APÉNDICE A
Scripts de Tratamiento de Datos

En el presente anexo se explica de forma concisa los *scripts* realizados para la extracción de datos en Python. El objetivo es poder consultar el código de extracción de datos de forma sencilla si se requiriese. De esta manera se adjuntan los ejecutables que tienen como objetivo el tratamiento de los datos para obtener resultados interpretables rápidamente y sin esfuerzo.

A.1 createDics.py

Esta función se encarga de recorrer el directorio donde se encuentran las aplicaciones de SPEC CPU 2006 y genera un diccionario de aplicaciones, que a su vez contiene una estructura matricial donde se guardan los datos a tratar. Esta función se utiliza posteriormente por todos los *scripts* para generar matrices de datos de las aplicaciones.

```
from __future__ import print_function
import pandas as pd
import os

#Def dictionarios anidados
def create_dic(path):
    pid_dictionary = dict()

    for bench in sorted(os.listdir(path)):
        int_reps_dir = path + bench + "/interval_reports/x86_ctx/"


        if len(os.listdir(path)) < 1:
            print("Benchmark " + bench + " err: no input files available")
            continue

        pid_dict = dict()
        for pid in os.listdir(int_reps_dir):
            if pid.endswith("~") == False:
                pidid = pid.split('.')[0]
                data_frame = pd.read_csv(int_reps_dir + pid, sep=',',
                header=0, usecols=None)
                pid_dict[pidid] = data_frame
```

```
        pid_dictionary[bench] = pid_dict

    return pid_dictionary



def create_dic_mod(path):
    pid_dictionary = dict()

    for bench in sorted(os.listdir(path)):
        int_reps_dir = path + bench + "/interval_reports/mod/"

        if len(os.listdir(path)) < 1:
            print("Benchmark " + bench + " err: no input files available")
            continue

        pid_dict = dict()
        for pid in os.listdir(int_reps_dir):
            if (pid == "dl1-0.intrep.csv"):
                if pid.endswith("~") == False:
                    pidid = pid.split('.')[0]
                    data_frame = pd.read_csv(int_reps_dir + pid, sep=',',
                                             header=0, usecols=None)
                    pid_dict[pidid] = data_frame

        pid_dictionary[bench] = pid_dict

    return pid_dictionary

def create_dic_p(path):
    pid_dictionary = dict()

    for bench in sorted(os.listdir(path)):
        # MOD AQUI
        int_reps_dir = path + bench + "/interval_reports/mod/"

        if len(os.listdir(path)) < 1:
            print("Benchmark " + bench + " err: no input files available")
            continue

        pid_dict = dict()
        for pid in os.listdir(int_reps_dir):
            if (pid == "dl1-0.intrep.csv"):
                if pid.endswith("~") == False:
                    pidid = pid.split('.')[0]
                    data_frame = pd.read_csv(int_reps_dir + pid, sep=',',
                                             header=0, usecols=None)
                    pid_dict[pidid] = data_frame

        pid_dictionary[bench] = pid_dict
```

```
    return pid_dictionary
```

A.2 generarCondorFile.py

Este *script* devuelve como resultado un fichero de texto para lanzarlo a la cola de Condor mediante el comando *condor_submit*. Lo genera a partir de una serie de parámetros como la ubicación de los ficheros de configuración de Multi2Sim, la cantidad de instrucciones a ejecutar, los periodos de volcado de datos, etcétera.

```
from __future__ import print_function
from sys import argv

import sys
import os

def files( path ):
        for file in os.listdir(path):
                if os.path.isfile(os.path.join(path, file)):
                        yield file

def create_file_header( condor_file, m2s_path ):
        condor = open(condor_file, "w")
        print ("+GPBatchJob = true", file=condor, end="\n")
        print ("+LongRunningJob = true", file=condor, end="\n")
        print ("Rank = -LoadAvg", file=condor, end="\n")
        print ("Universe = vanilla", file=condor, end="\n")
        print ("Executable = " + m2s_path + "/bin/m2s", file=condor,
        ↪  end="\n")
        print ("Environment = LD_LIBRARY_PATH=$ENV(LD_LIBRARY_PATH)",
        ↪  file=condor, end="\n\n")


if len(argv) < 4:
        print ("Usage: " + argv[0] + " <ctx_dir> <m2s_root_path>
        ↪  <condor_fname> <extra_args>", end="\n")
        exit()

script, path, m2s_root_path, out_file, extra_args = argv

src_path =  m2s_root_path + "src/"
sim_type = "--x86-sim detailed "
cpu_config = "--x86-config " + src_path +
↪  "configuraciones-sample/baseline_DDR4/4vias/cpuconfig_baseline "
mem_config = "--mem-config " + src_path +
↪  "configuraciones-sample/baseline_DDR4/4vias/cacheconf_4cores_512KB
↪  _2MB_baseline "
net_config = "--net-config " + src_path +
↪  "configuraciones-sample/baseline_DDR4/4vias/net_4core_baseline "
max_inst = "--x86-min-inst-per-ctx " + str(1000000000) + " "
ep_length = "--epoch-length " + str(12000000) + " "
```

```
rep_dir = "--reports-dir
 ↪  /nfs/alumnos/hutarsan/reports/baseline_DDR4/4vias/"

i = 0

create_file_header(out_file, m2s_root_path)

for name in files(path):
        if not name.find("ctxconfig.") < 0:
                print ("File: " + name, end="\n")
                ctx_config = "--ctx-config " + path + name + " "
                condor = open(out_file, "a")
                print ("Arguments = " + sim_type + cpu_config + mem_config
                 ↪  + net_config + ctx_config + max_inst + ep_length +
                 ↪  rep_dir + name.split(".")[1] + extra_args,
                 ↪  file=condor, end="\n\n")
                print ("Log =
                 ↪  /nfs/alumnos/hutarsan/outputs/baseline_DDR4/4vias/" +
                 ↪  name.split(".")[1] + ".log", file=condor, end="\n")
                print ("Output =
                 ↪  /nfs/alumnos/hutarsan/outputs/baseline_DDR4/4vias/" +
                 ↪  name.split(".")[1] + ".out", file=condor, end="\n")
                print ("Error =
                 ↪  /nfs/alumnos/hutarsan/outputs/baseline_DDR4/4vias/" +
                 ↪  name.split(".")[1]  + ".err", file=condor, end="\n")

                print ("Queue", file=condor, end="\n\n")
                i = i +1
```

A.3 missHitsAmontonadoPercnt.py

Este *script* genera una matriz con el porcentaje de desplazamientos dividido en los tramos 0, 1-3, 4-8 y 9-16 de la cache de datos L1. Además, divide este porcentaje entre aciertos y fallos. Recibe como parámetro la ubicación de los resultados de los benchmarks, el tamaño del conjunto y la cantidad de cabezales.

```
import pandas as pd
from sys import argv
import re
import createDics as cd


if len(argv) < 2:
    print("Usage: <Script> <Benchmark_path> <Set size> <Cabezales>\n")
    exit()
benchmark = argv[1].split("/")
print(argv)
longitud = len(benchmark)
benchmark = benchmark[longitud - 2]
rango =int(argv[3]) -1
```

```
aux = int(argv[4])
cabezales = int(aux)
# Var declaration
cond = True
i = 0
via = 0
ciclos_p = 0
sets = 0
num = 0
sets = 0
max = 0
vias = re.findall(r"(\d+)vias", argv[1])
vias = vias[0]
vias = int(vias)
total = 0
names = []
benchs = cd.create_dic(argv[1])
# df = pd.DataFrame(argv[1]+"interval_reports/mod/dl1-0.intrep.csv")
accesos = 0
hits = 0
misses = 0

names = []
t1 = []
t2 = []
t3 = []
t4 = []
t5 = []
miss1 = []
miss2 = []
miss3 = []
miss4 = []
miss5 = []
totalx = []
dic = dict()
acumulada = 0
#Variables para la media con 4 tramos
#mt0 = 0;
 #mt1 = 0;
#mt2 = 0;
#mt3 = 0;

# npy.zeros((n,n))
# Main

for benchmark,arc_name in benchs.items():
    #print(benchmark)
    #rint("\n")
    for arc_n,df in arc_name.items():
        df = pd.read_csv(argv[1]+benchmark+
        ↪  "/interval_reports/mod/dl1-0.intrep.csv")
    #Geting the name df
```

```
lrow = len(df.index) -1
names.append(benchmark)
total = 0
acumulada = 0
tope = int((rango+1)/cabezales)
for x in range(vias):
        for y in range(0,tope):
                total = total + df.at[lrow,
                 ↪  "dl1-0-c4t1-via-"+str(x)+"-ciclos-penalizacion-"
                 ↪  +str(y)]

totalx.append(total)
# print("[DEBUG] "+str(benchmark)+ ": " +str(media_t))
for x in range(0,vias):
        hits = hits+  df.at[lrow,
         ↪  "dl1-0-c4t1-via-"+str(x)+"-ciclos-penalizacion hits-0"]
        misses= misses + df.at[lrow,
         ↪  "dl1-0-c4t1-via-"+str(x)+"-ciclos-penalizacion misses-0"]
t1.append((hits/total)*100)
miss1.append((misses/total)*100)
acumulada+= (hits/total)*100 + (misses/total)*100
hits = 0
misses = 0

for x in range(0,vias):
        for y in range(1,4):
            hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
             ↪  "-ciclos-penalizacion hits-"+str(y)]
            misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
             ↪  "-ciclos-penalizacion misses-"+str(y)]
t2.append((hits/total)*100)
miss2.append((misses/total)*100)
acumulada += (hits / total) * 100 + (misses / total) * 100
hits = 0
misses = 0

for x in range(0,vias):
        for y in range(4,9):
            hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
             ↪  "-ciclos-penalizacion hits-"+str(y)]
            misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
             ↪  "-ciclos-penalizacion misses-"+str(y)]
t3.append((hits/total)*100)
miss3.append((misses/total)*100)
acumulada += (hits / total) * 100 + (misses / total) * 100
hits = 0
misses = 0

for x in range(0,vias):
        for y in range(9,17):
            hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
             ↪  "-ciclos-penalizacion hits-"+str(y)]
```

```
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion misses-"+str(y)]
    t4.append((hits/total)*100)
    miss4.append((misses/total)*100)
    acumulada += (hits / total) * 100 + (misses / total) * 100
    hits = 0
    misses = 0

    for x in range(0,vias):
            for y in range(17,tope):
                hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion hits-"+str(y)]
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion misses-"+str(y)]
    t5.append((hits/total)*100)
    miss5.append((misses/total)*100)
    acumulada += (hits / total) * 100 + (misses / total) * 100
    hits = 0
    misses = 0

#names.append("Media global")
#t1.append((mt0/(mt0+mt1+mt2+mt3))*100)
#t2.append((mt1/(mt0+mt1+mt2+mt3))*100)
#t3.append((mt2/(mt0+mt1+mt2+mt3))*100)
#t4.append((mt3/(mt0+mt1+mt2+mt3))*100)



#[DEBUG]
#print(len(t1))
#print(len(t2))
#print(len(t3))
#print(len(t4))




dic["Benchmark"] = names
dic["Tramo 0 hits %"] = t1
dic["Tramo 1-3 hits %"] = t2
dic["Tramo 4-8 hits %"] = t3
dic["Tramo 9-16 hits %"] = t4
dic["Tramo +16 hits %"] = t5
dic["Tramo 0 misses %"] = miss1
dic["Tramo 1-3 misses %"] = miss2
dic["Tramo 4-8 misses %"] = miss3
dic["Tramo 9-16 misses %"] = miss4
dic["Tramo +16 misses %"] = miss5
dic["Acumulada"] = acumulada


dff = pd.DataFrame(dic)
```

```
print(dff)
dff.to_csv(argv[2], sep= ",",index=False )
```

A.4 missHitsAmontonadoAbsoluto.py

Este *script* genera una matriz con el porcentaje de desplazamientos dividido en los tramos 0, 1-3, 4-8 y 9-16 de la cache de datos L1. Además, divide estos valores entre número absoluto de aciertos y fallos. Recibe como parámetro la ubicación de los resultados de los benchmarks, el tamaño del conjunto y la cantidad de cabezales.

```
import pandas as pd
from sys import argv
import re
import createDics as cd


if len(argv) < 2:
    print("Usage: <Script> <Benchmark_path> <Set size> <Headers>\n")
    exit()
benchmark = argv[1].split("/")
print(argv)
longitud = len(benchmark)
benchmark = benchmark[longitud - 2]
rango =int(argv[3]) -1
cabezales = int(argv[4])
# Var declaration
cond = True
i = 0
via = 0
ciclos_p = 0
sets = 0
num = 0
sets = 0
max = 0
vias = re.findall(r"(\d+)vias", argv[1])
vias = vias[0]
vias = int(vias)
total = 0
names = []
benchs = cd.create_dic(argv[1])
# df = pd.DataFrame(argv[1]+"interval_reports/mod/dl1-0.intrep.csv")
accesos = 0
hits = 0
misses = 0

names = []
t1 = []
t2 = []
t3 = []
t4 = []
t5 = []
```

```
miss1 = []
miss2 = []
miss3 = []
miss4 = []
miss5 = []
totalx = []
dic = dict()
#Variables para la media con 4 tramos
#mt0 = 0;
 #mt1 = 0;
#mt2 = 0;
#mt3 = 0;

# npy.zeros((n,n))
# Main

for benchmark,arc_name in benchs.items():
    #print(benchmark)
    #rint("\n")
    for arc_n,df in arc_name.items():
        df = pd.read_csv(argv[1]+benchmark+
        ↪  "/interval_reports/mod/dl1-0.intrep.csv")
    lrow = len(df.index) -1
    names.append(benchmark)
    total = 0
    tope = int((rango + 1) / cabezales)
    #print(vias)
    #print(benchmark)
    for x in range(0,vias):
            for y in range(0,tope):
                    total = total + df.at[lrow, "dl1-0-c4t1-via-"+str(x)+
                    ↪  "-ciclos-penalizacion-"+str(y)]

    totalx.append(total)
    # print("[DEBUG] "+str(benchmark)+ ": " +str(media_t))
    for x in range(0,vias):
            hits = hits+  df.at[lrow, "dl1-0-c4t1-via-"+str(x)+
            ↪  "-ciclos-penalizacion hits-0"]
            misses= misses + df.at[lrow, "dl1-0-c4t1-via-"+str(x)+
            ↪  "-ciclos-penalizacion misses-0"]
    t1.append(hits)
    miss1.append(misses)
    hits = 0
    misses = 0

    for x in range(0,vias):
            for y in range(1,4):
                hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                ↪  "-ciclos-penalizacion hits-"+str(y)]
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                ↪  "-ciclos-penalizacion misses-"+str(y)]
    t2.append(hits)
```

```
    miss2.append(misses)
    hits = 0
    misses = 0

    for x in range(0,vias):
            for y in range(4,9):
                hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion hits-"+str(y)]
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion misses-"+str(y)]
    t3.append(hits)
    miss3.append(misses)
    hits = 0
    misses = 0

    for x in range(0,vias):
            for y in range(9,17):
                hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion hits-"+str(y)]
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion misses-"+str(y)]
    t4.append(hits )
    miss4.append(misses )
    hits = 0
    misses = 0
    for x in range(0,vias):
            for y in range(17,tope):
                hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion hits-"+str(y)]
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion misses-"+str(y)]
    t5.append(hits )
    miss5.append(misses)
    hits = 0
    misses = 0

#names.append("Media global")
#t1.append((mt0/(mt0+mt1+mt2+mt3))*100)
#t2.append((mt1/(mt0+mt1+mt2+mt3))*100)
#t3.append((mt2/(mt0+mt1+mt2+mt3))*100)
#t4.append((mt3/(mt0+mt1+mt2+mt3))*100)



#[DEBUG]
#print(len(t1))
#print(len(t2))
#print(len(t3))
#print(len(t4))
```

```
dic["Benchmark"] = names
dic["Tramo 0 hits"] = t1
dic["Tramo 1-3 hits"] = t2
dic["Tramo 4-8 hits"] = t3
dic["Tramo 9-16 hits"] = t4
dic["Tramo +16 hits"] = t5
dic["Tramo 0 misses"] = miss1
dic["Tramo 1-3 misses"] = miss2
dic["Tramo 4-8 misses"] = miss3
dic["Tramo 9-16 misses"] = miss4
dic["Tramo +16 misses"] = miss5
dic["Total"] = totalx


dff = pd.DataFrame(dic)
print(tope)
print(dff)
dff.to_csv(argv[2], sep= ",",index=False )
```

A.5 mpkisToCsv.py

Este ejecutable obtiene los fallos de predicción por kilo-instrucción. Divide el MPKI entre MPKI de cache L1, L2 y L3 y lo genera a partir de las estructuras matriciales de datos generadas en createDics.py

```
import createDics as cd
import pandas as pd
from sys import argv

if len(argv) < 3:
    print("Usage: <Script> <ReportsDir> <nameCSV>\n")
    exit()

dic = dict()
benchs = cd.create_dic(argv[1])
aux1 = []
aux2 = []
aux3=[]
aux4= []


num = 0
for benchmark,arc_name in benchs.items():
    #print(benchmark)
    #rint("\n")
    for arc_n,df in arc_name.items():
        # print(benchmark)
        # print(df["pid100-l1-misses-int"])
        #L1 MISSES
        misses= sum(df["pid100-l1-misses-int"])
```

```
        #print(benchmark + " misses-> " + str(misses))
        lrow = len(df.index) -1
        inst = df.at[lrow, "pid100-uinsts"]
        #print(benchmark + "inst-> " + str(inst))
        aux1.append(benchmark)
        aux2.append(misses/(inst/1000))
        #print(benchmark + " MPKI_L1-> " + str(inst))
        #print(misses)
        #print(inst) #bien
        #L2 MISSES
        misses = sum(df["pid100-l2-misses-int"])
        inst = df.at[lrow, "pid100-uinsts"]
        aux3.append(1000 * misses / inst)
        #L3 MISSES
        misses = sum(df["pid100-l3-misses-int"])
        inst = df.at[lrow, "pid100-uinsts"]
        aux4.append(1000 * misses / inst)

dic["Benchmark"] = aux1
dic["mpki_L1"] = aux2
dic["mpki_L2"] = aux3
dic["mpki_L3"] = aux4



dff = pd.DataFrame(dic)
print(dff)
dff.to_csv(argv[2], sep= ",",index=False )
```

A.6 porcentajesHitsMissesL1.py

Este ejecutable devuelve el porcentaje de fallos y aciertos totales en la cache de datos L1. También devuelve el número de accesos a esta cache y el número absoluto de fallos y aciertos.

```
import createDics as cd
import pandas as pd
from sys import argv

if len(argv) < 3:
    print("Usage: <Script> <ReportsDir> <nameCSV>\n")
    exit()



dic = dict()
accesos_L1= 0
accesos_totales=0


benchs = cd.create_dic_mod(argv[1])
```

```
aux1= []
aux2= []
aux3= []
aux4= []
aux5=[]
aux6=[]


for benchmark,arc_name in benchs.items():
    for arc_n,df in arc_name.items():
        #MRU_hits_totales.append(MRU_hits)
        accesos_L1 = sum(df["dl1-0-hits-int"]) +
        ↪  sum(df["dl1-0-misses-int"])
        aciertos_L1 = sum(df["dl1-0-hits-int"])
        fallos_L1 = sum(df["dl1-0-misses-int"])
        lrow = len(df.index) - 1


        aux1.append(benchmark)
        aux2.append(accesos_L1)
        aux3.append(aciertos_L1)
        aux4.append(fallos_L1)
        aux5.append((aciertos_L1/accesos_L1)*100)
        aux6.append((fallos_L1/accesos_L1)*100)



dic[" Benchmarks"] = aux1
dic["Accesos L1-d"] = aux2
dic["Aciertos L1-d"] = aux3
dic[" Fallos L1-d"] = aux4
dic["Porcentaje aciertos L1-d"] = aux5
dic["Porcentaje fallos L1-d"] = aux6

dff = pd.DataFrame(dic)
print(dff)
dff.to_csv(argv[2], sep= ",",index=False )
```

A.7 ejecucionAmontonados.py

Este *script* utiliza missHitsAmontonadoAbsoluto.py y missHitsAmontonadoAbsolutoPercnt.py para automatizar la obtención de datos. Devuelve los resultados para 32 KB con 4 cabezales, 64 KB con 8 cabezales, 128 KB con 16 cabezales y 256 KB con 32 cabezales con 4 y 8 vías. Estos resultados por tramos muestran tanto la representación absoluta como la representación porcentual.

```
import sys

print("EMPEZANDO AMONTONADO ABSOLUTO")
script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
```

```
sys.argv = ["miss_hits_amontonado_absoluto.py",
 ↪  "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/4vias/256kb_32cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /256kb_32cap_amontonado_4v.csv","1024","32" ]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
 ↪  "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/4vias/128kb_16cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /128kb_16cap_amontonado_4v.csv","512","16"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
 ↪  "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/4vias/64kb_8cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /64kb_8cap_amontonado_4v.csv","256","8"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
 ↪  "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/4vias/32kb_4cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /32kb_4cap_amontonado_4v.csv","128","4"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
 ↪  "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/8vias/256kb_32cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /256kb_32cap_amontonado_8v.csv","512","32"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()
```

```
script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
 ↪  "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/8vias/128kb_16cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /128kb_16cap_amontonado_8v.csv","256","16"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
 ↪  "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/8vias/64kb_8cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /64kb_8cap_amontonado_8v.csv","128","8"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
 ↪  "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/8vias/32kb_4cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /32kb_4cap_amontonado_8v.csv","64","4"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()

print("ACABADO AMONTONADO ABSOLUTO")


script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/8vias/32kb_4cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /32kb_4cap_amontonado_8v_%.csv","64","4"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/8vias/64kb_8cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /64kb_8cap_amontonado_8v_%.csv","128","8"]
exec(miss_hits_porcentaje_amontonado)
```

```
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/8vias/128kb_16cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /128kb_16cap_amontonado_8v_%.csv","256","16"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/8vias/256kb_32cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /256kb_32cap_amontonado_8v_%.csv","512","32"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/4vias/256kb_32cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /256kb_32cap_amontonado_4v_%.csv","1024","32"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/4vias/128kb_16cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /128kb_16cap_amontonado_4v_%.csv","512","16"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/4vias/64kb_8cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /64kb_8cap_amontonado_4v_%.csv","256","8"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
```

```
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
 ↪  racetrack/reports/ent_comparativa/DL/4vias/32kb_4cap/",
 ↪  "/home/hutarsan/Documentos/racetrack/scripts
 ↪  /32kb_4cap_amontonado_4v_%.csv","128","4"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()


print("ACABADO AMONTONADO %")
```

A.8 WU_csv.py

Este ejecutable sirve para probar el WarmUp. Sirvió para implementarlo en el simulador y comprobar que no fallaba nada, ya que este no ofrece soporte para esta operación de manera nativa.

```
import createDics as cd
import pandas as pd
from sys import argv
import csv

if len(argv) < 2:
    print("Usage: <Script> <ReportsDir> \n")
    exit()

dic = dict()
benchs = cd.create_dic(argv[1])

WU_cycles = 7
for benchmark,arc_name in benchs.items():
    #print(benchmark)
    #rint("\n")
    for arc_n,df in arc_name.items():
        for x in range(0,WU_cycles):
            df = df.drop([x])
    #df.to_csv(r'Path where you want to store the exported CSV
    ↪  file\pid100.intrep.csv', sep='\t')
    df.to_csv(argv[1]+benchmark+
    "/interval_reports/x86_ctx/pid100.intrep.csv", sep= ",",index=False )
print("WU done!")
```

A.9 agruparUnificadaTC

Este script sirva para extraer en un archivo csv todos los datos y agruparlos para aquellas simulaciones que utilizan TapeC, ya que su estructura es ligeramente diferente.

```
import createDics as cd
import pandas as pd
from sys import argv
```

```
if len(argv) < 3:
    print("Usage: <Script> <ReportsDir> <nameCSV> <Ciclo max
     ↪  penalizacion>\n")
    exit()

dic = dict()
benchs = cd.create_dic_agrupado(argv[1])
print(argv[1])
aux1 = []
aux2 = []
pen_max = int(argv[3])
bench = ""
aux = 0

for benchmark,arc_name in benchs.items():
    for letra in benchmark:
        if(letra != "."):
            bench = bench+letra
        else:
            break

    if aux == 0:
        tops = list(arc_name.columns.values)
        tops.insert(0, "Benchmarks")
        res = arc_name.copy()
        res.insert(0, "Benchmark", bench)
        aux = 1
    else:
        arc_name.insert(0, "Benchmark", bench)
        res = res.append(arc_name)

    aux2.append(bench)
    bench = ""



print(res)
res.to_csv(argv[2], sep= ",",index=False )
```