start methods are provided for initialization and instead of yields they use calls to the futex syscall to lock the thread until they have been attended to reduce the processor load in some situations.

4.2 Pint: a Pin based trace generator

In order to ensure unwanted features will not hinder performance preprocessor based switches can be used to disable those you are not interested in using, also some other options can be set in this way.

The macros of interest here are PADSIZE which defines the amount of bytes of the cache line in order to prevent false sharing, MAXMEMSIZE which defines the maximum size a single memory access may have (used for amongst other setting the size of the buffers), USE_DATA which will enable the infrastructure for fetching and sending the accessed data in the events, MULTITHREADED which will enable the still incomplete multithreaded code, USE_STATES which will enable the fast forward, warm up and simulation state machine and DTRACE which will make pint output some debugging information.

Given the impact these features can have we decided to allow the user disable them at compile time. Also some of these features can be disabled at run time although they will still have some impact on the execution, in particular, USE_STATES will still cause the slowdowns of the conditionals introduced before the instrumentation calls to handle the state machine and USE_DATA will make the event size, and thus the queues larger.

A final option that can be disabled is mapping tracing after a context

change (disabled by default) and after a syscall. The reason for this is the great slowdown caused by this operation since it requires at least 3 system calls in order to be executed and parsing a large text file.

In Pint the instrumentation is added by the Instruction function, when given the choice between adding complexity here or in the instrumentation functions we should add it here since this function is executed with much less frequency than the instrumentation code. As can be seen this function just tells Pin to add calls to the proper instrumentation functions, either with previous conditionals if the state machine is being used or without them otherwise.

Here we should consider all the parsemaps functions which are wrappers around the original parsemaps function that will take care of generating the events may maps be added or deleted. Also, as it can be seen, this function will consume quite a lot of resources given the way in which it works. Sadly the PIN framework on which pint is based does not provide any API in order to distinguish the mappings made by the instrumentations (including the JIT caches and the instrumentation code itself) as a result a lot of events will be generated on the simulation status queue. In order to reduce this overhead we assume mappings may only change after either coming back from a context change (as is the case when the application is being ptraced by a debugger) or coming back from a system call, this reduces the overhead greatly but still generates a lot of spurious mapping changes that may pollute the simulator assumptions. We expect this issue to be fixed with the addition of a proper API on future versions of PIN. Unlike memory access information given the importance of the mapping information it is sent independently of the simulation mode as it is generated.

In order to take track of the memory accesses the RecordMemExec,

RecordMemRead, RecordMemPrefetch and RecordMemPreWrite functions are used. Also when the user is insterested in the data generated by these functions, RecordMemPreWrite changes its behavior so it can access the memory information provided before the access and a new function called RecordMemWrite and executed after the instruction finishes is added, the reason for this is that the written data can not be known otherwise.

In order to handle the state machine we have an enum called state which contains the current simulation state, a function called nextState which takes care of handling the previous variable and the one with the instruction counter, and is called only when the instruction counter reaches zero, we also have the StateCounter method that will decrease the instruction counter by one and say whether we have processed the last instruction or, we also have CounterDone which sends the events for starting or ending a simulation and finally we have the Instrument function which check whether instrumentation code should or not be run in the current state.

We finally have a few callbacks, ThreadStart used to notify the creation of new threads, ThreadFini used to notify its destruction and Fini which is called before the instrumented program exits and will generate the SERVER_DIED event.

4.3 Schnapps: a simple consumer of the traces

The code on Schnapps is all written on the main function given its simplicity.

First, the queues are negotiated with Pint, afterwards, the variables holding the stats are initialized to 0 and we state we are not simulating anything.

With that done we enter the main loop that will process information until the trace generator reports that it has died. In this loop, the data from

the thread queue is extracted and added to the statistics. Afterwards, the main queue is checked for events like mappings being added/removed and these changes are printed. And finally control signals are handled properly, including the beginning of a simulation (by setting the stats to 0) and the end (by printing the simulation stats).

Finally, once outside of the loop and with the simulation finished, we print the total stats.

4.4 Gin5: a gem5 trace player

The biggest amount of coding is likely to have been written in these classes since we had to revamp the trace readers and the trace CPUs so they would work with the current memory system used by gem5.

The MemTraceReader class is a very simple class providing a single method called getNextRequest that will provide either a pointer to the next Request to be played on the memory system or a NULL pointer along with the reason why it was provided.

The memory requests are represented by the MemTraceRequest which returns packets through the getNextPkt method.

The PinReader class is derived from the MemTraceReader class and aside from handling the Pint queues also adds some callback to delete the queues when done.

Finally the TraceCPU class provides the MemPort classes and the Tick-Event classes which are required by the simulator and is the responsible of requesting the data to the reader when necessary and sending the requests to the memory system through the proper port. From a CPU point of view it emulates a system without a TLB (we basically take the LSBs of the address

to convert the virtual addresses we get into physical addresses) with ports for an instruction and a data cache.

An example gem5 configuration using this class is also provided in the pintrace.py file.

Chapter 5

Results

The benchmark results can be seen in the following table (extracted from the annexed .ods file).

It surprises us to get a slowdown as high as 804x in the case of LU and also the fact that fmm only got a 94x slowdown in the Graphite benchmarks. Anyway, if we discard the fmm benchmark we can see that our system performs better than graphite in all cases using a single processor.

Application	Graphite slowdown	Pint slowdown
barnes	N/A	287
cholesky	346	361
fft	3978	284
fmm	94	322
lu_cont	4007	557
lu_non_cont	3061	804
ocean_cont	515	317
ocean_non_cont	433	360
radiosity	N/A	498
radix	1648	199
raytrace	N/A	279
volrend	N/A	404
water_nsquared	2465	509
water_spatial	966	683

Table 5.1: Slowdown comparison between Graphite with 8 cores and Pint with one

Chapter 6

Conclusions

The project development has taken a long time given the research components it had yet, its development helped us have a good insight on how to improve simulators speed.

Also, given the promising results obtained with the benchmarks (worst case of 804 when simulating, best case of 199 with a mean of 360,5 and an average of 419) run during the development and testing of this fairly limited version we think that ideas like simulation segmentation and instrumentation based simulation on independent process may help to the development of faster and more powerful simulators and will continue with its development.

We expect to see in the future heavily multithreaded simulators where each processor has its own group of threads each handling the different stages independently in order to speed up execution times on multiprocessor machines.

We also expect to see in the future more simulators used the process based separation between the data collection routines responsible of the execution of the program and the simulation itself in order to allow for the usage of higher level languages with less restrictions whilst still providing high performance

and native execution of the simulated programs.

6.1 Improvements for next release

In the next release we intend to have a fully parallel instrumentation framework, we will also reimplement the trace simulator as a full gem5 CPU so it can have proper TLB handling and can be extended internally with more complex models. Finally we will change the queuing system so the simulator knows how many events will be generated by the instruction being executed before these events are handled down. We will also interconnect Multi2sim with gem5 in order to prove the powerfulness of Mead.

Once we release the next version we intend to publish a paper on a publication on this topic.

Appendix A

User manual

A.1 Building

In order to build the sources it suffices with running the `make` command on the sources directory.

A.2 Pint

Running the Pint pintool is quite easy and for that it is enough to run:
`./pin -t source/tools/SimpleExamples/obj-intel64/pinatrace.so - command arguments`
Options can be set by setting the desired switches between pinatrace.so and the –

Currently the following options are available:
`-f number` : adds the set number of instruction to be run in the fast forward state (used many times it will set more instruction counts to be run the next time we go back to said state)
`-w number` : adds the set number of instruction to be run in the warm up

state (used many times it will set more instruction counts to be run the next time we go back to said state)
`-s number` : adds the set number of instruction to be run in the simulation state (used many times it will set more instruction counts to be run the next time we go back to said state)
`-syscallmap {0,1}` : disables, if 0, or enables, if 1, the checking of process mappings after returning from a syscall
`-ctxchangemap {0,1}` : disables, if 0, or enables, if 1, the checking of process mappings after a context change
`-values {0,1}`: disables, if 0, or enables, if 1, the copying of data along with the memory events

A.3 Schnapps

For running Schnapps just run `./consumer`

A.4 Gin5

Gin5 requires a python file setting the system to be emulated. An example of such system can be found in the `pintrace.py` file. Once you have set up your system on a python file you just need to run the `gem5.fast` binary followed by the file containing the system being defined.

Scripts to set up systems may take arguments from the command line if introduced after the script file. Our example file does not make use of this feature but others may.

Appendix B

Relevant source code

pinatrace.h

```
1  #ifndef PINATRACE_H
2  #define PINATRACE_H
3  #include <linux/futex.h>
4  #include <sys/ipc.h>
5  #include <sys/sem.h>
6  #include <sys/shm.h>
7  #include <sys/syscall.h>
8  #include <sys/types.h>
9  #include <unistd.h>
10
11 #include <csignal>
12 #include <cstdio>
13 #include <cstdlib>
14 #include <cstring>
15 #include <new>
16
17 #define likely(x)       __builtin_expect(!!(x), 1)
18 #define unlikely(x)     __builtin_expect(!!(x), 0)
19
20 //Compile time configuration
21 #define PADSIZE 64          // 64 byte line size as per intel specs
22 #define MAXMEMSIZE 32       // 256 bits as per AVX, needs to be increased on the future
23 //#define USE_DATA 1        // Add codepaths to obtain the data on the memory accesses
24 //#define MULTITHREADED 1 // Add codepaths to allow for multithreaded applications
25 //#define USE_STATES 1      // Use a state machine to allow for fastforward and warm up states
26 //#define DTRACE 1          // Add debuging output
27 //#define USE_YIELD 1       // Use yield() or the pin equivalent when waiting for the other thread
28 #define QSIZE_ 1024         // Default FIFO queue size
29
30 #ifdef USE_YIELD
31 #ifdef PIN_H
32 #define YIELD PIN_Yield
33 #else
34 #include <sched.h>
35 #define YIELD sched_yield
36 #endif
37 #else
38 #define YIELD()
39 #endif
40
41 #define cachealigned __attribute__ ((aligned (PADSIZE)))
42
43
44 #ifndef PIN_H
45  typedef void VOID;
46  typedef u_int32_t UINT32;
47  typedef u_int8_t UINT8;
48 #endif
49
50 enum DataType { INVALDATA, STARTTH, ACCMEM };
```

```
51  enum AccessType {ACCEXEC, ACCREAD, ACCWRITE, ACCPREFETCH };
52
53
54  // #define PAD(n) ((((n) + ( PADSIZE - 1)) / PADSIZE ) * PADSIZE)
55
56  #include <cassert>
57
58  #ifdef DTRACE
59  #define dcprintf(c,...) if(c) fprintf (stderr, __VA_ARGS__)
60  #define dprintf(...) fprintf (stderr, __VA_ARGS__)
61  #define dcputs(c,a) if(c) fputs((a),stderr)
62  #define dputs(a) fputs((a),stderr)
63  #else
64  #define dcprintf(...)
65  #define dprintf(...)
66  #define dcputs(c,a)
67  #define dputs(a)
68  #endif
69
70  //TODO: use alignments instead of paddings
71  //TODO: use other padded struct for the data from read to write
72  class MemAccess {
73  private:
74      //We are not going to use derivate classes here for efficiency
75      AccessType type;
76      VOID * ea; // Effective address of the access
77      #ifdef USE_DATA
78      char data[MAXMEMSIZE]; // Contains either the data executed/read or the data contained before writing
79      char wdata[MAXMEMSIZE]; // This is valid only when the data access is a write contains the written data
80      #endif
81      UINT32 size; // Size of the access
82      #ifdef MULTITHREADED
83      UINT32 tid; // The ID of the thread generating the access
84      #endif
85      #ifdef USE_DATA
86      inline void setData() {
87          assert(PIN_SafeCopy(data, ea, size) == size);
88      }
89      inline void copyData(const MemAccess &ma) {
90              memcpy(data, ma.data, size);
91              if (type == ACCWRITE )
92                  memcpy(wdata, ma.wdata, size);
93      }
94      #endif
95  public:
96      #ifdef USE_DATA
97      inline void setWdata() {
98          assert(PIN_SafeCopy(wdata, ea, size) == size);
99      }
100     #endif
101     inline void MemAccessSet (AccessType type, VOID * ea, UINT32 size
102                               #ifdef MULTITHREADED
103                               , UINT32 tid
104                               #endif
105                               ) {
106         this->type = type;
107         this->ea = ea;
108         this->size = size;
109         #ifdef MULTITHREADED
110         this->tid = tid;
111         #endif
112         #ifdef USE_DATA
113         if (type != ACCPREFETCH) {
114             setData();
115         }
116         #endif
117     }
118     inline void MemAccessSet (const MemAccess &ma) {
119         type = ma.type;
120         ea = ma.ea;
121         size = ma.size;
122         #ifdef MULTITHREADED
123         tid = ma.tid;
124         #endif
125         #ifdef USE_DATA
126         if (type != ACCPREFETCH) {
127             copyData(ma);
128         }
129         #endif
130     }
131     void show(FILE *f); //Requires the C LOCK
132     inline AccessType getType() const { return type;}
133     inline void* getEA() const { return ea;}
```

```
134     #ifdef USE_DATA
135     inline const void* getData() const { return data;}
136     inline const void* getWData() const { return wdata;}
137     #endif
138     inline UINT32 getSize() const { return size;}
139     #ifdef MULTITHREADED
140     inline UINT32 getTid() const { return tid;}
141     #endif
142 } cachealigned;
143
144 union SimDataU {
145     class MemAccess ma;
146 };
147
148 class SimData {
149 private:
150     DataType type;
151     SimDataU data;
152 public:
153     SimData() : type(INVALDATA) {
154     }
155     SimData(DataType _type) : type(_type) {
156     }
157     inline DataType getType () const {
158         return type;
159     }
160     inline void setType (DataType _type) {
161         type = _type;
162     }
163     inline MemAccess & getMa () {
164         type = ACCMEM;
165         return data.ma;
166     }
167     inline const MemAccess & getCMa () const {
168         assert(type == ACCMEM);
169         return data.ma;
170     }
171 };
172
173 enum InstEventType {
174     ADDMAPPING, //A mapping was added during the last context change/syscall
175     REMOVEMAPPING, //A mapping was removed during the last context change/syscall
176     ADDTHREAD, //A new execution thread has been spawned
177     REMOVETHREAD, //An execution thread has ceased existing
178 };
179
180 struct range {
181     unsigned long int b; //begin
182     unsigned long int e; //end
183     inline bool operator < (const struct range &r) const {
184         //There shouldn't be overlapping ranges (at least in theory);
185         return b < r.b;
186     }
187 };
188
189 union InstEventData {
190     range r;
191     //TODO: handle per thread access queue creation here
192 };
193
194
195 class InstEvent {
196 private:
197     InstEventType type;
198     InstEventData data;
199 public:
200     inline InstEvent() { }
201     inline void SetInstEvent (InstEventType _type, range _r) {
202         type = _type;
203         data.r = _r;
204     }
205     inline InstEventType getType () const {
206         return type;
207     }
208     inline range getRange () {
209         assert(type == ADDMAPPING || type == REMOVEMAPPING);
210         return data.r;
211     }
212 } cachealigned;
213 // This is a class implementing lockless single producers single consumer queues
214 // They are very useful for fast efficient IPC through shared memory though you
215 // need to ensure the structure being queued has all the necessary data inside
216 // i.e. doesn't uses references.
```

```
217  // Currently we use them for two purpouses, passing events related to memory
218  // mappings and threads between the instrumentation and the simulator and
219  // passing around the memory acceses of each thread.
220
221  //We only use QSIZE -1 thus there is always one element free for processing before queueing.
222  #define NEXTQELEM(v) (((v) + 1) % QSIZE)
223
224  enum shmstatus {NONE = 0,//Initial state
225      CLIENT_ACK=1, //The client confirms reception of previous state
226      SERVER_STARTED=2, //The server has just started
227      SERVER_DIED=3, //The server has died
228      //This ones refer to the next instruction pushed to the queue (so they include up until the ACCEXEC after that)
229      SERVER_SIM_START=4, //We are going to jump into simulation reset stats
230      SERVER_SIM_END=5 //We have ended simulation reset stats
231  };
232
233
234  // A lockless single producer single consumer queue, with more than 1 you will need locks
235  template <class T, int QSIZE=QSIZE_> class SHMQ {
236      T queue[QSIZE] cachealigned;
237      volatile sig_atomic_t qhead cachealigned;
238      volatile sig_atomic_t qtail cachealigned;
239      // Elements are inserted on the head and removed from the tail like a snake.
240      volatile sig_atomic_t control  cachealigned;
241  public:
242      inline SHMQ () : qhead(0), qtail(0) {
243      }
244      inline T & gethead() { return queue[qhead]; }
245      inline T & gettail() {
246          assert(!empty());
247          return queue[qtail];
248      }
249      inline bool full() {return NEXTQELEM(qhead) == qtail; }
250      inline bool empty() {return qtail == qhead; }
251      //Wait for the queue not to be full
252      inline void wait_full() {
253          while(unlikely(full())) YIELD();
254      }
255      //Wait for the queue not to be empty (If the server dies it will never be)
256      inline bool wait_empty_cond() {
257          return (empty() && control == CLIENT_ACK);
258      }
259      inline void wait_empty() {
260          while(unlikely(wait_empty_cond())) YIELD();
261      }
262      inline void wait_not_empty() {
263          while(unlikely(!empty())) YIELD();
264      }
265      inline void push() {
266          assert(!full());
267          qhead = NEXTQELEM(qhead);
268      }
269      //Wait if necessary then push
270      inline void wait_push() {
271          wait_full();
272          push();
273      }
274      inline void pop() {
275          assert (!empty());
276          qtail = NEXTQELEM(qtail);
277      }
278      inline enum shmstatus receive_control () {
279          if (control == CLIENT_ACK) return NONE;
280          return (enum shmstatus)control;
281      }
282      inline void ack_control () {
283          while (unlikely(control == CLIENT_ACK)) YIELD();
284          control = CLIENT_ACK;
285      }
286      inline void send_control(enum shmstatus st) {
287          assert(st != CLIENT_ACK); //For this we should use ack_control instead
288          control = st;
289          //Wait for the ACK
290          while (unlikely(control != CLIENT_ACK)) YIELD();
291      }
292      inline void wait_start() {
293          sig_atomic_t control_;
294          while ((control_ = control) != SERVER_STARTED) syscall(SYS_futex, &control,FUTEX_WAIT,control_,NULL,NULL,0);
295          control = CLIENT_ACK;
296          syscall(SYS_futex, &control,FUTEX_WAKE,1,NULL,NULL,0);
297      }
298      inline void tell_start() {
299          sig_atomic_t control_;
```

```
300             control = SERVER_STARTED;
301             syscall(SYS_futex, &control,FUTEX_WAKE,1,NULL,NULL,0);
302             //Wait for the ACK
303             while ((control_ = control) != CLIENT_ACK) syscall(SYS_futex, &control,FUTEX_WAIT,SERVER_STARTED,NUL
304         }
305     };
306
307     typedef SHMQ<SimData> SimDataq;
308     typedef SHMQ<InstEvent> InstEventq;
309
310     SimDataq * server_init2();
311     SimDataq * client_init2();
312     void server_fini2(SimDataq *q);
313     void client_fini2(SimDataq *q);
314
315     //TODO: Fix the case where the client is the one doing the finalization
316
317     //TODO: access queues should be created dynamically and passed through the event queue
318     SimDataq * get_q2(int &shmid) {
319         SimDataq * q;
320         if ((shmid = shmget(2684, sizeof(SimDataq), IPC_CREAT | 0666)) < 0) {
321             perror("shmget");
322             exit(1);
323         }
324         void *shm;
325         if ((shm = shmat(shmid, NULL, 0)) == (void *) -1) {
326             perror("shmat");
327             exit(1);
328         }
329         q = static_cast<SimDataq*>(shm);
330         return q;
331     }
332
333
334     SimDataq * server_init2() {
335         int shmid;
336         SimDataq * q = get_q2(shmid);
337         new (q) SimDataq(); //We use a placement new so we have the SimDataq in the shared memory
338         q->tell_start();
339         return q;
340     }
341
342     SimDataq * client_init2() {
343         int shmid;
344         SimDataq * q = get_q2(shmid);
345         q->wait_start();
346         //Since we are connected we tell the OS the segement can be deleted
347         if (shmctl(shmid,IPC_RMID,NULL) < 0)
348             perror("shmctl");
349         return q;
350     }
351
352     void server_fini2(SimDataq *q) {
353         q->send_control(SERVER_DIED);
354     }
355
356     void client_fini2(SimDataq *q) {
357         q->ack_control();
358         q->~SimDataq();
359     }
360
361     //TODO: with propper template usage this could get prettier
362     InstEventq * server_init();
363     InstEventq * client_init();
364     void server_fini(InstEventq *q);
365     void client_fini(InstEventq *q);
366
367     InstEventq * get_q(int &shmid) {
368         InstEventq * q;
369         if ((shmid = shmget(2687, sizeof(InstEventq), IPC_CREAT | 0666)) < 0) {
370             perror("shmget");
371             exit(1);
372         }
373         void *shm;
374         if ((shm = shmat(shmid, NULL, 0)) == (void *) -1) {
375             perror("shmat");
376             exit(1);
377         }
378         q = static_cast<InstEventq*>(shm);
379         return q;
380     }
381
382
```

```
383  InstEventq * server_init() {
384      int shmid;
385      InstEventq * q = get_q(shmid);
386      new (q) InstEventq(); //We use a placement new so we have the InstEventq in the shared memory
387      q->tell_start();
388      return q;
389  }
390
391  InstEventq * client_init() {
392      int shmid;
393      InstEventq * q = get_q(shmid);
394      q->wait_start();
395      //Since we are connected we tell the OS the segement can be deleted
396      if (shmctl(shmid,IPC_RMID,NULL) < 0)
397          perror("shmctl");
398      return q;
399  }
400
401  void server_fini(InstEventq *q) {
402      q->send_control(SERVER_DIED);
403  }
404
405  void client_fini(InstEventq *q) {
406      q->ack_control();
407      q->~InstEventq();
408  }
409
410
411  #endif
```

pinatrace.cpp

```
1  /*BEGIN_LEGAL
2   * Intel Open Source License
3   *
4   * Copyright (c) 2002-2011 Intel Corporation. All rights reserved.
5   *
6   * Redistribution and use in source and binary forms, with or without
7   * modification, are permitted provided that the following conditions are
8   * met:
9   *
10  * Redistributions of source code must retain the above copyright notice,
11  * this list of conditions and the following disclaimer.  Redistributions
12  * in binary form must reproduce the above copyright notice, this list of
13  * conditions and the following disclaimer in the documentation and/or
14  * other materials provided with the distribution.  Neither the name of
15  * the Intel Corporation nor the names of its contributors may be used to
16  * endorse or promote products derived from this software without
17  * specific prior written permission.
18  *
19  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
20  * ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
21  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
22  * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE INTEL OR
23  * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
24  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
25  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
26  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
27  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
28  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
29  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
30  * END_LEGAL */
31 /* ================================================================== */
32 /*
33  *  @ORIGINAL_AUTHOR: Robert Cohn
34  */
35
36 /* ================================================================== */
37 /*! @file
38  *  This file contains an ISA-portable PIN tool for tracing memory accesses.
39  */
40
41 #include "pin.H"
42 #include "pinatrace.H"
43 #include <iostream>
44
45 #include <set>
46 #include <algorithm>
47
48 // Use when calling C and C++ library functions
49 PIN_LOCK c_lock;
50
51 // FILE *TraceFile;
52 FILE *StatsFile;
53
54 // KNOB<string> KnobOutputFile(KNOB_MODE_WRITEONCE, "pintool",
55 //                             "o", "pinatrace.out", "specify trace file name");
56 KNOB_COMMENT fcomment( "pintool:trace", "Options for the tracing behaviour");
57 #ifdef USE_STATES
58 KNOB<UINT64> Knobf(KNOB_MODE_APPEND, "pintool:trace",
59                    "f", "", "Number of instructions to fast forward. Must be used as many times as -w and
60 KNOB<UINT64> Knobw(KNOB_MODE_APPEND, "pintool:trace",
61                    "w", "", "Number of instructions to use for structure warming. Must be used as many times a
62 KNOB<UINT64> Knobs(KNOB_MODE_APPEND, "pintool:trace",
63                    "s", "", "Number of instructions to use for simulation. Must be used as many times as -f ar
64 #endif
65 KNOB_COMMENT mcomment( "pintool:trace", "Options for map tracing");
66 KNOB<BOOL> KnobSyscallMap(KNOB_MODE_WRITEONCE, "pintool:trace",
67                           "syscallmap", "1", "Check mappings after returning from a syscall");
68 KNOB<BOOL> KnobCtxChangeMap(KNOB_MODE_WRITEONCE, "pintool:trace",
69                           "ctxchangemap", "0", "Check mappings after a context change");
70 #ifdef USE_DATA
71 KNOB<BOOL> KnobValues(KNOB_MODE_WRITEONCE, "pintool:trace",
72                           "values", "1", "Output memory values reads and written");
73 #endif
74
75 #ifdef MULTITHREADED
76 PIN_LOCK h_lock;
77  static  TLS_KEY wMemAccess;
78 #define GetQLock(tid) GetLock(&(h_lock), tid+1)
79 #define ReleaseQLock() ReleaseLock(&(h_lock))
80 #else
81 #define GetQLock(tid)
82 #define ReleaseQLock()
```

```
83  #endif
84
85  static SimDataq *q;
86  static InstEventq *iq;
87
88  static FILE *mout;
89
90  void parsemaps(void) {
91      FILE *f;
92      static set<range> prev;
93      set<range> s, rem, add;
94      range r;
95      //TODO: we'll need to have a lock here to ensure threads don't collide
96      f = fopen("/proc/self/maps","r");
97      while (fscanf(f,"%lx-%lx",&(r.b),&(r.e)) == 2) {
98          s.insert(r);
99  //          Use this when wanting to copy the contents for verification
100 //          printf("%lx-%lx ",r.b,r.e);
101 //          int c;
102 //          while ((c = fgetc(f)) != '\n') putchar(c);
103 //          putchar('\n');
104         while (fgetc(f) != '\n');
105     };
106     //We don't need the file any more so release the FD
107     fclose(f);
108
109     //Notify removals first to prevent intersections
110     set_difference(prev.begin(), prev.end(), s.begin(), s.end(),
111                    inserter(rem, rem.begin()));
112     for (set<range>::iterator it = rem.begin(); it != rem.end(); it++) {
113         iq->gethead().SetInstEvent(REMOVEMAPPING,*it);
114         iq->wait_push();
115 //          printf("- %lx-%lx\n",it->b,it->e);
116     }
117
118     //Now notify additions
119     set_difference(s.begin(), s.end(), prev.begin(), prev.end(),
120                    inserter(add, add.begin()));
121     for (set<range>::iterator it = add.begin(); it != add.end(); it++) {
122         iq->gethead().SetInstEvent(ADDMAPPING,*it);
123         iq->wait_push();
124 //          printf("+ %lx-%lx\n",it->b,it->e);
125     }
126
127     //Ok finally we'll make the old set this one
128     prev = s;
129 }
130
131 void parsemaps1(THREADID _1, CONTEXT *_2, SYSCALL_STANDARD _3, VOID *_4) {
132     fputs("Syscall!\n",mout);
133     parsemaps();
134 }
135
136 void parsemaps2(THREADID _1, CONTEXT_CHANGE_REASON _2, const CONTEXT *_3, CONTEXT *_4, INT32 _5, VOID *_6) {
137     fputs("Context change!\n",mout);
138     parsemaps();
139 }
140
141 void parsemaps3(void) {
142     fputs("Initial!\n",mout);
143     parsemaps();
144 }
145
146
147 static char AccessType2Char (AccessType at) {
148     switch (at) {
149         case ACCEXEC: return 'X';
150         case ACCREAD: return 'R';
151         case ACCWRITE: return 'W';
152         case ACCPREFETCH: return 'P';
153         default: return '?';
154     }
155 }
156
157 #ifdef USE_DATA
158 static VOID EmitMem(FILE *f, VOID * data, UINT32 size)
159 {
160     switch(size)
161     {
162         case 0:
163             break;
164
165             /* TODO: Here we do some assuptions about sizes, a propper program should fill them properly*/
```

```
166                 case 1:
167                     fprintf(f,"0x%02hhx",(*static_cast<UINT8*>(data)));
168                     break;
169
170                 case 2:
171                     fprintf(f,"0x%04hx",(*static_cast<UINT16*>(data)));
172                     break;
173
174                 case 4:
175                     fprintf(f,"0x%08x",(*static_cast<UINT32*>(data)));
176                     break;
177
178                 case 8:
179                     fprintf(f,"0x%016lx",(*static_cast<UINT64*>(data)));
180                     break;
181
182                 default:
183                     if (size > 0) {
184                         fprintf(f,"0x%02hhx",(*static_cast<UINT8*>(data)));
185                         for (UINT32 i = 1; i < size; i++)
186                         {
187                             fprintf(f,"%02hhx",(static_cast<UINT8*>(data)[i]));
188                         }
189                     }
190                     break;
191     }
192 }
193 #endif
194
195 void MemAccess::show(FILE *f) {
196     fprintf(f,
197             #ifdef MULTITHREADED
198             "%10u"
199             #endif
200             "␣%c␣%#016lx%␣3d␣",
201             #ifdef MULTITHREADED
202             (UINT32) tid,
203             #endif
204             AccessType2Char(type), (unsigned long int) ea, size);
205     #ifdef USE_DATA
206     if (KnobValues) {
207         if (type != ACCPREFETCH) {
208             EmitMem(f, data, size);
209             if (type == ACCWRITE) {
210                 fputs("␣->␣",f);
211                 EmitMem(f, wdata, size);
212             }
213         }
214     }
215     #endif
216     fputs("\n",f);
217 }
218
219 static INT32 Usage()
220 {
221     fputs(
222     "This␣tool␣produces␣a␣memory␣address␣trace.\n"
223     "For␣each␣memory␣access␣(execute/read/write/prefetch)␣the␣ea␣is␣recorded\n"
224     "\n", stderr);
225
226     fputs(KNOB_BASE::StringKnobSummary().c_str(),stderr);
227
228     fputs("\n",stderr);
229
230     return -1;
231 }
232
233 static VOID PIN_FAST_ANALYSIS_CALL RecordMemExec(VOID * ip, UINT32 size
234                             #ifdef MULTITHREADED
235                             , THREADID tid
236                             #endif
237                            )
238 {
239     //TODO: use a per thread lockless queue
240     GetQLock(tid+1);
241     q->gethead().getMa().MemAccessSet(ACCEXEC,ip,size
242     #ifdef MULTITHREADED
243     ,tid
244     #endif
245     );
246     iq->wait_not_empty();
247     q->wait_push();
248     ReleaseQLock();
```

```
249  }
250
251  static VOID PIN_FAST_ANALYSIS_CALL RecordMemRead(VOID * ea, UINT32 size
252                                #ifdef MULTITHREADED
253                                , THREADID tid
254                                #endif
255                               )
256  {
257      //TODO: use a per thread lockless queue
258      GetQLock(tid+1);
259      q->gethead().getMa().MemAccessSet(ACCREAD, ea, size
260      #ifdef MULTITHREADED
261      ,tid
262      #endif
263      );
264      iq->wait_not_empty();
265      q->wait_push();
266      ReleaseQLock();
267  }
268
269  static VOID PIN_FAST_ANALYSIS_CALL RecordMemPrefetch(VOID * ea, UINT32 size
270                                    #ifdef MULTITHREADED
271                                    , THREADID tid
272                                    #endif
273                                   )
274  {
275      //TODO: use a per thread lockless queue
276      GetQLock(tid+1);
277      q->gethead().getMa().MemAccessSet(ACCPREFETCH, ea, size
278      #ifdef MULTITHREADED
279      ,tid
280      #endif
281      );
282      iq->wait_not_empty();
283      q->wait_push();
284      ReleaseQLock();
285  }
286
287
288  static VOID PIN_FAST_ANALYSIS_CALL RecordMemPreWrite(VOID * ea, UINT32 size
289                                    #ifdef MULTITHREADED
290                                    , THREADID tid
291                                    #endif
292                                   )
293  {
294      #ifdef USE_DATA
295      #ifdef MULTITHREADED
296      MemAccess *ma = static_cast<MemAccess *>(PIN_GetThreadData(wMemAccess, tid));
297      ma->MemAccessSet(ACCWRITE, ea, size, tid);
298      #else
299      q->gethead().getMa().MemAccessSet(ACCWRITE, ea, size);
300      #endif
301      #else
302      GetQLock(tid+1);
303      q->gethead().getMa().MemAccessSet(ACCWRITE, ea, size
304      #ifdef MULTITHREADED
305      ,tid
306      #endif
307      );
308      iq->wait_not_empty();
309      q->wait_push();
310      ReleaseQLock();
311      #endif
312
313  }
314
315
316  #ifdef USE_DATA
317  static VOID PIN_FAST_ANALYSIS_CALL RecordMemWrite(
318                                #ifdef MULTITHREADED
319                                THREADID tid
320                                #endif
321                               )
322  {
323      #ifdef MULTITHREADED
324      MemAccess *ma = static_cast<MemAccess *>(PIN_GetThreadData(wMemAccess, tid));
325      #ifdef USE_DATA
326      ma->setWdata();
327      #endif
328      #else
329      #ifdef USE_DATA
330      q->gethead().getMa().setWdata();
331      #endif
```

```
332     #endif
333
334     //TODO: use a per thread lockless queue
335     GetQLock(tid+1);
336     #ifdef MULTITHREADED
337     q->gethead().getMa().MemAccessSet(*ma);
338     #endif
339     iq->wait_not_empty();
340     q->wait_push();
341     ReleaseQLock();
342 }
343 #endif
344
345 #ifdef USE_STATES
346 static UINT64 inscount = -1;
347 static enum state { FASTFORWARD = 0, WARMING = 1, SIMULATION = 2} state = SIMULATION;
348 static UINT32 fIndex = 0;
349 static UINT32 wIndex = 0;
350 static UINT32 sIndex = 0;
351
352 inline VOID nextState() {
353     do {
354         switch (state) {
355             case FASTFORWARD:
356                 inscount = Knobw.Value(wIndex);
357                 wIndex++;
358                 state = WARMING;
359                 break;
360             case WARMING:
361                 inscount = Knobs.Value(sIndex);
362                 sIndex++;
363                 state = SIMULATION;
364                 break;
365             case SIMULATION:
366                 //If we are done simulating stop
367                 if (Knobf.NumberOfValues() == fIndex)
368                     PIN_ExitApplication(0);
369                 inscount = Knobf.Value(fIndex);
370                 fIndex++;
371                 state = FASTFORWARD;
372                 break;
373             default:
374                 fputs("Unknown state\n",stderr);
375                 PIN_ExitApplication(1);
376                 break;
377         }
378     } while (inscount == 0);
379 }
380
381 static ADDRINT PIN_FAST_ANALYSIS_CALL StateCounter(VOID * ip
382                                 #ifdef MULTITHREADED
383                                 THREADID tid
384                                 #endif
385                                )
386 {
387     inscount--;
388     dprintf("ins %p!\n",ip);
389     if (inscount == 0)
390         dputs("switch!\n");
391     return inscount == 0;
392 }
393
394 static ADDRINT PIN_FAST_ANALYSIS_CALL Instrument (
395 #ifdef MULTITHREADED
396 THREADID tid
397 #endif
398 )
399 {
400     return state != FASTFORWARD;
401 }
402
403
404 static VOID PIN_FAST_ANALYSIS_CALL CounterDone(
405                             #ifdef MULTITHREADED
406                             THREADID tid
407                             #endif
408                            )
409 {
410     enum state orig = state;
411     if (orig == SIMULATION) {
412         q->send_control(SERVER_SIM_END);
413     }
414     nextState();
```

```
415         if (state == SIMULATION) {
416             q->send_control(SERVER_SIM_START);
417         }
418         dprintf("state:%d->%d\n",orig,state);
419         //We only want to change the instrumentation when switching from any state to the fast forward state or viceversa
420 //      if ((orig == FASTFORWARD && state != FASTFORWARD)||(orig != FASTFORWARD && state == FASTFORWARD))
421 //TODO: this isn't working as expected :( (yet)
422 //          PIN_RemoveInstrumentation (); //reinstrument the program
423 }
424 #endif
425
426 //Instrumentation
427 static VOID Instruction(INS ins, VOID *v)
428 {
429     //Also using the IF - then callback system make PIN more likely to inline the counter code
430 #ifdef USE_STATES
431     INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR)StateCounter, IARG_FAST_ANALYSIS_CALL,
432                      IARG_INST_PTR,
433                      #ifdef MULTITHREADED
434                      IARG_THREAD_ID,
435                      #endif
436                      IARG_END);
437     INS_InsertThenCall(ins, IPOINT_BEFORE, (AFUNPTR)CounterDone, IARG_FAST_ANALYSIS_CALL,
438                        #ifdef MULTITHREADED
439                        IARG_THREAD_ID,
440                        #endif
441                        IARG_END);
442 #endif
443     //if (state != FASTFORWARD) {
444 #ifdef USE_STATES
445         INS_InsertIfCall(ins, IPOINT_BEFORE, (AFUNPTR)Instrument, IARG_FAST_ANALYSIS_CALL,
446                          #ifdef MULTITHREADED
447                          IARG_THREAD_ID,
448                          #endif
449                          IARG_END);
450         INS_InsertThenCall
451 #else
452         INS_InsertCall
453 #endif
454                          (ins, IPOINT_BEFORE, (AFUNPTR)RecordMemExec, IARG_FAST_ANALYSIS_CALL,
455                          IARG_INST_PTR,
456                          IARG_UINT32,INS_Size(ins),
457                          #ifdef MULTITHREADED
458                          IARG_THREAD_ID,
459                          #endif
460                          IARG_END);
461
462         if (INS_IsMemoryRead(ins))
463         {
464 #ifdef USE_STATES
465             INS_InsertIfPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR)Instrument, IARG_FAST_ANALYSIS_CALL,
466                                        #ifdef MULTITHREADED
467                                        IARG_THREAD_ID,
468                                        #endif
469                                        IARG_END);
470             INS_InsertThenPredicatedCall
471 #else
472             INS_InsertPredicatedCall
473 #endif
474                                         (ins, IPOINT_BEFORE,
475                                         (AFUNPTR)(INS_IsPrefetch(ins)?RecordMemPrefetch:RecordMemRead), IARG_FAST_ANALYS
476                                         IARG_MEMORYREAD_EA,
477                                         IARG_MEMORYREAD_SIZE,
478                                         #ifdef MULTITHREADED
479                                         IARG_THREAD_ID,
480                                         #endif
481                                         IARG_END);
482         }
483
484         if (INS_HasMemoryRead2(ins))
485         {
486 #ifdef USE_STATES
487             INS_InsertIfPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR)Instrument, IARG_FAST_ANALYSIS_CALL,
488                                          #ifdef MULTITHREADED
489                                          IARG_THREAD_ID,
490                                          #endif
491                                          IARG_END);
492             INS_InsertThenPredicatedCall
493 #else
494             INS_InsertPredicatedCall
495 #endif
496                                         (ins, IPOINT_BEFORE, (AFUNPTR)(INS_IsPrefetch(ins)?RecordMemPrefetch:RecordMemRe
497                                         IARG_MEMORYREAD2_EA,
```

```
498                                             IARG_MEMORYREAD_SIZE,
499                                             #ifdef MULTITHREADED
500                                             IARG_THREAD_ID,
501                                             #endif
502                                             IARG_END);
503             }
504
505             // instruments stores using a predicated call, i.e.
506             // the call happens iff the store will be actually executed
507             if (INS_IsMemoryWrite(ins))
508             {
509     #ifdef USE_STATES
510                 INS_InsertIfPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR)Instrument, IARG_FAST_ANALYSIS_CALL,
511                                            #ifdef MULTITHREADED
512                                            IARG_THREAD_ID,
513                                            #endif
514                                            IARG_END);
515                 INS_InsertThenPredicatedCall
516     #else
517                 INS_InsertPredicatedCall
518     #endif
519                                             (ins, IPOINT_BEFORE, (AFUNPTR)RecordMemPreWrite, IARG_FAST_ANALYSIS_CAL
520                                             IARG_MEMORYWRITE_EA,
521                                             IARG_MEMORYWRITE_SIZE,
522                                             #ifdef MULTITHREADED
523                                             IARG_THREAD_ID,
524                                             #endif
525                                             IARG_END);
526     #ifdef USE_DATA
527     #ifdef USE_STATES
528                 INS_InsertIfPredicatedCall(ins, IPOINT_BEFORE, (AFUNPTR)Instrument, IARG_FAST_ANALYSIS_CALL,
529                                            #ifdef MULTITHREADED
530                                            IARG_THREAD_ID,
531                                            #endif
532                                            IARG_END);
533                 INS_InsertThenPredicatedCall
534     #else
535                 INS_InsertPredicatedCall
536     #endif
537                                             (ins,
538                                             (!INS_HasFallThrough(ins)?IPOINT_TAKEN_BRANCH:IPOINT_AFTER),
539                                             (AFUNPTR)RecordMemWrite, IARG_FAST_ANALYSIS_CALL,
540                                             #ifdef MULTITHREADED
541                                             IARG_THREAD_ID,
542                                             #endif
543                                             IARG_END);
544     #endif
545             }
546         //}
547     }
548
549     // Multithread stuff:
550
551
552     #ifdef MULTITHREADED
553     static VOID ThreadStart(THREADID tid, CONTEXT *ctxt, INT32 flags, VOID *v)
554     {
555         MemAccess *ma = new MemAccess();
556         PIN_SetThreadData(wMemAccess, ma, tid);
557     }
558
559
560     static VOID ThreadFini(THREADID tid, const CONTEXT *ctxt, INT32 code, VOID *v)
561     {
562         MemAccess *ma = static_cast<MemAccess *>(PIN_GetThreadData(wMemAccess, tid));
563         delete ma;
564     }
565     #endif
566
567     //TODO: maybe integrate this into the queue class and the socket per thread protocol
568     // static bool ending = false;
569     // static THREADID processor;
570     // static PIN_THREAD_UID processoruid;
571     //
572     //static VOID ProcessQueue (VOID *nothing) {
573     //      THREADID tid = PIN_ThreadId();
574     //      while (!ending || !q->empty()) {
575     //          GetLock(&c_lock, tid);
576     //          while (!q->empty()) {
577     //              q->gettail();
578     //              q->pop();
579     //          }
580     //          ReleaseLock(&c_lock);
```

```
581 //          //Let others fill the queue
582 //          YIELD();
583 //      }
584 // }
585
586 static VOID Fini(INT32 code, VOID *v)
587 {
588 //      ending = true;
589 //      PIN_WaitForThreadTermination (processoruid,PIN_INFINITE_TIMEOUT,NULL);
590     if (KnobSyscallMap || KnobCtxChangeMap)
591         fclose(mout);
592     server_fini2(q);
593     server_fini(iq);
594 }
595
596 int main(int argc, char *argv[])
597 {
598     if( PIN_Init(argc,argv) )
599     {
600         return Usage();
601     }
602
603 #ifdef USE_STATES
604     if (!(Knobf.NumberOfValues() == Knobw.NumberOfValues() && Knobf.NumberOfValues()==Knobs.NumberOfValues()))
605     {
606         fputs("The number of occurrences of -f -h and -s must be the same.",stderr);
607         return Usage();
608     }
609 #endif
610
611     iq=server_init();
612     q=server_init2();
613     q->gethead().setType(STARTTH);//TODO move to the thread start callbacks
614     q->wait_push();
615
616 #ifdef USE_STATES
617     if(Knobf.NumberOfValues() >= 1)
618         nextState();
619     //This one is done due to the way instrumentation works
620     inscount++;
621     //Send the simu start command if necessary
622     if(state == SIMULATION)
623 #endif
624         q->send_control(SERVER_SIM_START);
625     INS_AddInstrumentFunction(Instruction, 0);
626     PIN_AddFiniUnlockedFunction(Fini, 0);
627
628     //Open the output file
629     if (KnobSyscallMap || KnobCtxChangeMap)
630         mout = fopen("maptrace.txt","w");
631
632
633     //Monitor syscalls and so for mapping changes
634     if (KnobSyscallMap)
635         PIN_AddSyscallExitFunction(parsemaps1, NULL);
636     //Monitor also after context changes since if we are ptraced mappings may have changed
637     if (KnobCtxChangeMap)
638         PIN_AddContextChangeFunction (parsemaps2, NULL);
639     //Although pin uses codecaches it hides this details from the instrumentation code so our instructions caches don
    break.
640     //This means the instruction addresses we get are mapped to the mappings corresponding to the libraries and not t
641     //code so we don't have to worry about changes to these mappings, but, since we still can't discern them from app
642     //mappings we still have to reserve space for them in the simulator space. This also means we'll be having some m
643     //in the map space almost always until PIN provides an api to discern pin/tool mappings from application ones.
644
645     //Thread Callbacks
646     InitLock(&c_lock);
647
648     #ifdef MULTITHREADED
649     InitLock(&h_lock);
650     wMemAccess = PIN_CreateThreadDataKey(0);
651     PIN_AddThreadStartFunction(ThreadStart, 0);
652     PIN_AddThreadFiniFunction(ThreadFini, 0);
653     #endif
654
655     //Start queue processor thread
656 //      processor = PIN_SpawnInternalThread (  ProcessQueue, NULL, 0, &processoruid);
657 //      if (processor == INVALID_THREADID) return 1;
658     //Initial map loading
659     if (KnobSyscallMap || KnobCtxChangeMap)
660         parsemaps3();
661     PIN_StartProgram();
662
```

```
663         return 0;
664     }
```

consumer.cpp

```
#include <pinatrace.h>
#include <cstdint>
#include <cinttypes>

int main() {
    InstEventq *iq;
    iq = client_init();
    SimDataq *q;
    q= client_init2();
    uint64_t nins = 0;
    uint64_t nins2 = 0;
    uint64_t nrea = 0;
    uint64_t nrea2 = 0;
    uint64_t nwri = 0;
    uint64_t nwri2 = 0;
    uint64_t npre = 0;
    uint64_t npre2 = 0;
    uint64_t sins = 0;
    uint64_t sins2 = 0;
    uint64_t srea = 0;
    uint64_t srea2 = 0;
    uint64_t swri = 0;
    uint64_t swri2 = 0;
    uint64_t spre = 0;
    uint64_t spre2 = 0;
    uint64_t mark = 0;
    uint64_t mark2 = 0;
    bool simulating = false;
    while (q->receive_control() != SERVER_DIED) {
        while(!q->empty()) {
            assert(q->gettail().getType() != INVALDATA);
            if (q->gettail().getType() == ACCMEM) {
                const MemAccess &ma = q->gettail().getCMa();
                mark  ^= (uint64_t) ma.getEA();
                mark2 ^= (uint64_t) ma.getEA();
                switch(ma.getType()) {
                    case ACCEXEC:
                        nins ++;
                        nins2++;
                        sins  += ma.getSize();
                        sins2 += ma.getSize();
                        break;
                    case ACCREAD:
                        nrea ++;
                        nrea2++;
                        srea  += ma.getSize();
                        srea2 += ma.getSize();
                        break;
                    case ACCWRITE:
                        nwri ++;
                        nwri2++;
                        swri  += ma.getSize();
                        swri2 += ma.getSize();
                        break;
                    case ACCPREFETCH:
                        npre ++;
                        npre2++;
                        spre  += ma.getSize();
                        spre2 += ma.getSize();
                        break;
                    default:
                        puts("Unexpected access type!");
                }
            }
            q->pop();
        }
        //Have we just emptied the buffer or has an event happened?
        while(!iq->empty()) {
            if (iq->gettail().getType() == REMOVEMAPPING){
                range r=iq->gettail().getRange();
                printf("- %lx-%lx\n",r.b,r.e);
            } else if (iq->gettail().getType() == ADDMAPPING){
                range r=iq->gettail().getRange();
                printf("+ %lx-%lx\n",r.b,r.e);
            }
            iq->pop();
        }
        switch(q->receive_control()) {
            case SERVER_DIED:
                if (!simulating)
                    break;
            case SERVER_SIM_END:
```

```
 83                  simulating = false;
 84                  puts  ("Simulation␣statistics:");
 85                  puts  ("Number␣of␣accesses:");
 86                  printf("␣␣instructions:␣%"PRIu64"\n",nins);
 87                  printf("␣␣reads␣␣␣␣␣␣␣:␣%"PRIu64"\n",nrea);
 88                  printf("␣␣writes␣␣␣␣␣␣:␣%"PRIu64"\n",nwri);
 89                  printf("␣␣prefetches␣␣:␣%"PRIu64"\n",npre);
 90                  printf("␣␣total␣␣␣␣␣␣␣:␣%"PRIu64"\n",nins+nrea+nwri+npre);
 91                  puts  ("Total␣accessed␣memory␣by␣type␣(bytes):");
 92                  printf("␣␣instructions:␣%"PRIu64"\n",sins);
 93                  printf("␣␣reads␣␣␣␣␣␣␣:␣%"PRIu64"\n",srea);
 94                  printf("␣␣writes␣␣␣␣␣␣:␣%"PRIu64"\n",swri);
 95                  printf("␣␣prefetches␣␣:␣%"PRIu64"\n",spre);
 96                  printf("␣␣total␣␣␣␣␣␣␣:␣%"PRIu64"\n",sins+srea+swri+spre);
 97                  printf("Execution␣mark:␣%"PRIx64"\n",mark);
 98                  q->ack_control ();
 99                  break;
100              case SERVER_SIM_START:
101                  nins = 0;
102                  nrea = 0;
103                  nwri = 0;
104                  npre = 0;
105                  sins = 0;
106                  srea = 0;
107                  swri = 0;
108                  spre = 0;
109                  mark = 0;
110                  simulating = true;
111                  q->ack_control ();
112                  break;
113          }
114          //Wait for buffer to refill
115          while(q->wait_empty_cond() && iq->wait_empty_cond()) YIELD();
116      }
117      puts  ("Total␣statistics:");
118      puts  ("Number␣of␣accesses:");
119      printf("␣␣instructions:␣%"PRIu64"\n",nins2);
120      printf("␣␣reads␣␣␣␣␣␣␣:␣%"PRIu64"\n",nrea2);
121      printf("␣␣writes␣␣␣␣␣␣:␣%"PRIu64"\n",nwri2);
122      printf("␣␣prefetches␣␣:␣%"PRIu64"\n",npre2);
123      printf("␣␣total␣␣␣␣␣␣␣:␣%"PRIu64"\n",nins2+nrea2+nwri2+npre2);
124      puts  ("Total␣accessed␣memory␣by␣type␣(bytes):");
125      printf("␣␣instructions:␣%"PRIu64"\n",sins2);
126      printf("␣␣reads␣␣␣␣␣␣␣:␣%"PRIu64"\n",srea2);
127      printf("␣␣writes␣␣␣␣␣␣:␣%"PRIu64"\n",swri2);
128      printf("␣␣prefetches␣␣:␣%"PRIu64"\n",spre2);
129      printf("␣␣total␣␣␣␣␣␣␣:␣%"PRIu64"\n",sins2+srea2+swri2+spre2);
130      printf("Execution␣mark:␣%"PRIx64"\n",mark2);
131      client_fini2(q);
132      client_fini(iq);
133      return 0;
134  }
```

mem_trace_reader.hh

```
1  /*
2   * Copyright (c) 2004-2005 The Regents of The University of Michigan
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions are
7   * met: redistributions of source code must retain the above copyright
8   * notice, this list of conditions and the following disclaimer;
9   * redistributions in binary form must reproduce the above copyright
10  * notice, this list of conditions and the following disclaimer in the
11  * documentation and/or other materials provided with the distribution;
12  * neither the name of the copyright holders nor the names of its
13  * contributors may be used to endorse or promote products derived from
14  * this software without specific prior written permission.
15  *
16  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
17  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
18  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
19  * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
20  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
21  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
22  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
23  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
24  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
25  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
26  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
27  *
28  * Authors: Erik Hallnor
29  */
30
31 /**
32  * Definitions for a pure virtual interface to a memory trace reader.
33  */
34
35 #ifndef __MEM_TRACE_READER_HH__
36 #define __MEM_TRACE_READER_HH__
37
38 #include "mem/packet.hh"
39 #include "mem/request.hh"
40 #include "params/MemTraceReader.hh"
41 #include "sim/sim_object.hh"
42
43 /**
44  * This class contains the info of the trace request and some useful methods to
45  * split it
46  */
47 class MemTraceRequest : public FastAlloc {
48     Addr _paddr;
49     unsigned _size;
50     Request::Flags _flags;
51     Tick _time;
52     int _asid;
53     Addr _vaddr;
54     int _contextId;
55     int _threadId;
56     Addr _pc;
57     MemCmd _cmd;
58 public:
59     MemTraceRequest () {}
60     MemTraceRequest(Addr paddr, int size, Request::Flags flags,
61                     MemCmd::Command cmd)
62     : _paddr(paddr), _size(size), _flags(flags), _time(curTick()), _cmd(cmd)
63     { }
64
65     MemTraceRequest(Addr paddr, int size, Request::Flags flags, Tick time,
66                     MemCmd::Command cmd)
67     : _paddr(paddr), _size(size), _flags(flags), _time(time), _cmd(cmd)
68     { }
69
70     ~MemTraceRequest() {}  // for FastAlloc
71
72     /**
73      * Are we scheduled to run already
74      */
75     inline bool mustRun () {
76         return _time <= curTick();
77     }
78
79     inline Tick time() {
80         return _time;
81     }
82
```

```
83      inline bool isInstFetch () {
84          return _flags.isSet(Request::INST_FETCH);
85      }
86
87      inline bool lastPacketSent () {
88          return _size == 0;
89      }
90
91      /**
92       * Get the next packet with proper bounds for this block size
93       * Will return NULL when done
94       */
95      PacketPtr getNextPkt (int bsize, Packet::NodeID dest, MasterID mid) {
96          if (lastPacketSent()) {
97              return NULL;
98          }
99          //Base address of the block
100         Addr base = (_paddr & ~(bsize - 1));
101         //Current block maxsize
102         int msize = bsize - (_paddr - base);
103         //Minimum
104         if (msize > _size) msize = _size;
105         //Generate tthe request and the packet
106         RequestPtr req = new Request(_paddr, msize, _flags, mid);
107         PacketPtr pkt = new Packet(req,_cmd,dest);
108         pkt->dataDynamicArray(new char[msize]);
109         //Calculate the new base address and size
110         _paddr += msize;
111         _size -= msize;
112         return pkt;
113     }
114 };
115
116 typedef MemTraceRequest * MemTraceRequestPtr;
117
118
119 /**
120  * Pure virtual base class for memory trace readers.
121  */
122 class MemTraceReader : public SimObject
123 {
124   public:
125       enum reason {EOT,STAT_RESET,STAT_DUMP};
126     /** Construct this MemoryTrace reader. */
127     MemTraceReader(const MemTraceReaderParams *p) : SimObject(p) {}
128
129 //TODO: redo doc pkt should contain time, request, command and data.
130     /**
131      * Read the next request from the trace. Returns the request in the
132      * provided RequestPtr and the cycle of the request in the return value.
133      * @param req Return the next request from the trace.
134      * @return The cycle of the request, 0 if none in trace.
135      */
136     virtual MemTraceRequestPtr getNextRequest(enum reason &reason) = 0;
137
138 };
139
140 #endif //__MEM_TRACE_READER_HH__
```

pin_reader.hh

```
1  /*
2   * Copyright (c) 2004-2005 The Regents of The University of Michigan
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions are
7   * met: redistributions of source code must retain the above copyright
8   * notice, this list of conditions and the following disclaimer;
9   * redistributions in binary form must reproduce the above copyright
10  * notice, this list of conditions and the following disclaimer in the
11  * documentation and/or other materials provided with the distribution;
12  * neither the name of the copyright holders nor the names of its
13  * contributors may be used to endorse or promote products derived from
14  * this software without specific prior written permission.
15  *
16  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
17  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
18  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
19  * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
20  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
21  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
22  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
23  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
24  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
25  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
26  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
27  *
28  * Authors: Erik Hallnor
29  */
30
31 /**
32  * @file
33  * Definition of a memory trace reader for a M5 memory trace.
34  */
35
36 #ifndef __Pin_READER_HH__
37 #define __Pin_READER_HH__
38
39 #include "cpu/trace/reader/mem_trace_reader.hh"
40 #include "cpu/trace/reader/pin_atrace.hh"
41 #include "params/PinReader.hh"
42
43 /**
44  * A memory trace reader for a pin memory trace.
45  */
46 class PinReader : public MemTraceReader
47 {
48     friend class DeleteQueuesCallback;
49     /** The trace. */
50     SimDataq *q;
51     /** Information on mapping changes */
52     InstEventq *iq;
53     bool simulating; //Wether we are in simulation state or not
54     bool drop; //Should we drop the next element (has it been processed)
55
56   protected:
57       void removeQueues();
58   public:
59     /**
60      * Construct an M5 memory trace reader.
61      */
62     PinReader(const PinReaderParams *p);
63
64     ~PinReader();
65
66
67     //TODO: redo doc pkt should contain time, request, command and data.
68     /**
69     * Read the next request from the trace. Returns the request in the
70     * provided RequestPtr and the cycle of the request in the return value.
71     * @param req Return the next request from the trace.
72     * @return The cycle of the request, 0 if none in trace.
73     */
74     virtual MemTraceRequestPtr getNextRequest(MemTraceReader::reason &reason);
75 };
76
77 #endif // __PIN_READER_HH__
```

pin_reader.cc

```
/*
 * Copyright (c) 2004-2005 The Regents of The University of Michigan
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met: redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer;
 * redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution;
 * neither the name of the copyright holders nor the names of its
 * contributors may be used to endorse or promote products derived from
 * this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * Authors: Erik Hallnor
 */

/**
 * @file
 * Declaration of a memory trace reader for a pin memory trace.
 */

#include "base/callback.hh"
#include "cpu/trace/reader/pin_reader.hh"
#include "sim/sim_exit.hh"
#include <set>

//TODO: look why the user interrupt received event doesn't calls the Callback

/** Callback to clean the queues*/
class DeleteQueuesCallback : public Callback {
public:
    DeleteQueuesCallback();
    void process();
};
static DeleteQueuesCallback dqc;


/** List of PinReader elements for the queue deleting callback **/
static std::set<PinReader *> readers;

DeleteQueuesCallback::DeleteQueuesCallback() {
    registerExitCallback(this);
}

void DeleteQueuesCallback::process() {
    for (std::set<PinReader *>::iterator it = readers.begin(); it != readers.end(); it++) {
        (*it)->removeQueues();
    }
}

//TODO: Send client finalization events if necessary

void PinReader::removeQueues() {
    if (q) {
        client_fini2(q);
        q = NULL;
    }
    if (iq) {
        client_fini(iq);
        iq = NULL;
    }
    warn("Done");
}

PinReader::PinReader(const PinReaderParams *p) : MemTraceReader(p),simulating(false) {
    iq = client_init();
    q = client_init2();
    //Wait for the initial event
```

```
83      while(q->empty()) { YIELD();}
84      drop = true;
85      readers.insert(this);
86  }
87
88  PinReader::~PinReader() {
89      removeQueues();
90      readers.erase(this);
91  }
92
93
94  MemTraceRequestPtr PinReader::getNextRequest(MemTraceReader::reason &reason)
95  {
96      MemCmd::Command cmd;
97      MemTraceRequestPtr req;
98      Request::Flags flags;
99      if (drop) {
100         assert(!q->empty());
101         q->pop(); // Drop previous data
102     }
103     while (true) {
104         //Wait for new traces if the server died just send NULL
105         while(q->wait_empty_cond() && iq->wait_empty_cond()) YIELD();
106         //TODO: this still needs some cleaning, the CPU must end any accesses before the reset, same before the dump
107         switch (q->receive_control()) {
108             case SERVER_DIED:
109                 //The last dump should be made by m5 itself
110                 reason = MemTraceReader::EOT;
111                 drop = false;
112                 return NULL;
113             case SERVER_SIM_END:
114                 simulating = false;
115                 q->ack_control();
116                 reason = MemTraceReader::STAT_DUMP;
117                 drop = false;
118                 return NULL;
119             case SERVER_SIM_START:
120                 simulating = true;
121                 q->ack_control();
122                 reason = MemTraceReader::STAT_RESET;
123                 drop = false;
124                 return NULL;
125             case NONE:
126                 break;
127             default:
128                 warn("State not supported!");
129         }
130         if (!q->empty()) {
131             switch (q->gettail().getType()) {
132                 case ACCMEM: {
133                     const MemAccess &ma = q->gettail().getCMa();
134                     switch (ma.getType()) {
135                         case ACCEXEC:
136                             flags.set(Request::INST_FETCH);
137                             cmd = MemCmd::ReadReq;
138                             break;
139                         case ACCREAD:
140                             cmd = MemCmd::ReadReq;
141                             break;
142                         case ACCWRITE:
143                             cmd = MemCmd::WriteReq;
144                             break;
145                         case ACCPREFETCH:
146                             flags.set(Request::PREFETCH);
147                             cmd = MemCmd::ReadReq;
148                             break;
149                         default:
150                             panic("Access type unknown");
151                     }
152                     Addr ea = (Addr)ma.getEA();
153                     ea &= (Addr)134217727; // 128Mb -1 :P
154                     //By default time is set to 0
155                     req = new MemTraceRequest((Addr)ea,(int)ma.getSize(),flags,cmd);
156                     drop = true;
157                     return req;
158                 }
159                 case STARTTH:
160                 case INVALDATA:
161                 default:
162                     panic("Unexpected data type");
163             }
164         }
165         while (!iq->empty()) {
```

```
166                 //Process mapping changes
167                 if (iq->gettail().getType() == REMOVEMAPPING) {
168                     //range r=iq->gettail().getRange();
169                     //TODO: remove mapping from TLB
170                 } else if (iq->gettail().getType() == ADDMAPPING) {
171                     //range r=iq->gettail().getRange();
172                     //TODO: add mapping from TLB
173                 }
174                 iq->pop();
175             }
176         }
177     }
178
179     PinReader *
180     PinReaderParams::create()
181     {
182         return new PinReader(this);
183     }
```

trace_cpu.hh

```
1  /*
2   * Copyright (c) 2004-2005 The Regents of The University of Michigan
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions are
7   * met: redistributions of source code must retain the above copyright
8   * notice, this list of conditions and the following disclaimer;
9   * redistributions in binary form must reproduce the above copyright
10  * notice, this list of conditions and the following disclaimer in the
11  * documentation and/or other materials provided with the distribution;
12  * neither the name of the copyright holders nor the names of its
13  * contributors may be used to endorse or promote products derived from
14  * this software without specific prior written permission.
15  *
16  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
17  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
18  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
19  * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
20  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
21  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
22  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
23  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
24  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
25  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
26  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
27  *
28  * Authors: Erik Hallnor
29  */
30
31 /**
32  * @file
33  * Declaration of a memory trace CPU object. Uses a memory trace to drive the
34  * provided memory hierarchy.
35  */
36
37 #ifndef __CPU_TRACE_TRACE_CPU_HH__
38 #define __CPU_TRACE_TRACE_CPU_HH__
39
40 #include <string>
41
42 #include "mem/mem_object.hh"
43 #include "mem/packet.hh" // for RequestPtr
44 #include "mem/port.hh"
45 #include "params/TraceCPU.hh"
46 #include "sim/eventq.hh" // for Event
47 #include "sim/sim_object.hh"
48
49 // Forward declaration.
50 class MemTraceReader;
51
52 enum CMD { Invalid, Read, Write, Writeback};
53
54 /**
55  * A cpu object for running memory traces through a memory hierarchy.
56  */
57 class TraceCPU : public MemObject
58 {
59   private:
60       class MemPort : public Port
61       {
62           TraceCPU *tcpu;
63           PacketPtr retryPkt;
64           bool accessRetry;
65       public:
66           MemPort(const std::string &_name, TraceCPU *_tcpu)
67           : Port(_name, _tcpu), tcpu(_tcpu)
68           { accessRetry = false; }
69
70           bool locked() {
71               return accessRetry;
72           }
73           void sendPkt(PacketPtr pkt);
74       protected:
75
76           virtual bool recvTiming(PacketPtr pkt);
77
78           virtual Tick recvAtomic(PacketPtr pkt);
79
80           virtual void recvFunctional(PacketPtr pkt);
81
82           virtual void recvRangeChange();
```

```
83
84              virtual void recvRetry();
85          };
86          /** Port for instruction trace requests, if any. */
87          MasterID _instMasterId;
88          MemPort icache;
89          /** Port for data trace requests, if any. */
90          MasterID _dataMasterId;
91          MemPort dcache;
92
93          /** Data reference trace. */
94          MemTraceReader *dataTrace;
95
96          /** Number of outstanding requests. */
97          int outstandingRequests;
98
99          /** Next packet conatining data, time, request, command, etc */
100         MemTraceRequestPtr nextRequest;
101
102         /** Reason for the packet to be NULL */
103         MemTraceReader::reason reason;
104
105         /** Next request. */
106         MemCmd::Command nextCmd;
107
108         /**
109          * Event to call the TraceCPU::tick
110          */
111         class TickEvent : public Event
112         {
113           private:
114             TraceCPU *cpu;
115
116           public:
117             TickEvent(TraceCPU *c) : Event(CPU_Tick_Pri), cpu(c) {}
118             void process() { cpu->tick(); }
119             virtual const char *description() const { return "TraceCPU tick"; }
120         };
121
122         TickEvent tickEvent;
123         inline Tick ticks(int numCycles) const { return numCycles; }
124
125       public:
126         /**
127          * Construct a TraceCPU object.
128          */
129         TraceCPU(const TraceCPUParams *p);
130
131         inline Tick ticks(int numCycles) { return numCycles; }
132
133         /**
134          * Perform all the accesses for one cycle.
135          */
136         void tick();
137
138         /**
139          * Handle a completed memory request.
140          */
141         void completeRequest(PacketPtr req);
142
143         virtual Port *getPort(const std::string &if_name, int idx = -1);
144     };
145
146     #endif // __CPU_TRACE_TRACE_CPU_HH__
```

trace_cpu.cc

```
1  /*
2   * Copyright (c) 2004-2005 The Regents of The University of Michigan
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without
6   * modification, are permitted provided that the following conditions are
7   * met: redistributions of source code must retain the above copyright
8   * notice, this list of conditions and the following disclaimer;
9   * redistributions in binary form must reproduce the above copyright
10  * notice, this list of conditions and the following disclaimer in the
11  * documentation and/or other materials provided with the distribution;
12  * neither the name of the copyright holders nor the names of its
13  * contributors may be used to endorse or promote products derived from
14  * this software without specific prior written permission.
15  *
16  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
17  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
18  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
19  * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
20  * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
21  * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
22  * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
23  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
24  * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
25  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
26  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
27  *
28  * Authors: Erik Hallnor
29  */
30
31 /**
32  * @file
33  * Declaration of a memory trace CPU object. Uses a memory trace to drive the
34  * provided memory hierarchy.
35  */
36
37 #include <algorithm> // For min
38
39 #include "cpu/trace/reader/mem_trace_reader.hh"
40 #include "cpu/trace/trace_cpu.hh"
41 // #include "mem/base_mem.hh" // For PARAM constructor
42 // #include "mem/mem_interface.hh"
43 //#include "params/TraceCPU.hh"
44 #include "base/statistics.hh"
45 #include "mem/packet.hh"
46 #include "sim/eventq.hh"
47 #include "sim/sim_events.hh"
48 #include "sim/sim_exit.hh"
49 #include "sim/system.hh"
50
51 using namespace std;
52
53 TraceCPU::TraceCPU(const TraceCPUParams *p)
54 : MemObject(p),
55 _instMasterId(p->sys->getMasterId(name() + ".inst")), icache("instructions",this),
56 _dataMasterId(p->sys->getMasterId(name() + ".data")), dcache("data",this),
57 dataTrace(p->trace), outstandingRequests(0), tickEvent(this)
58 {
59     nextRequest = dataTrace->getNextRequest(reason);
60     schedule(&tickEvent,curTick() + ticks(1));
61 }
62
63 //TODO: fix unaligned accesses out of block boundaries
64 void
65 TraceCPU::tick()
66 {
67     assert(outstandingRequests >= 0);
68     assert(outstandingRequests < 1000);
69     int instReqs = 0; //TODO convert to stats
70     int dataReqs = 0; //TODO convert to stats
71     while (!nextRequest) {
72         if ( outstandingRequests ) return;
73         switch (reason) {
74             case MemTraceReader::EOT:
75                 // No more requests to send. Finish trailing events and exit.
76                 //TODO: fix this
77                 //          if (queue()->empty()) {
78                     exitSimLoop("end of memory trace reached");
79                 //          } else {
80                     //              if(!tickEvent.scheduled())
81                 //                  schedule(&tickEvent,queue()->nextTick() + ticks(1));
82                 //          }
```

```
83                      return;
84                  case MemTraceReader::STAT_RESET:
85                      nextRequest = dataTrace->getNextRequest(reason);
86                      Stats::reset();
87                      break;
88                  case MemTraceReader::STAT_DUMP:
89                      nextRequest = dataTrace->getNextRequest(reason);
90                      Stats::dump();
91                      break;
92              }
93          }
94          if (nextRequest->mustRun()) {
95              int bsize = 0;
96              if (nextRequest->isInstFetch()) {
97                  bsize=icache.peerBlockSize();
98              } else {
99                  bsize=dcache.peerBlockSize();
100             }
101
102             //Rest of the request: get the new address and the new size
103             if (nextRequest->isInstFetch()) {
104                 PacketPtr nextPkt = nextRequest->getNextPkt(bsize,0,_instMasterId);
105                 //          assert(nextPkt->req->thread_num < 4 && "Not enough threads");
106                 nextPkt->setSrc(0);
107                 ++instReqs;
108 //                DPRINTF("id %d initiating %sread at addr %x (blk %x) expecting %x\n",
109 //                id, do_functional ? "functional " : "", req->getPaddr(),
110 //                blockAddr(req->getPaddr()), *result);
111                 icache.sendPkt(nextPkt);
112             } else {
113                 PacketPtr nextPkt = nextRequest->getNextPkt(bsize,0,_dataMasterId);
114                 //          assert(nextPkt->req->thread_num < 4 && "Not enough threads");
115                 nextPkt->setSrc(0);
116                 ++dataReqs;
117                 if(nextPkt->cmd.isRead()) {
118 //                    DPRINTF("id %d initiating %sread at addr %x (blk %x) expecting %x\n",
119 //                    id, do_functional ? "functional " : "", req->getPaddr(),
120 //                    blockAddr(req->getPaddr()), *result);
121                 } else if(nextPkt->cmd.isWrite()) {
122 //                    DPRINTF(MemTest, "initiating %swrite at addr %x (blk %x) value %x\n",
123 //                    do_functional ? "functional " : "", req->getPaddr(),
124 //                    blockAddr(req->getPaddr()), data & 0xff);
125                 } else panic("CMD not implemented");
126                 dcache.sendPkt(nextPkt);
127             }
128             //If we are done with the current packet we go for the next.
129             if (nextRequest->lastPacketSent()){
130                 delete nextRequest;
131                 nextRequest = dataTrace->getNextRequest(reason);
132             }
133         } else if(!tickEvent.scheduled())
134             schedule(&tickEvent,max(curTick() + ticks(1), (nextRequest?nextRequest->time():0)));
135     }
136
137     Port *
138     TraceCPU::getPort(const std::string &if_name, int idx)
139     {
140         if (if_name == "data")
141             return &dcache;
142         else if (if_name == "instructions")
143             return &icache;
144         else
145             panic("No Such Port\n");
146     }
147
148
149     bool
150     TraceCPU::MemPort::recvTiming(PacketPtr pkt)
151     {
152         if (pkt->isResponse()) {
153             tcpu->completeRequest(pkt);
154         } else {
155             // must be snoop upcall
156             assert(pkt->isRequest());
157             assert(pkt->getDest() == Packet::Broadcast);
158         }
159         return true;
160     }
161
162     Tick
163     TraceCPU::MemPort::recvAtomic(PacketPtr pkt)
164     {
165         panic("Atomic accesses not supported");
```

```
166         // must be snoop upcall
167         assert(pkt->isRequest());
168         assert(pkt->getDest() == Packet::Broadcast);
169         return curTick();
170     }
171
172     void
173     TraceCPU::MemPort::recvFunctional(PacketPtr pkt)
174     {
175         //Do nothing if we see one come through
176         return;
177     }
178
179     void
180     TraceCPU::MemPort::recvRangeChange()
181     {
182     }
183
184     void
185     TraceCPU::MemPort::recvRetry()
186     {
187         if (sendTiming(retryPkt)) {
188     //          DPRINTF(MemTest, "accessRetry setting to false\n");
189             accessRetry = false;
190             retryPkt = NULL;
191         }
192     }
193
194
195     void
196     TraceCPU::MemPort::sendPkt(PacketPtr pkt) {
197     //      if (atomic) {
198     //          cachePort.sendAtomic(pkt);
199     //          completeRequest(pkt);
200     //      }
201     //      else
202         tcpu->outstandingRequests++;
203         if (!sendTiming(pkt)) {
204     //          DPRINTF(MemTest, "accessRetry setting to true\n");
205
206             accessRetry = true;
207             retryPkt = pkt;
208         }
209     }
210
211     // TODO: handle stats
212     // void
213     // TraceCPU::regStats()
214     // {
215     //      using namespace Stats;
216     //
217     //      numReadsStat
218     //      .name(name() + ".num_reads")
219     //      .desc("number of read accesses completed")
220     //      ;
221     //
222     //      numWritesStat
223     //      .name(name() + ".num_writes")
224     //      .desc("number of write accesses completed")
225     //      ;
226     //
227     //      numExecsStat
228     //      .name(name() + ".num_exec")
229     //      .desc("number of execution accesses completed")
230     //      ;
231     // }
232
233     void
234     TraceCPU::completeRequest(PacketPtr pkt)
235     {
236         Request *req = pkt->req;
237
238         outstandingRequests--;
239     //      DPRINTF(MemTest, "completing %s at address %x (blk %x) %s\n",
240     //                  pkt->isWrite() ? "write" : "read",
241     //                  req->getPaddr(), blockAddr(req->getPaddr()),
242     //                  pkt->isError() ? "error" : "success");
243
244         //Remove the address from the list of outstanding
245
246         if (pkt->isError()) {
247             warn("Access failed for %s at %x\n",
248                     pkt->isWrite() ? "write" : "read", req->getPaddr());
```

```
249         } else {
250             //TODO: handle stats
251 //          if (pkt->isRead()) {
252 //              numReads++;
253 //              numReadsStat++;
254 //          } else {
255 //              assert(pkt->isWrite());
256 //              numWrites++;
257 //              numWritesStat++;
258 //          }
259         }
260 
261         pkt->deleteData();
262         delete pkt->req;
263         delete pkt;
264         if(!tickEvent.scheduled())
265             schedule(&tickEvent,max(curTick() + ticks(1), (nextRequest?nextRequest->time():0)));
266 }
267 
268 TraceCPU *
269 TraceCPUParams::create()
270 {
271     return new TraceCPU(this);
272 }
273 
274 /* To convert*/
275 
276 
277 // void
278 // MemTest::completeRequest(PacketPtr pkt)
279 // {
280 //      Request *req = pkt->req;
281 //
282 //      if (issueDmas) {
283 //          dmaOutstanding = false;
284 //      }
285 //
286 //      DPRINTF(MemTest, "completing %s at address %x (blk %x) %s\n",
287 //              pkt->isWrite() ? "write" : "read",
288 //              req->getPaddr(), blockAddr(req->getPaddr()),
289 //              pkt->isError() ? "error" : "success");
290 //
291 //      MemTestSenderState *state =
292 //      dynamic_cast<MemTestSenderState *>(pkt->senderState);
293 //
294 //      uint8_t *data = state->data;
295 //      uint8_t *pkt_data = pkt->getPtr<uint8_t>();
296 //
297 //      //Remove the address from the list of outstanding
298 //      std::set<unsigned>::iterator removeAddr =
299 //      outstandingAddrs.find(req->getPaddr());
300 //      assert(removeAddr != outstandingAddrs.end());
301 //      outstandingAddrs.erase(removeAddr);
302 //
303 //      if (pkt->isError()) {
304 //          if (!suppress_func_warnings) {
305 //              warn("Functional Access failed for %x at %x\n",
306 //                   pkt->isWrite() ? "write" : "read", req->getPaddr());
307 //          }
308 //      } else {
309 //          if (pkt->isRead()) {
310 //              if (memcmp(pkt_data, data, pkt->getSize()) != 0) {
311 //                  panic("%s: read of %x (blk %x) @ cycle %d "
312 //                  "returns %x, expected %x\n", name(),
313 //                        req->getPaddr(), blockAddr(req->getPaddr()), curTick(),
314 //                        *pkt_data, *data);
315 //              }
316 //
317 //              numReads++;
318 //              numReadsStat++;
319 //
320 //              if (numReads == (uint64_t)nextProgressMessage) {
321 //                  ccprintf(cerr, "%s: completed %d read, %d write accesses @%d\n",
322 //                           name(), numReads, numWrites, curTick());
323 //                  nextProgressMessage += progressInterval;
324 //              }
325 //
326 //              if (maxLoads != 0 && numReads >= maxLoads)
327 //                  exitSimLoop("maximum number of loads reached");
328 //          } else {
329 //              assert(pkt->isWrite());
330 //              funcPort.writeBlob(req->getPaddr(), pkt_data, req->getSize());
331 //              numWrites++;
```

```
332 //              numWritesStat++;
333 //          }
334 //      }
335 //
336 //      noResponseCycles = 0;
337 //      delete state;
338 //      delete [] data;
339 //      delete pkt->req;
340 //      delete pkt;
341 //      if (!tickEvent.scheduled())
342 //          schedule(tickEvent, curTick() + ticks(1));
343 // }
344
345 // void
346 // MemTest::tick()
347 // {
348 //
349 //      //make new request
350 //      /*      unsigned cmd = random() % 100;
351 //       *      unsigned offset = random() % size;
352 //       *      unsigned base = random() % 2;
353 //       *      uint64_t data = random();
354 //       *      unsigned access_size = random() % 4;
355 //       *      bool uncacheable = (random() % 100) < percentUncacheable;
356 //       *
357 //       *      unsigned dma_access_size = random() % 4; */
358 //      unsigned cmd = 0;
359 //      offset++;
360 //      unsigned base = 0;
361 //      uint64_t data = random();
362 //      unsigned access_size = 0;
363 //      bool uncacheable = false;
364 //
365 //      unsigned dma_access_size = random() % 4;
366 //
367 //      //If we aren't doing copies, use id as offset, and do a false sharing
368 //      //mem tester
369 //      //We can eliminate the lower bits of the offset, and then use the id
370 //      //to offset within the blks
371 //      //      offset = blockAddr(offset);
372 //      //      offset += id;
373 //      //      access_size = 0;
374 //      //      dma_access_size = 0;
375 //
376 //      Request *req = new Request();
377 //      Request::Flags flags;
378 //      Addr paddr;
379 //
380 //      if (uncacheable) {
381 //          flags.set(Request::UNCACHEABLE);
382 //          paddr = uncacheAddr + offset;
383 //      } else  {
384 //          paddr = ((base) ? baseAddr1 : baseAddr2) + offset;
385 //      }
386 //      bool do_functional = false;
387 //
388 //      if (issueDmas) {
389 //          paddr &= ~((1 << dma_access_size) - 1);
390 //          req->setPhys(paddr, 1 << dma_access_size, flags);
391 //          req->setThreadContext(id,0);
392 //      } else {
393 //          paddr &= ~((1 << access_size) - 1);
394 //          req->setPhys(paddr, 1 << access_size, flags);
395 //          req->setThreadContext(id,0);
396 //      }
397 //      assert(req->getSize() == 1);
398 //
399 //      uint8_t *result = new uint8_t[8];
400 //
401 //      if (cmd < percentReads) {
402 //          // read
403 //
404 //          // For now we only allow one outstanding request per address
405 //          // per tester This means we assume CPU does write forwarding
406 //          // to reads that alias something in the cpu store buffer.
407 //          if (outstandingAddrs.find(paddr) != outstandingAddrs.end()) {
408 //              delete [] result;
409 //              delete req;
410 //              return;
411 //          }
412 //
413 //          outstandingAddrs.insert(paddr);
414 //
```

```
415 //          // ***** NOTE FOR RON: I'm not sure how to access checkMem. - Kevin
416 //          funcPort.readBlob(req->getPaddr(), result, req->getSize());
417 //
418 //          ccprintf(cerr,
419 //                   "id %d initiating %sread at addr %x (blk %x) expecting %x\n",
420 //                   id, do_functional ? "functional " : "", req->getPaddr(),
421 //                   blockAddr(req->getPaddr()), *result);
422 //
423 //          PacketPtr pkt = new Packet(req, MemCmd::ReadReq, Packet::Broadcast);
424 //          pkt->setSrc(0);
425 //          pkt->dataDynamicArray(new uint8_t[req->getSize()]);
426 //          MemTestSenderState *state = new MemTestSenderState(result);
427 //          pkt->senderState = state;
428 //
429 //          if (do_functional) {
430 //              assert(pkt->needsResponse());
431 //              pkt->setSuppressFuncError();
432 //              cachePort.sendFunctional(pkt);
433 //              completeRequest(pkt);
434 //          } else {
435 //              sendPkt(pkt);
436 //          }
437 //      } else {
438 //          // write
439 //
440 //          // For now we only allow one outstanding request per addreess
441 //          // per tester.  This means we assume CPU does write forwarding
442 //          // to reads that alias something in the cpu store buffer.
443 //          if (outstandingAddrs.find(paddr) != outstandingAddrs.end()) {
444 //              delete [] result;
445 //              delete req;
446 //              return;
447 //          }
448 //
449 //          outstandingAddrs.insert(paddr);
450 //
451 //          DPRINTF(MemTest, "initiating %swrite at addr %x (blk %x) value %x\n",
452 //                  do_functional ? "functional " : "", req->getPaddr(),
453 //                  blockAddr(req->getPaddr()), data & 0xff);
454 //
455 //          PacketPtr pkt = new Packet(req, MemCmd::WriteReq, Packet::Broadcast);
456 //          pkt->setSrc(0);
457 //          uint8_t *pkt_data = new uint8_t[req->getSize()];
458 //          pkt->dataDynamicArray(pkt_data);
459 //          memcpy(pkt_data, &data, req->getSize());
460 //          MemTestSenderState *state = new MemTestSenderState(result);
461 //          pkt->senderState = state;
462 //
463 //          if (do_functional) {
464 //              pkt->setSuppressFuncError();
465 //              cachePort.sendFunctional(pkt);
466 //              completeRequest(pkt);
467 //          } else {
468 //              sendPkt(pkt);
469 //          }
470 //      }
471 // }
```

pintrace.py

```
# Copyright (c) 2006-2007 The Regents of The University of Michigan
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are
# met: redistributions of source code must retain the above copyright
# notice, this list of conditions and the following disclaimer;
# redistributions in binary form must reproduce the above copyright
# notice, this list of conditions and the following disclaimer in the
# documentation and/or other materials provided with the distribution;
# neither the name of the copyright holders nor the names of its
# contributors may be used to endorse or promote products derived from
# this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
# A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
# OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
# LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
# Authors: Ron Dreslinski

import optparse
import sys

import m5
from m5.objects import *

parser = optparse.OptionParser()

#parser.add_option("-m", "--maxtick", type="int", default=m5.MaxTick,
                  #metavar="T",
                  #help="Stop after T ticks")

(options, args) = parser.parse_args()

if args:
     print "Error: script doesn't take any positional arguments"
     sys.exit(1)

# define prototype L1 cache
proto_l1 = BaseCache(size = '32kB', assoc = 4, block_size = 128,
                     latency = '1ns', tgts_per_mshr = 1)

proto_l1.mshrs = 1

pr = PinReader()

tcpu = TraceCPU(trace = pr)

# next comes L1 cache, if any
#prototypes.insert(0, proto_l1)

# system simulated

system = System(physmem = PhysicalMemory(latency = "100ns"))

new_bus = Bus(clock="500MHz", width=16)
system.physmem.cpu_side_bus = new_bus
system.physmem.port = new_bus.master

data_l1 = BaseCache(size = '32kB', assoc = 4, block_size = 64,
                    latency = '1ns', tgts_per_mshr = 8)
data_l1.mshrs = 1

ins_l1 = BaseCache(size = '32kB', assoc = 4, block_size = 64,
                   latency = '1ns', tgts_per_mshr = 8)
ins_l1.mshrs = 1

new_bus.cache = [ data_l1, ins_l1 ]
new_bus.slave = data_l1.mem_side
new_bus.slave = ins_l1.mem_side

data_l1.cpu = tcpu

tcpu.data = data_l1.cpu_side
```

```
83  tcpu.instructions = ins_l1.cpu_side
84  #def make_level(spec, prototypes, attach_obj, attach_port):
85        #fanout = spec[0]
86        #parent = attach_obj # use attach obj as config parent too
87        #if len(spec) > 1 and (fanout > 1 or options.force_bus):
88              #new_bus = Bus(clock="500MHz", width=16)
89              #new_bus.port = getattr(attach_obj, attach_port)
90              #parent.cpu_side_bus = new_bus
91              #attach_obj = new_bus
92              #attach_port = "port"
93        #objs = [prototypes[0]() for i in xrange(fanout)]
94        #if len(spec) > 1:
95              ## we just built caches, more levels to go
96              #parent.cache = objs
97              #for cache in objs:
98                    #cache.mem_side = getattr(attach_obj, attach_port)
99                    #make_level(spec[1:], prototypes[1:], cache, "cpu_side")
100       #else:
101             ## we just built the MemTest objects
102             #parent.cpu = objs
103             #for t in objs:
104                   #t.test = getattr(attach_obj, attach_port)
105                   #t.functional = system.funcmem.port
106
107 #make_level(treespec, prototypes, system.physmem, "port")
108
109 # ----------------------
110 # run simulation
111 # ----------------------
112
113 root = Root( full_system = False, system = system )
114 root.system.mem_mode = 'timing'
115
116 root.system.system_port = root.system.physmem.port
117
118 # Not much point in this being higher than the L1 latency
119 m5.ticks.setGlobalFrequency('1ns')
120
121 # instantiate configuration
122 m5.instantiate()
123
124 # simulate until program terminates
125 exit_event = m5.simulate(m5.MaxTick)
126
127 print 'Exiting @ tick', m5.curTick(), 'because', exit_event.getCause()
```

Bibliography

[1] Kenneth Barr. *Dinerotool*. Oct. 2005. URL: http://kbarr.net.

[2] Nathan Binkert et al. "The gem5 simulator". In: *SIGARCH Comput. Archit. News* 39.2 (Aug. 2011), pp. 1–7. ISSN: 0163-5964. DOI: 10.1145/2024716.2024718. URL: http://doi.acm.org/10.1145/2024716.2024718.

[3] Zhongliang Chen et al. *The Multi2Sim Simulation Framework*. URL: http://www.multi2sim.org/files/multi2sim-r277.pdf.

[4] *Circular buffer*. Nov. 2012. URL: http://en.wikipedia.org/w/index.php?title=Circular_buffer&oldid=522370238#Always_Keep_One_Slot_Open.

[5] H. J. Curnow and B. A. Wichmann. "A synthetic benchmark". In: *The Computer Journal* 19.1 (1976), pp. 43–49. DOI: 10.1093/comjnl/19.1.43. eprint: http://comjnl.oxfordjournals.org/content/19/1/43.full.pdf+html. URL: http://comjnl.oxfordjournals.org/content/19/1/43.abstract.

[6] Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. "Gprof: A call graph execution profiler". In: *SIGPLAN Not.* 17.6 (June 1982), pp. 120–126. ISSN: 0362-1340. DOI: 10.1145/872726.806987. URL: http://doi.acm.org/10.1145/872726.806987.

[7] Mark Hill and Jan Edler. *Dinero IV Trace-Driven Uniprocessor Cache Simulator*. Feb. 1998. URL: http://www.cs.wisc.edu/~markhill/DineroIV/.

[8] Chi-Keung Luk et al. "Pin: building customized program analysis tools with dynamic instrumentation". In: *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*. PLDI '05. Chicago, IL, USA: ACM, 2005, pp. 190–200. ISBN: 1-59593-056-6. DOI: 10.1145/1065010.1065034. URL: http://doi.acm.org/10.1145/1065010.1065034.

[9] J.E. Miller et al. "Graphite: A distributed parallel simulator for multicores". In: *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. Jan. 2010, pp. 1 –12. DOI: 10.1109/HPCA.2010.5416635. URL: http://groups.csail.mit.edu/carbon/docs/graphite_hpca2010_preprint.pdf.

[10] Vijay Janapa Reddi et al. "PIN: a binary instrumentation tool for computer architecture research and education". In: *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*. WCAE '04. Munich, Germany: ACM, 2004. DOI: 10.1145/1275571.1275600. URL: http://doi.acm.org/10.1145/1275571.1275600.

[11] Cloyce D. Spradling. "SPEC CPU2006 Benchmark Tools". In: *SIGARCH Computer Architecture News* 35 (1 Mar. 2007).

[12] Richard M. Stallman. *GDB manual: the GNU source-level debugger*. 2nd, GDB version 2.5. Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA, Tel: (617) 876-3296, Feb. 1988, pp. ii + 63.

[13] Richard M. Stallman. *Using and Porting GNU CC*. Tech. rep. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA, Tel: (617) 876-3296: Free Software Foundation, Inc., 1988.

[14] *The gcc website*. URL: http://gcc.gnu.org/.

[15] *The gdb website*. URL: http://www.gnu.org/software/gdb/.

[16] *The Gem5 website*. URL: http://www.gem5.org/.

[17] *The gprof website*. URL: http://sourceware.org/binutils/docs/gprof/.

[18] *The Graphite website*. URL: http://groups.csail.mit.edu/carbon/?page_id=111.

[19] *The modified SPLASH-2 website*. URL: www.capsl.udel.edu/splash/.

[20] *The Multi2Sim website*. URL: http://www.multi2sim.org/.

[21] *The Pin website*. URL: http://software.intel.com/en-us/articles/pintool/.

[22] *The SPEC CPU2006 website*. URL: http://www.spec.org/cpu2006/.

[23] *The SPLASH-2 website*. URL: http://web.archive.org/web/http://www-flash.stanford.edu/apps/SPLASH/.

[24] vanDooren. *Creating a thread safe producer consumer queue in C++ without using locks*. Jan. 2007. URL: http://msmvps.com/blogs/vandooren/archive/2007/01/05/creating-a-thread-safe-producer-consumer-queue-in-c-without-using-locks.aspx.

[25] Reinhold P. Weicker. "Dhrystone: a synthetic systems programming benchmark". In: *Commun. ACM* 27.10 (Oct. 1984), pp. 1013–1030. ISSN: 0001-0782. DOI: 10.1145/358274.358283. URL: http://doi.acm.org/10.1145/358274.358283.

[26] S.C. Woo et al. “The SPLASH-2 Programs: Characterization and Methodological Considerations”. In: *Proc. of the 22nd International Symposium on Computer Architecture*. June 1995.