The MPsim Simulation Tool

Carmelo Acosta †, Francisco J. Cazorla ⋆, Alex Ramirez †⋆, Mateo Valero †⋆

† Universitat Politecnica de Catalunya
HiPEAC European Network of Excellence
Barcelona, Spain
{cacosta,aramirez,mateo}@ac.upc.edu

⋆ Barcelona Supercomputing Center
Barcelona, Spain
{francisco.cazorla,alex.ramirez,mateo.valero}@bsc.es

Abstract

In order to evaluate novel ideas, computer architects require simulation tools which model a target architecture. According to the specific accuracy requirements we find very specific simulators, which model a single architecture with high accuracy and computational cost, like the ones typically used in the industry, and general purpose simulators with a less accurate model but requiring less computational cost, like the ones typically used in the academia. Focusing on the latter, flexible simulation tools allow evaluating a wide range of system configuration, requiring low effort to evaluate novel ideas. Consequently, the flexibility is a main characteristic to be considered by computer architects when selecting a general-purpose simulation tool.

In this paper it is presented a highly-flexible general-purpose simulation tool : the MPsim. *It allows simulating a wide range of processor types, both single core (Superscalar, SMT) and multi core (CMP, CMP+SMT), both homogeneous and heterogeneous configurations. It is put special emphasis on the simulator flexibility and how it is obtained. The simulation results included indicate that high-flexibility may be obtained without hardly compromising the computational cost in a general purpose simulator.*

1 Introduction

Computer Architecture has experienced great advances in the last decades. Thus, we have witnessed the raise of Superscalars, Simultaneous Multithreading (SMT) and on-chip Multiprocessors (CMP) among others. All these novel ideas had to be evaluated prior to their usage in order to measure their benefits and potential. To perform this evaluation, computer architects require simulation tools which model the corresponding idea and allow simulating its execution results, employing a set of benchmarks. The accuracy of the model employed is in tune with the research requirements. Thus, while in industry computer architects are highly constrained to an specific product, requiring a highly accurate model, in the academia computer architects generally focus on more long term and less specific research topics. Obviously, the computational cost of the model employed is directly proportional to its accuracy. Consequently, the research in the academia generally employs general-purpose simulation tools, closer to their research interests and computational possibilities.

Among the general-purpose simulation tools typically employed in the academia during the last decade we find Simplescalar [4] and SMTsim [9] simulators. The Simplescalar models a single-core Superscalar processor with 5 pipeline stages while the SMTsim models a single-core Superscalar/SMT processor with 8 pipeline stages. On top of both simulators, several branch predictors and instruction fetch policies, so as new proposals, may be added. Regarding the Memory Subsystem, both simulators model two cache levels (optionally up to the third cache level), with a single Instruction Cache, Data Cache, ITLB, DTLB, L2 Cache. However, while the Simplescalar has a very simple memory model, in which each memory access is deterministically resolved, the SMTsim non-deterministically manages the memory accesses by means of an event queue, which cronologically stores all memory requests.

In this paper we present the *MPsim* simulator, a highly-flexible simulator based on SMTsim. It allows simulating a wide range of processor types both single core (Superscalar, SMT) and multi core (CMP, CMP+SMT), both homogeneous and heterogeneous configurations; so as providing a complete set of simulation alternatives. It is put special emphasis on the simulator flexibility and how it is obtained. The *MPsim Parameter Interface* allows to easily declare complex system configurations without needing to recompile the simulator source code. Both core-specific and memory subsystem configuration parameters may be gathered into parameter files comprising reusable configuration repositories. The simulation results included indicate that high-flexibility may be obtained without hardly compromising the computational cost in a general-purpose simulator.

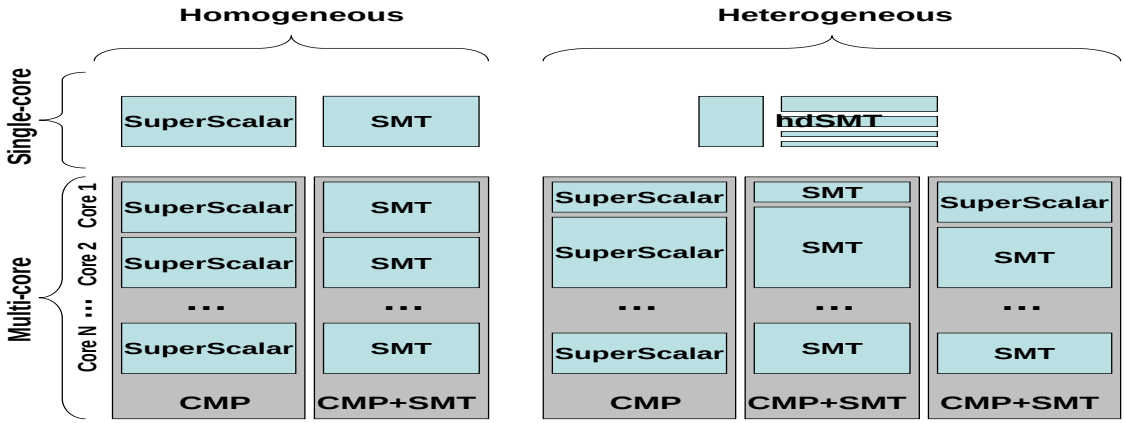

Figure 1. MPsim Processor Types.

2 MPsim overview

The *MPsim* is a cycle-accurate simulation tool based on the SMTsim simulator. Its design focus on the simulator *flexibility* and *functionality*, striving at the same time to involve the least computational cost possible. The simulator's *flexibility* does not refer only to the amount of simulation alternatives provided to the user but also to the configuration easiness and adaptability to future modifications. The *MPsim Parameter Interface* ease the declaration of complex simulation configurations. It allows to maintain configuration file repositories that may be reused in different simulations without needing to recompile the simulator's source code.

The *MPsim* allows simulating a wide range of processor types both single core (Superscalar, SMT) and multi core (CMP, CMP+SMT). By using the *NUM_CORES* parameter it may be specified the number of cores in the simulated system. All the remainder core-specific parameters will carry the suffix *_Px*, where *x* stands for the core number (e.g., *IFETCH_POLICY_P1 ICOUNT* declares that the core number 1 use the ICOUNT IFetch Policy). These suffixes allow to individually configure each core, making possible heterogeneous[1] system configurations. Thus, although each simulated system core is comprised of at least 8 pipeline stages, the specific pipeline depth may be individually declared for each constituent system core. To configure entire systems, both homogeneous and heterogeneous, each simulated core may be individually declared by using both the command line or configuration files. The *MPsim Parameter Interface* allows passing text files comprising all core-specific parameters. These configuration files may be reused in multiple declarations as simulation inputs to configure each simulated system core (e.g., *-pf_P1 POWER5* specifies the file POWER5 to configure the core number 1). Figure 1 shows the processor types that can be simulated using *MPsim*.

In order to reduce computational costs, the *MPsim* provides a trace-driven[2] front-end. Although trace-driven, the *MPsim* also permits simulating the impact of executing along wrong paths on the branch predictor and the instruction cache by having a separate basic block dictionary in which information of all static instructions is contained. The *MPsim* input traces are collected from the most representative 300 million instruction segment of each input benchmark, following the idea presented in [7]. Each program is compiled with the *–O2 –non_shared* options using DEC Alpha AXP-21264 C/C++ compiler and executed using the reference input set. These input traces can be indistinctly read from little-endian/big-endian machines, since the *MPsim* automatically detects the machine characteristics and read data accordingly.

The *MPsim* functionality, provided to the user by means of its flexible *Parameter Interface*, includes a long list of simulation alternatives. Regarding simulation itself, the *MPsim* provides simulation forwarding, numerous simulation statistics and histograms, so as six different simulation finalization modes. Regarding computer architecture alternatives, the *MPsim* provides a set of branch predictors and instruction fetch policies from which select the desired one, thread migration between cores, so as multibanked multiported caches. All these functionality items may be easily activated/deactivated by the user, according to her needs, using the appropriate parameter for each case (e.g., *STATS_INTERVAL 0* deactivates the intermediate IPC statistics). As a matter of example, by means of the *STATS_INTERVAL*, *MAX_NUM_INTERVALS*, *STATS_FORWARDING* and *MAX_NUM_STATS_FILES* pa-

[1] The term heterogeneous refer to different amount of processor resources, like instruction queue entries and number of registers.

[2] The execution-driven functionality is currently being developed.

rameters it may be obtained intermediate simulation IPC statistics (interval IPC, IPC variability and in-flight L1 misses) in separate dump files.

The *MPsim* also allows some extent of clustering when defininining the system to be modeled. Thus, the *SHARED_FETCH_UNIT* and *SHARED_REGISTER_FILE* parameters allow sharing a single Fetch Unit and Register File respectively, among all defined system cores. Since a single Fetch Unit may be shared among multiple cores, we indistinctly refer to pipeline/core in the remainder of the paper. However, recall that the only difference is the value of the *SHARED_FETCH_UNIT* (i.e., pipeline = true, core = false). As a matter of example, in an hdSMT [2] processor (see Figure 1) both the Fetch Unit and the Register File are shared among all constituent pipelines.

3 Parameter Interface

In order to provide high-flexibility the *MPsim* simulator includes a *lexical analyzer*, yielding a versatile *Parameter Interface*. It scans the simulator call creating pairs of parameter name and value, which are inserted in an inner *Parameter Data Base*. There is not a fixed parameter declaration order, with the only assumption that every argument which begins with a dash is considered a parameter name and the immediate following argument is considered its value (e.g., the simulator call *mpsim -arg1 arg2* includes the parameter arg1 with value arg2). Whenever a single parameter name is declared more than once, the value in the *Parameter Data Base* corresponds to the last parameter declaration. The *Parameter Interface Library* includes functions to adquire each parameter from the *Parameter Data Base* to the simulator inner structures. This way, the addition of new functionality benefits from an easy way to adquire configuration parameters.

The special parameter name *parms_file* (or simply *pf*) is reserved to indicate a configuration parameter file, with the parameter value indicating the file path. The use of parameter files permits to declare an unlimited number of parameters, allowing more complex simulation configurations. Additionally, by using parameter files, that may also include comments (using #), it is possible to keep *configuration file repositories*. Although the parameter files may include any sort of parameters, the main repositories used are comprised of *cores*, *machines* and *memory subsystems* declarations. In order to ease multicore configurations and repositories maintenance, it may be added the suffix *_Px* to a parameter file name declaration, with *x* identifying a given core. This suffix indicates that all the parameters included in the corresponding file are related to the specified *x* core (e.g., *-pf_P0 file1* declares the file *file1* as input to configure the first core in the simulated system). The *Parameter Interface* then automatically adds this suffix to each parameter name included in the file. Thus, a single core file may be used to configure multiple cores in a multicore configuration; or in different simulation calls.

Once scanned the whole simulator call, the resulting *Parameter Data Base*, that comprises all declared pairs of parameter name and value, is used in the subsequent *Simulator Initialization Phase*. During this phase the content of the *Parameter Data Base* is used to initialize the corresponding simulator structures and variables. Any sort of parameters may be requested by the simulator developer by using the *NeedValue* and *GiveValue* functions from the *Parameter Interface Library*. Whenever a parameter is compulsory, and does not admit a deffault value, it is used the *NeedValue*, which automatically stops the initialization phase and prompts an error message in absence of the specified parameter. Otherwise, it is used the *GiveValue* function.

Figure 2 illustrates the high-flexibility of the *MPsim Parameter Interface*. In the example, 3 configuration files stored in the simulator's repositories are used to configure a Cell-like processor with a simple simulator call. Given the files *PPE* and *SPE*, that include all core-specific configuration parameters for Cell PPE-like and SPE-like cores respectively, and the file *Cell*, that include all Memory Subsystem related parameters and relations for a Cell-like configuration, the simulator call shown in Figure 2 is enough to configure a Cell-like simulation[3].

The *Lexical Analyzer*, included in the *MPsim Parameter Interface*, scans the whole simulator call shown in Figure 2 automatically accessing to the corresponding files in the repositories. The *Lexical Analyzer* uses the suffix information included in the simulator call (i.e., *_Px* in the *-pf_Px* argument, with *x* indicating the specific core) to create the corresponding pairs of parameter name and value that are inserted into the *Parameter Data Base*. Thus, although there is a single *MAXTHREADS* parameter declaration in PPE and SPE files stored in the cores repository (see Figure 2), multiple *MAXTHREADS* pairs are inserted in the *Parameter Data Base*, one per each of the 9 delcared cores. Once the whole simulator call is scanned, including the parameter files, the subsequent *Simulator Initialization Phase* uses the resulting *Parameter Data Base* and the *Interface Library* functions to set up the simulator inner structures and prepare the subsequent simulation. Thus, during the multipipeline environment initialization (i.e., *init_multipipeline*, see Figure 2) it is used the function *NeedValue* to initialize the simulator from the information contained in the *Parameter Data Base*, modeling an heterogeneous multi-core processor comprised of 9 cores (i.e., *NUMCORES*), each one containing *MAXTHREADS* hardware contexts (i.e., a dual-thread PPE and 8 single-thread SPEs). After the initialization phase, the simulation begins.

[3] Although not included in the simulator call for simplicity, it should be also specified the workload to simulate.

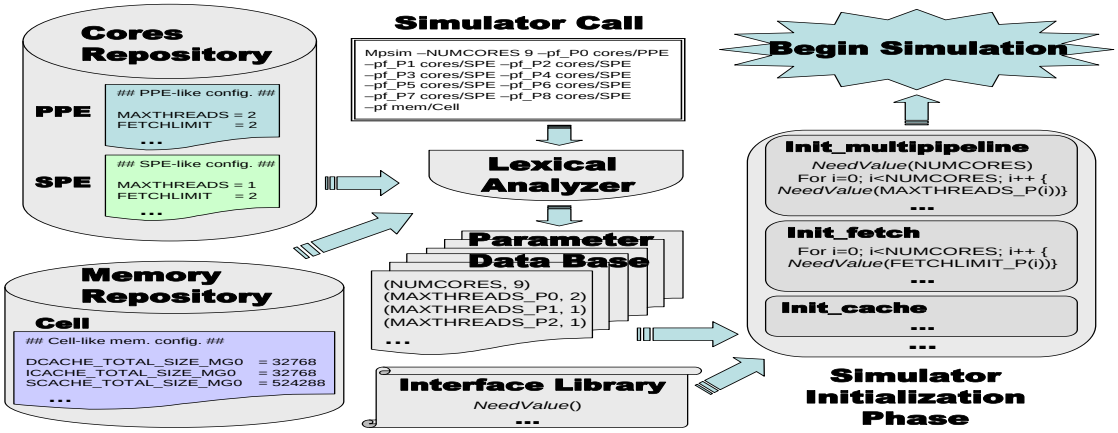

Figure 2. Parameter Interface Example for a Cell-like configuration.

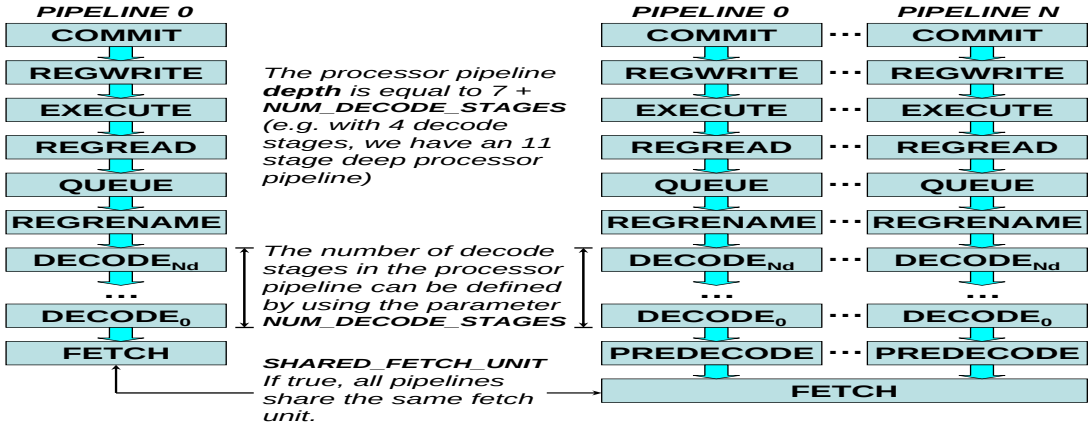

Figure 3. MPsim Processor Pipeline Stages.

4 The Pipeline

The *MPsim* is a cycle-accurate simulator in which each simulated system core is comprised of at least 8 pipeline stages, as shown in Figure 3. However, each system core may be defined with a different pipeline depth, adding idle pipeline stages in between *Decode* and *RegRename* stages. As a matter of example, to specify an 11-stage execution pipeline in any of the declared cores it is set the parameter *NUM_DECODE_STAGES 4*.

In case of sharing the Fetch Unit among all pipelines (see Section 2) a new pipeline stage, called *Predecode*, is automatically added by the *MPsim* to each pipeline. The *Predecode* stage works as a buffer (with user-definable capacity using the *PREDECODE_QUEUE_SIZE* parameter) between the shared Fetch Engine and the decode stage of each constituent pipeline, which may have a different pipeline width. As a matter of example, in a given cycle an 8-wide shared Fetch Engine passes 8 instructions to a 4-wide pipeline; 4 instructions passes to that pipeline decode stage while another 4 instructions are buffered in *Predecode* until the next simulation cycle.

The pipeline resources and implemented policies may be easily declared using the *MPsim Parameter Interface* (see Section 3). Each Fetch Unit declared in a simultaneous multithreaded system (i.e., the shared Fetch Unit in an hdSMT processor or each Fetch Unit in a CMP+SMT processor) may be configured with a different Instruction Fetch Policy, which determines from which thread/s to fetch instructions each cycle. To define the IFetch Policy used by each Fetch Unit we employ the *IFETCH_POLICY_Px* parameter, where *x* corresponds to the processor pipeline number. The user may select any from Round Robin [10], ICOUNT [10], STALL [8], FLUSH [8], and FLUSH_PLUS_PLUS [3]. In a similar way, each Fetch Unit declared in a system may be configured with a different branch predictor, using the *predictor_Px* parameter, where x corresponds to the processor pipeline number. In this case, the user may chose any from GSHARE [6], PERCEPTRON [5] and PERFECT predictor.

4.1 Thread Migration

Multicore configurations can be simulated in either STATIC or DYNAMIC fashion, using the *THREADS_MIGRATION* parameter. STATIC simulations assume no thread migrations, from core to core, during the whole simulation. DYNAMIC simulations may experience thread migrations according to the specified *MIGRATION_INTERVAL* parameter value (measured in simulated cycles). The assignment of all simulated threads to any of the defined cores is specified by the *FIRST_T2P_ASSIG_POLICY* parameter. It may be chosed from *NRR* (Naive Round Robin) and *CUSTOM*, using the *ASSIG_TH_X_P* parameter in the latter case to specify each assignment (e.g., *ASSIG_TH_1_P 0* assigns the thread 1 to the core 0).

In DYNAMIC simulations, the thread migrations are triggered according to the specified *MIGRATION_HEURISTIC* parameter value. Among the available migration heuristics it can be chosed the *BEST_DYNAMIC*. In a *BEST_DYNAMIC* simulation every *MIGRATION_INTERVAL* simulated cycles all possible thread migrations are considered by the simulator, chosing for each interval the one which yields the highest throughput.

5 The Memory Subsystem

The *MPsim* Memory Subsystem inherits the SMTsim's foundations, having an event queue to manage all memory requests in a non-deterministic fashion. Whenever a memory request experience an L1 Cache miss it is inserted a memory request in the queue, arranged by cronological request time (in simulated cycles). According to the specific system configuration, memory hit/miss, and contention, the memory request may have to traverse the L2 Cache, the L3 Cache and the L1-L2 intercomunication bus, so as accessing to a TLB. The memory request queue is regularly accessed by the simulator, triggering each request in the corresponding simulation cycle. As described in Section 5.2, the *MPsim* structures this memory event queue into two layers for multicore configurations with an L2 Access Arbiter implemented.

Unlike SMTsim, with a *fixed* Memory Subsystem definition, the *MPsim* provides the user a *fully-flexible* Memory Subsystem. Thus, it may be configured a Memory Subsystem comprised of any number of memory components (DTLBs, ITLBs, DCaches, ICaches, L1-L2 Buses, L2 Caches and L3 Caches) so as relations, between memory components and execution pipelines. The *MPsim Parameter Interface* allows to specify the desired number of components[4] by using the *NUM_L3_CACHES*, *NUM_L2_CACHES*, *NUM_BUSES*, *NUM_ICACHES*, *NUM_DCACHES*, *NUM_ITLBS*, *NUM_DTLBS* parameters. Once declared, the user may configure each of the components' characteristics individually, using command line parameters or parameter files (e.g., a DTLB is configured with *DTLBPENALTY*, *DPGSIZE* and *DTLB_SIZE* parameters). As a consequence, not all components of the same type must have the same characteristics, allowing heterogeneous memory configurations. To ease this configuration, each memory component is associated to a single *Memory Group (MG)*, as shown in Figure 4 (e.g., I_0\$, $ITLB_0$, D_0\$, $DTLB_0$, BUS_0, $L2_0$ and $L3_0$ belong to the first *Memory Group*). Thus, when specifying a component characteristic we add the suffix *_MGx*, where *x* stands for the *Memory Group*, to refer to a particular memory component (e.g., the *DTLB_SIZE_MG0* parameter value specifies the size of the $DTLB_0$, belonging to the first *Memory Group*).

The *MPsim* Memory Subsystem does not assume any implicit relation between any two components[5], allowing the user to explicitly declare the desired relations. The *Memory Groups*, used to univocally refer to each memory component declared in the system, do not imply real memory component relations (e.i., D_0\$ does not necessarily use BUS_0 to communicate with the second level of cache). To specify the desired memory component relations the *MPsim Parameter Interface* provides a simple *Regular Expression Grammar (REG)*, shown in Figure 5. This *REG*, implemented as part of the *Lexical Analyzer* included in the *MPsim Parameter Interface* (See Section 3), allows to establish a relation between any two memory components. These relations are focused on the first level of cache; the user specifies for each first level cache (i.e., D\$ and I\$) both the execution pipeline and the remainder memory components that are related with that specific component. The flexibility provided by this simple grammar allows to declare complex memory configurations, including *N:M* relations as is the case of first level caches and TLBs (i.e., a single Data Cache may use more than one DTLB).

As a matter of example, Figure 6 shows a Memory Subsystem example for a 3-core system. To specify all the constituent memory components shown in Figure 6 it should be used the following declaration:

-NUM_DCACHES 3 -NUM_ICACHES 2 -NUM_DTLBS 2
-NUM_ITLBS 1 -NUM_BUSES 2 -NUM_L2_CACHES 2
-NUM_L3_CACHES 1

Once declared all the memory components, the relations between them are declared using the memory relation grammar shown in Figure 5, as depicted in Figure 7. For a Memory Subsystem to be fully declared, every first level cache

[4]There must be at least 1 declared component of each type except for L3 Caches, which are optional.

[5]Unless a single component of any type was declared (e.g., in a system with a single DCache all cores must access to that DCache). In that case the corresponding relations with other components are implicitly assumed.

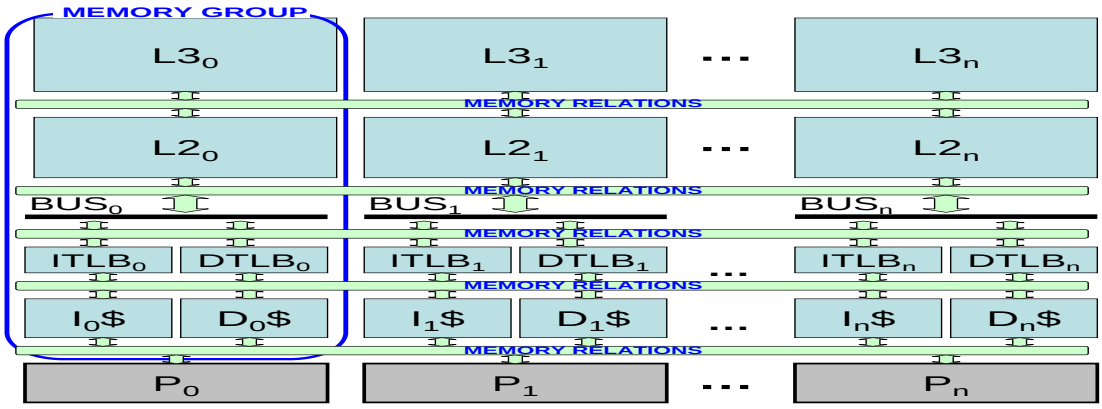

Figure 4. MPsim Memory Subsystem.

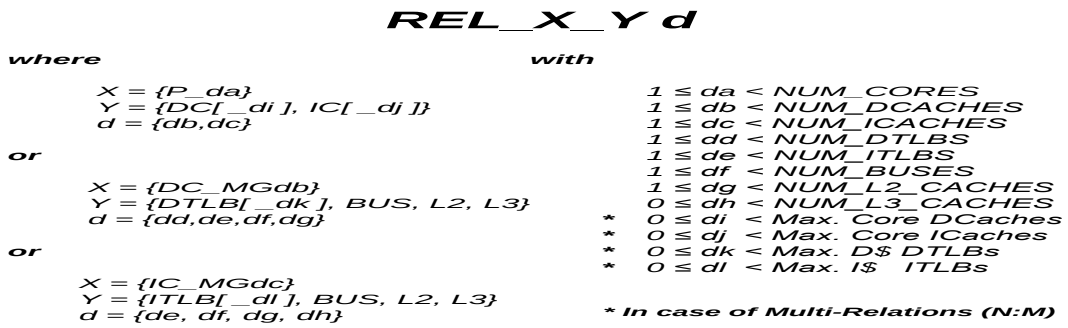

Figure 5. MPsim Memory Relation Regular Expression Grammar.

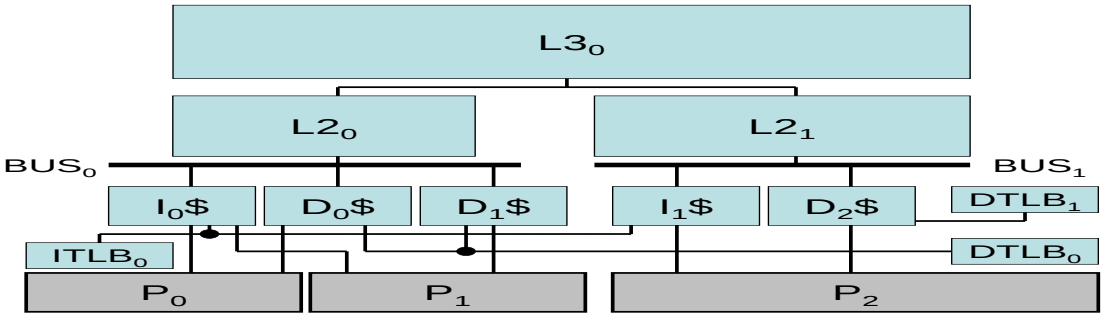

Figure 6. MPsim Memory Subsystem Example.

(ICaches and DCaches) must be related with some pipeline (or multiple pipelines), TLB (or multiple), L1-L2 bus, L2 Cache and optionally with some L3 Cache. Finally, each memory component is configured using its specific parameters (e.g., *-DTLBPENALTY_MG1 300 -DPGSIZE_MG1 13 -DTLB_SIZE_MG1 512* configures the DTLB number 1 with 512 entries, a miss penalization of 300 cycles and a 8Kb virtual page size –2 to 13–). As with pipeline configuration, the *MPsim Parameter Interface* allows to maintain a Memory Subsystems & Relations Repository (*memHierarchies* directory) and use them to declare more complex configurations. As a matter of example, let be POWER5_MEM and POWER5_MEM_rels the configuration files comprising all memory component configuration parameters and the relations between them, respectively, to configure a POWER5-like [1] Memory Subsystem. We would use the following declaration to fully configure a POWER5-like Memory Subsystem:

-pf memHierarchies/POWER5_MEM

-pf memHierarchies/POWER5_MEM_rels

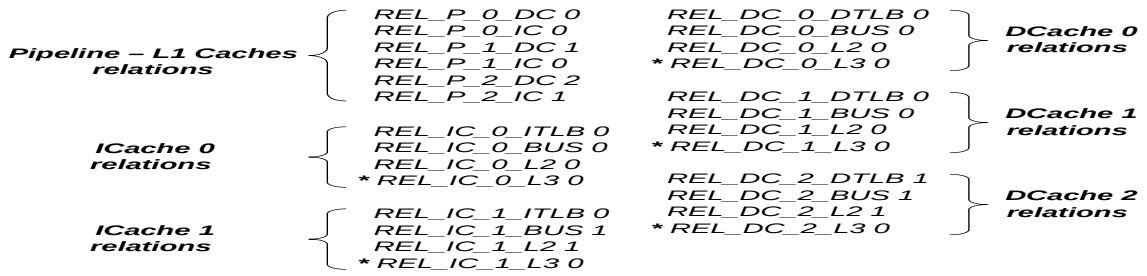

Figure 7. MPsim Memory Component Relations Example.

Since it is not possible to cover all *MPsim* functionality, due to space constraints, we focus on two main Memory Subsystem functionality issues in the remainder of this section. In Section 5.1 we describe the Multibanked and Multiported Cache functionality and the L2 Cache Access Arbiter in Section 5.2.

5.1 Multibanked & Multiported Caches

For each cache declared in the Memory Subsystem, it is possible to specify the number of banks in which it will be splitted. The *MPsim Parameter Interface* provides this functionality by means of the ICACHEBANKS_MGx, DCACHEBANKS_MGx, L2CACHEBANKS_MGx, L3CACHEBANKS_MGx parameters, where *x* stands for the specific Memory Group. Additionally, each cache may be configured with a different number of access ports, using the NUM_DCACHE_PORTS_MGx, NUM_ICACHE_PORTS_MGx, L2CACHEBANKPORT_MGx parameters, where *x* stands for the specific Memory Group. As a matter of example, the following declaration configures a 4-bank 4-port L2 Cache and an 8-bank 2-port DCache, shown in Figure 8 :

-NUM_DCACHES 1 -DCACHEBANKS_MG0 8

-NUM_DCACHE_PORTS_MG0 2 -NUM_L2_CACHES 1

-L2CACHEBANKS 4 -L2CACHEBANKPORTS_MG0 4

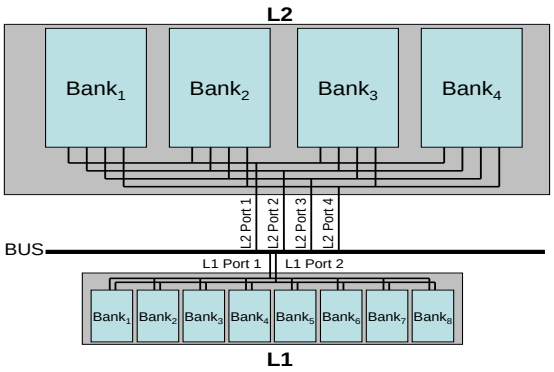

Figure 8. 4-bank 4-port L2 Cache and 8-bank 2-port L1 Cache Example.

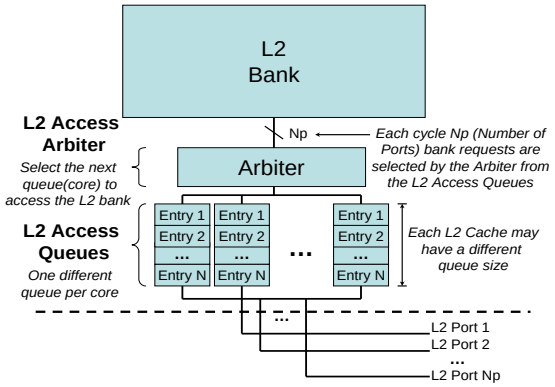

Figure 9. L2 Cache Access Arbiter.

5.2 L2 Cache Access Arbiter

The *MPsim* allows defining multicore system configurations in which many cores may share a single L2 Cache. In order to cope with the L2 Cache contention among all cores the *MPsim* provides an L2 Cache Access Arbiter, that can be activated using the *L2_ACC_ARBITER* parameter. The *MPsim* L2 Cache Access Arbiter, shown in Figure 9, manages the access to each L2 Cache bank using a queue per each core sharing that L2 Cache. Each of these queues buffer the core's L2 Cache access requests until the user-definable L2 Arbiter removes it from the corresponding queue and triggers the L2 Cache Bank access; as many requests allowed per simulated cycle as L2 Cache ports defined in the Memory Subsystem declaration. Whenever an L2 Access Queue gets full the corresponding core is temporarily stopped (no forward progress in any pipeline stage) until some queue entry gets empty.

6 Computational Cost

Although high-flexibility constitutes a very important characteristic for a general-purpose simulator it may not be achieved regardless its computational cost. Due to finite computational resources, computer architects require simulation tools that are able to yield results in a limited amount of time, according to reseach deadlines. It must be kept in mind that the results obtained from such a simulation tool generally constitute a first step in a multistep evaluation process. Due to their limited accuracy, general-purpose simulators are normally used to both identify architectural challenges and obtain general trends. In this sense, the flexibility offered by the selected simulation tool is of crucial importance. The tool's ability to allow simulating multiple architectural alternatives with low effort helps to both accelerate and improve this first evaluation step. Once identified the architectural challenge and evaluated a possible solution, more accurate results may be obtained employing either a more specific (and complex) simulator or FPGAs [11].

Although focused on high-flexibility, the *MPsim* design strives to involve the least computational cost possible. The idea is to provide a flexible and easy-to-use simulation tool, that allows the user to simulate a wide range of simulation alternatives with low effort, able to yield simulation results in a reasonable time. Although a priori conflicting, it may be found a satisfactory balance between these design goals. Following are enumerated some of the main design decisions employed in the *MPsim* development:

1. *Parameter Interface*. Providing high-flexibility should not interfere with the simulation itself. The parameter interface should be adaptable to accommodate future simulation improvements but it should not interfere with the inner simulation structures.

2. *Initialization Phase*. The configuration parameters acquisition, performed by a a flexible and easy-to-use parameter interface, should be immediately followed by an Initialization Phase. During this preparatory phase it should be anticipated all the work possible according to the simulation configuration. Thus, while some simulator modules could be fully deactivated, without compromosing neither memory nor processing in the subsequent simulation, others could be devoted enough memory to get rid of time consuming dynamic memory allocation/deallocation. According to the specific simulation configuration and the available resources, the Initialization Phase may considerable reduce the subsequent simulation computational cost.

3. *Avoid unnecessary work*. Instead of requiring function calls to determine whether a module is activated or not during the simulation, each module may include macros. Without compromising neither the code legibility nor modularity, a macro including a conditional branch to the corresponding function call may reduce the additional cost for deactivated modules; adding only an extra conditional branch for activated ones. Furthermore, since modules are activated/deactivated only durign the *Initialization Phase*, these branches are easily predictable.

In order to give an idea of the computational cost involved by the *MPsim* simulation tool, following are included some simulation results. For this set of experiments we use the *SPEC2000 benchmark suite*. From them we collected traces of the most representative 300 million instruction segment of each benchmark, following the idea presented in [7]. Each program is compiled with the *–O2 –non_shared* options using DEC Alpha AXP-21264 C/C++ compiler and executed using the reference input set. Fortran programs are compiled with the DIGITAL Fortran 90/Fortran 77 compilers. The fast forwards applied to each application, in order to obtain the traces, are shown in Tables 2 and 3.

Using the resulting traces, we collected workloads comprised of 1,2,4 and 8 benchmarks, shown in Table 1. The name of each workload[6] is *xWy*, where *x* stands for the number of threads involved and *y* stands for the workload identifier (e.g., *4W2* identifies the second workload with 4 threads). Each workload with size *x* is simulated on a *CMP+SMT* implementation with shared L2 Cache and $\frac{x}{2}$ two-hardware-context *SMT* cores implementing *ICOUNT* [10] policy; both single-thread and dual-thread workloads are simulated on a single-core implementation. Both core-specific and memory subsystem configuration parameters are shown in Table 1.

All workloads were simulated on a Dual-Core 2 Intel Xeon processor with 2,333GHz, 1333MHz FSB, and 4MB cache running Linux 2.6.15. Figures 10, 11, 12, and 13, show the time required to simulate each workload until any of the comprising benchmarks finish simulating 300 million instructions. Except for the *181.mcf*, with a pathological bad memory behavior due to nested memory references, all single-thread workloads are fully simulated in about twelve minutes time, which constitutes a reasonably low computational cost. As could be a priori expected, doubling the number of benchmarks in the workload (i.e., dual-thread workloads *–2Wy–*) doubles the required simulation time, as shown in Figure 11. Adding more dual-thread *SMT* cores, and consequently simultaenously simulating more benchmarks, increases the required simulation time as shown in Figures 12 and 13.

[6] Except for single-thread workloads, represented by the name of the corresponding benchmark.

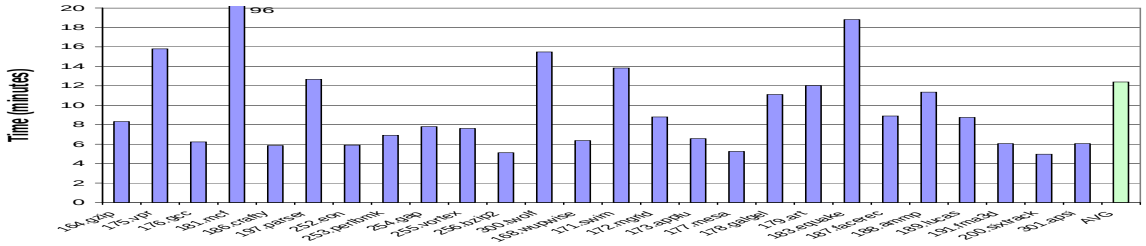

Figure 10. Single-Core Single-Thread Simulation Cost.

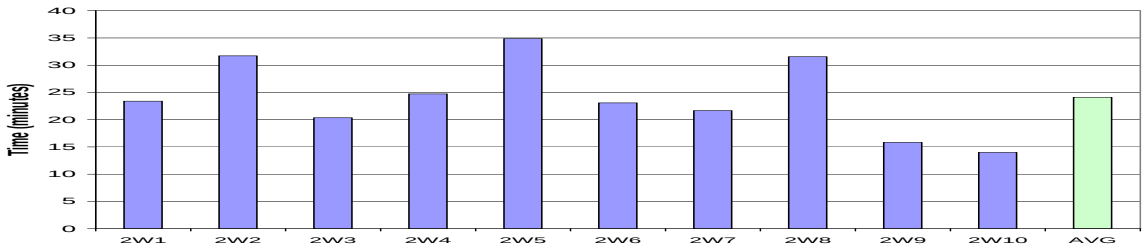

Figure 11. Single-Core Dual-Thread Simulation Cost.

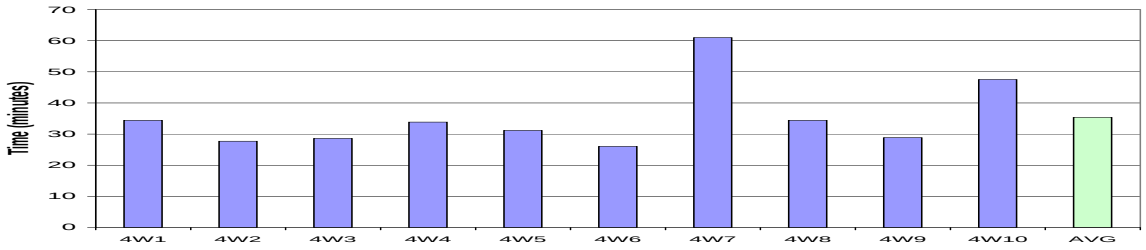

Figure 12. Dual-Core Dual-Thread Simulation Cost.

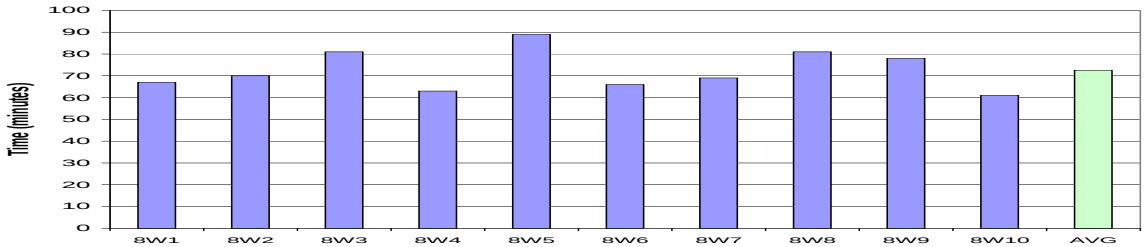

Figure 13. Quad-Core Dual-Thread Simulation Cost.

7 Conclusions

In this paper it is presented the *MPsim*, a highly-flexible general-purpose simulation tool. The *MPsim* constitutes a cycle-accurate multi-purpose simulation tool which allows simulating a wide range of processor types. Both single core (Superscalar, SMT) and multi core (CMP, CMP+SMT), so as homogeneous and heterogeneous configurations, are available and may be employed by means of its flexible *MPsim Parameter Interface*.

The *MPsim* has been developed, and keeps on evolving, to constitute a simulation tool to assist computer architecture research in a wide range of scenarios. The programming phylosophy employed in the *MPsim* development favors *high-flexibility*, without critically compromise computational costs, so as new ideas could be easily added to the simulation tool functionality. In this paper it is put special emphasis on how this flexibility is obtained and the computational cost it involves. The simulation results included confirm that high-flexibility may be provided in a general-purpose simulator without hardly compromising its computational cost.

Acknowledgements

This work has been supported by the Ministry of Education of Spain under contract TIN2007-60625, the Barcelona SuperComputing Center(BSC) and the HiPEAC European Network of Excellence. The authors wish to thank Daniel Ortega, Ayose Falcon, Jeroen Vermoulen, Ruben Gran, and

Simulation Parameters	
Pipeline depth	11 stages
Pipeline width	6 instructions/cycle
Queues Entries	64 int, 64 fp, 64 ld/st
Execution Units	4 int, 3 fp, 2 ld/st
Physical Registers	320 regs.
ROB Size*	256 entries
Branch Predictor	perceptron (4K local, 256 perceps.)
BTB	256 entries, 4-way associative
RAS*	100 entries

Simulation Parameters	
L1 I-Cache	64KB, 4-way, 8 banks
L1 D-Cache	32KB, 4-way, 8 banks
L1 lat./miss	3/22 cycs.
I-TLB ,D-TLB	512 ent. Full-associative
TLB miss	300 cycles
L2 Cache	4MB, 12-way, 4 banks
L2 latency	15 cycles
Main Memory lat.	250 cycles

	Number of Threads		
Name	2	4	8
xW1	b, j	b, q, t, j	d, l, b, g, i, j, c, f
xW2	n, e	l, n, p, e	b, g, m, n, a, h, o, p
xW3	d, a	d, s, r, a	m, n, r, q, i, j, e, h
xW4	g, f	g, b, m, f	l, b, g, m, n, r, f, s
xW5	r, p	r, j, f, p	q, b, c, k, e, a, o, t
xW6	b, j	b, q, t, j	d, l, b, g, i, j, c, f
xW7	n, e	l, n, p, e	b, g, m, n, a, h, o, p
xW8	d, a	d, s, r, a	m, n, r, q, i, j, e, h
xW9	g, f	g, b, m, f	l, b, g, m, n, r, f, s
xW10	r, p	r, j, f, p	q, b, c, k, e, a, o, t

gzip	a	eon	h	apsi	o	facerec	v
vpr	b	gap	i	wupwise	p	applu	w
gcc	c	vortex	j	equake	q	galgel	x
mcf	d	bzip2	k	lucas	r	ammp	y
crafty	e	twolf	l	mesa	s	mgrid	z
perlbmk	f	art	m	fma3d	t		
parser	g	swim	n	sixtrack	u		

Table 1. Simulation parameters and Workloads. (resources marked with * are replicated per thread)

Oliverio J. Santana for their support and help in the *MPsim* development proccess.

Benchmark name	Input	Language	Fast Forward (Millions)
164.gzip	graphic	C	68.100
175.vpr	place	C	2.100
176.gcc	166.i	C	14.000
181.mcf	inp.in	C	43.500
186.crafty	crafty.in	C	74.700
191.parser	ref.in	C	83.100
252.eon	cook	C++	57.600
253.perlbmk	splitmail.535	C	45.300
254.gap	ref.in	C	79.800
255.vortex	lendian1.raw	C	58.200
256.bzip2	inp.program	C	13.500
300.twolf	ref	C	324.300

Table 2. FastForward used for each Spec INT 2000 Benchmark.

Benchmark name	Input	Language	Fast Forward (Millions)
168.wupwise	wupwise.in	Fortran77	263.100
171.swim	swim.in	Fortran77	47.100
172.mgrid	mgrid.in	Fortran77	187.800
173.applu	applu.in	Fortran77	10.200
177.mesa	frames100 + msea.in	C	294.600
178.galgel	gagel.in	Fortran90	175.800
179.art	-scanfile c756hel.in -trainfile1 a10.img -trainfile2 hc.img -stride 2 -startx 110 -starty 200 -endx 160 -endy 240 -objects 10	C	13.200
183.equake	inp.in	C	27.000
188.ammp	ammp.in	C	13.200
189.lucas	lucas2.in	Fortran90	30.000
191.fma3d	fma3d.in	Fortran90	10.500
200.sixtrack	sixtrack.in	Fortran77	173.500
301.apsi	apsi.in	Fortran77	192.600

Table 3. FastForward used for each Spec FP 2000 Benchmark.

References

[1] J. M. Tendler R. J. Eickemeyer . Sinharoy, R. N. Kalla and J. B. Joyner. POWER5 system microarchitecture. *IBM Journal of Research and Development. 49(4/5):505–52*, 2005.

[2] C. Acosta, A. Falcón, A. Ramírez, and M. Valero. A Complexity-Effective Simultaneous Multithreading Architecture. In *Proc. of ICPP-35*, 2005.

[3] F. J. Cazorla, E. Fernández, A. Ramirez, and M. Valero. Improving Memory latency aware fetch policies for SMT processors. In *Proc. of ISHPC-V*, 2003.

[4] R. Desikan, D. Ernst, M. Guthaus, N.S. Kim, E. Larson, T. Mudge, H. Murukathampoondi, K. Sankaralingam, B. Yoder, T. Austin, and D. Burger. SimpleScalar Version 4.0 Release. *held in conjunction with MICRO-34*, December 2001.

[5] D.A. Jimnez and C. Lin. Dynamic Branch Prediction with Perceptrons. In *Proc. of HPCA-7*, pages 197–206, 2001.

[6] S. McFarling. Combining Branch Predictors. Technical Report WRL-TN-36, 1993.

[7] T. Sherwood, E. Perelman, and B. Calder. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications. In *Proc. of PACT-10*, 2001.

[8] D. M. Tullsen and J. A. Brown. Handling Long-latency loads in a Simultaneous Multithreaded Processor. In *Proc. of MICRO-34*, 2001.

[9] D. M. Tullsen, S. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proc. of ISCA-22*, 1995.

[10] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In *Proc. of ISCA-23*, 1996.

[11] W. Wolf. FPGA-Based System Design. Prentice Hall, 2004.