# Manycore Simulation for Peta-scale System Design: Motivation, Tools, Challenges and Prospects

Javad Zarrin[a,*], Rui L. Aguiar[b], João Paulo Barraca[b]

[a]*Instituto de Telecomunicações, Campus Universitário de Santiago, Aveiro, Portugal*
[b]*Universidade de Aveiro, Aveiro, Portugal*

**Abstract**

The architecture design of peta-scale computing systems is complex and presents lots of difficulties to designs, as current tools lack support for relevant features of future scenarios. Novel systems must be designed with great care and tools, such as manycore architecture simulators, must be adapted accordingly. However, current simulation tools are very slow, often specific-purpose-oriented, suffer from various issues and are rarely able to simulate thousands of cores. The emergence of peta-scale systems and the upcoming manycore era brings nevertheless new challenges to computing systems and architectures, adding further difficulties and requirements on the development of the corresponding simulators. Furthermore, the design of architecture simulators for manycore systems involve methods and techniques from various interdisciplinary research areas, which in turn brings more challenges in different aspects. As system complexity grows, the growth of the simulation capacity is being outpaced (reaching the so called simulation wall). In this paper, we present the challenges for simulating future large scale manycore environments, and we investigate the adequacy of current modeling and simulation tools, methodologies and techniques. The aim of this work is to highlight how current approaches can best deal with the identified problems, smoothing the challenges of research in future peta-scale systems.

*Keywords:* architectural simulations, manycore simulators, full-system simulation, manycore systems, computer architecture

## 1. Introduction

Current trends in computation technology have focused in improving performance by increasing the number of cores per die (parallelism) rather than by increasing the rate of clock frequency of each core, due to the exhaustion of the Moore's law. Many companies and academic communities pushed this trend, designing multicore and manycore systems with capacity of tens to hundreds of cores per single die. These manycore processors are more like data-centers-on-a-chip than previous single processors, as a complex communication network connects the different cores. It is predictable in a near future to consider systems with a very large interconnect network in manycore machines with dimensions from thousands to millions of cores.

A consequence is that both system design and programming concepts must increasingly focus in heterogeneous parallelism. Future parallel and distributed applications, compilers, operating systems and tools must be able to scale well with the hardware nature of manycore and distributed execution. However, increasing the number of cores on a die increases the complexity of hardware designs, and has considerable impacts which result in enlargement of the potential design-space. Moreover, it brings serious challenges particularly for memory hierarchies and on-die interconnect bandwidth, both within the die and off die.

Systems must then be designed with great care and tools, such as manycore architecture simulators, must be adapted to address these disruptive challenges. Manycore simulators (i.e. manycore architecture simulators) can assist researchers from different areas. These areas include software (such as programming models, operating systems, compilers, etc.), hardware and computing architectures to model and assess future systems. The upcoming manycore era brings new challenges to computer architects that must be paralleled by the development of adequate architecture simulators. Ideally, to simulate a fully parallel system we can expect that an efficient architecture simulator should (at-least) be able to be parallelizable, and use the benefit of concurrency, enabling faster evaluation of future systems. Moreover, architecture simulators should provide a highly scalable, fast and accurate model to describe, emulate/simulate and measure the hardware details, memory hierarchy and interconnection networks. Furthermore, they must meet the stringent requirements along the lines of productivity, multi-modeling, synchronization, modularity, and event sampling capabilities. However, these properties are not true for the majority of the currently available architecture simulators. As complexity grows, the performance of a single simulated CPU core slows down, and the usage of these sequential simulators (i.e. architecture simulators) will be mainly limited by the performance of simulating a single CPU.

Note that, in this paper, we use the term "simulator" to refer to "architecture simulator" or "manycore architecture simulator" which differs from its more general meaning. We use the term "manycore simulation tools" in a slightly more general way to cover a wider range of contexts (methods and techniques). In

*Corresponding author. Tel:+351 234 377 900, Fax: +351 234 377 901
*Email addresses:* javad@av.it.pt (Javad Zarrin), ruilaa@ua.pt (Rui L. Aguiar), jpbarraca@ua.pt (João Paulo Barraca)

other words, a simulator (architecture simulator) is a type of manycore simulation tool. Furthermore, the term "emulator" (or architecture emulator), as used in this paper, also differs from its general meaning in other research fields. In this paper, an emulator means a simulator which lacks support for performance measurements. In fact, we associate an emulator with functional correctness only. This means that the notion of time for an emulator is imprecise and often just a representation of the wall-clock time of the host. We use the term "emulation" to refer both to "functional simulation" and to describe the act of an emulator.

The rest of this paper is organized as follows. In Section II, we discuss why manycore architectural simulation is needed, particularly for research on peta-scale systems. Section III presents in detail our taxonomy to addresses current modeling and simulation tools, as well as the methodologies that could be exploited and enhanced in order to design next generation efficient simulators. In Section IV, we provide an overview of recently proposed simulation tools for architectural analysis, which are able to simulate the entire execution cycle of application for the target systems. Other simulator types, such as those that mainly focus on physical modeling aspects (e.g. power, energy and thermal) or interconnect simulators (e.g. Network on Chip simulators), are out of the scope of paper for conciseness. In Section V, we extract and elaborate a set of major problems and challenging issues created by manycore simulation. Finally, in Section VI, we present our conclusions, followed by a discussion of future directions for research. This includes possible approaches and solutions which can be used to solve the problems and deal with the challenges identified.

## 2. Simulation and Peta-scale Systems

Peta-scale systems are defined as systems which are able to provide peta-FLOPS, millions of billions of FLoating OPerations per Second, computational power [1, 2]. They can be described as the increasingly massive and dynamic networks of interconnected diverse processors and components (i.e. elements). Such as system, as a whole, exhibits a set of properties and behaviors among the elements, which are not distinguishable from the properties of the individual processors and components. These systems are only on its infancy currently, but in future peta-scale manycore systems, we can expect to have computing nodes with more than 10 thousands cores per node. We can also expect to have much more diversity (heterogeneity) of cores, interconnections and architecture designs compared to today systems.

Architectural simulation is a common method for studying and analyzing different architectures, designs and algorithms for various target systems through imitating the operation of real-world processes, processors and systems over time. Architectural simulation acts as a low cost alternative to experimentation on real systems by representation of key characteristics, behaviors and functions of the real systems. The objective of simulation is to provide capability to researchers and designers to flexibly and efficiently explore a design space. This can include analyzing the performance of current systems (e.g. architecture assessment), acquiring and predicting processor/system behaviors and evaluating novel designs. Simulation enables today's designers to analyze and predict different aspects (such as performance, reliability and efficiency) of future's machines. This means that simulation is a particularly useful tool when the desired target systems, such as peta-scale systems, do not currently exist in reality. However, current simulation tools are very slow, often specific-purpose-oriented, suffer from various issues and are rarely able to simulate more than 2000 cores (we discuss current simulation tools further in Section 4). Furthermore, introducing the concept of peta-scale system presents more requirements that must be fulfilled by current simulation tools. We discuss these requirements and their corresponding issues further in Section 5.

Figure 1 demonstrates a generic structure for (software-based) simulation. As it is shown in the figure, a simulator is an application software which runs on single host or multiple networked host machines (distributed simulation). The target system is the system which needs to be simulated by simulator. Depending on user requirements the target system might be a partial system or a full system including target (simulated) OS. The term "target Instruction Set Architecture (or target ISA)", as used in this paper, refers to the ISA of the processor architecture simulated. Similarly, the term "host ISA" refers to the ISA of the host machine which runs the simulation. Note that, it might be possible to have some similarities between a target system and its host system (e.g. target ISA = host ISA or target OS = host OS). In such a case, a simulator can alternatively employ a different simulation approach which might be faster (or easier to implement) (see Section 3). More details of this structure are discussed in the next section.

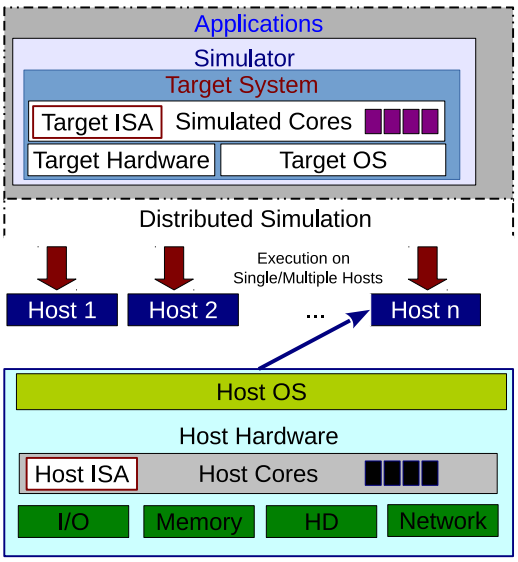


Figure 1: Overall Simulation Structure

## 3. Manycore Simulation

Simulators are essential tools for the design and evaluation of computer and system architectures. Taking in consideration the

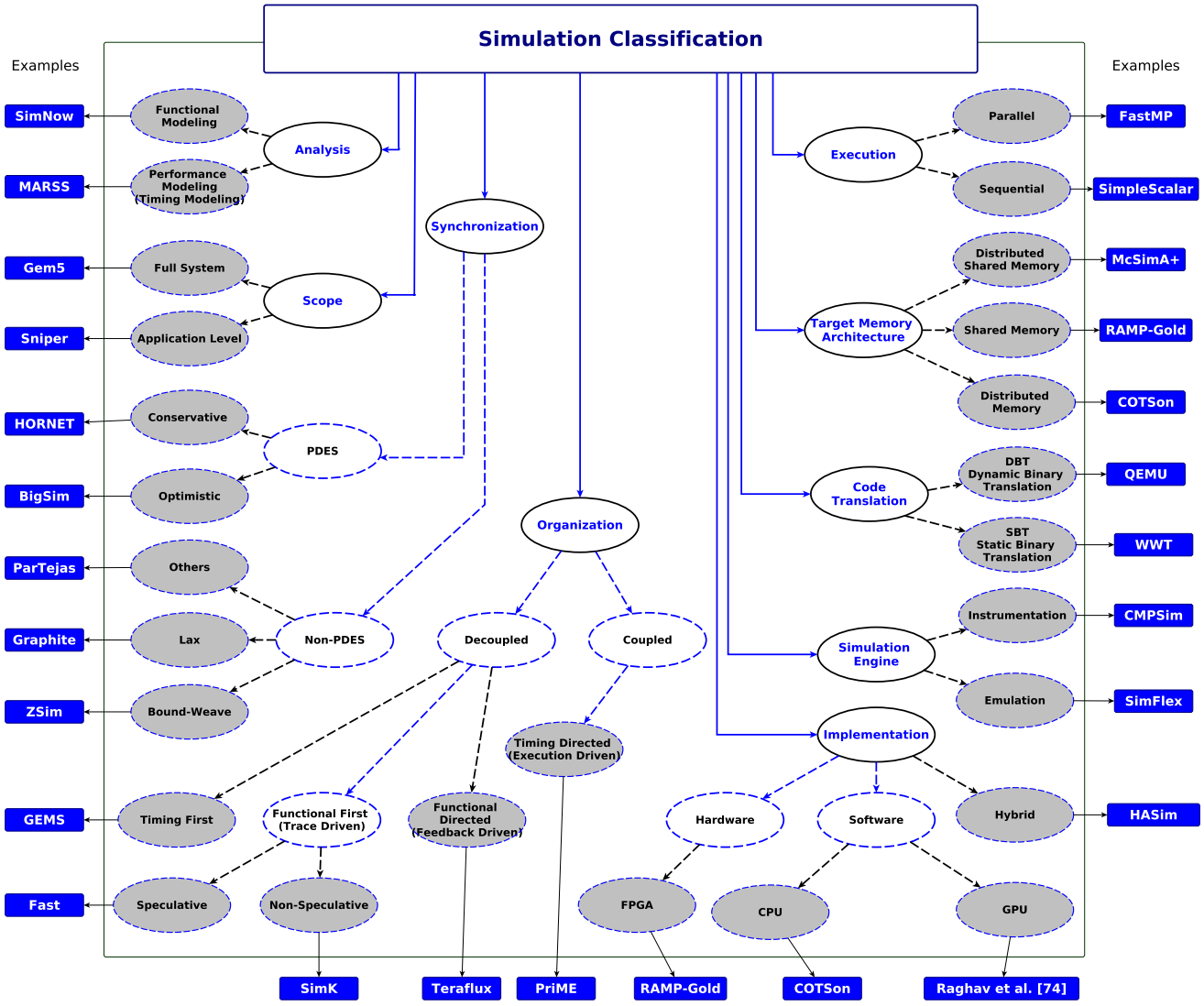


Figure 2: Simulation Classifications

objective of the evaluation, there are a large variety of different simulation tools, techniques and methods in the current literature. Figure 2 presents a snapshot of the most important classifications for manycore simulation tools and related technologies. Figure 2 also presents example simulator for each category (note that a simulator can be in multiple categories). We discuss these classifications and their related concepts in detail in Section 3.1.

### *3.1. Terminology and Classifications*

There are plentiful tools, methods and techniques for manycore simulation in the current literature. These can be classified along many dimensions, such as: user/application-level vs full-system; functional vs timing; trace-driven vs execution-driven; cycle-driven vs event-driven. In the following, we discuss and briefly explain the terminology and concepts associated with some of the most important classifications.

**Emulators vs Simulators:** Emulators are tools able to demonstrate the functional behavior of the system, focused on the exact reproduction for an external system. They repeat the function of a target platform on a host platform. Simulators differ from the former because they consider an abstract, simplified model and do not try to replicate all the aspects of a system. A simulator in addition to ensuring the functional correctness, must provide capability to study and analyze the performance of proposed system/hardware designs by using accurate timing information.

**Functional vs Timing/Performance:** According to our terminology (see Section 1), architecture emulators include only functional models while simulators contain both functional and performance models. The functional model (functional partition) is in charge for the correct execution of the target Instruction Set Architecture (ISA) (i.e. architectural modeling). In addition, functional models may provide facility to observe the interactions among processors, memory and I/O peripherals without modeling micro-architectural details. As examples of the tasks for a functional model we can mention aspects as decoding in-

structions, updating simulator memory and verifying the floating point operations.

The performance model (timing partition) is responsible to drive a functional model. This is done by providing accurate timing information in a way that simulating a particular micro-architecture (at least in a specific aspect) would be possible (i.e. micro-architectural modeling). In other words, a timing model is able to determine the time the target architecture takes to execute an instruction. Examples of the tasks for the timing model include making decisions to select the next instruction for execution, tracking branch mispredictions, and predicting the clock cycles to execute instructions.

Creating a new functional model might be complex in terms of implementation, optimization and verification, but after that, it can modularly be coupled and reused across various timing models. On the other hand, implementation of only a timing model with capability to reuse an existing functional model can be much easier and less time consuming than developing a complete simulator (including both functional and timing models) from scratch. Unlike functional models, the correctness of target architectures (in term of ISA functional) is not of primary concern for timing models. What matters is to track and control the accuracy of micro-architecture-specific timing details [3].

**Simulation Scope:** An architectural simulator is defined as a piece of software which mimics the behavior of a real computer system with ability to estimate performance and outputs for a given input (application). It may model different computer devices and components (i.e. only a single target microprocessor, or an entire computer system including processors, memory system, and I/O devices) with different level of details. The simulation scope specifies the scope of target systems that an architectural simulator can model. According to this, we classify the processor simulators (i.e. architectural simulators) into two categories: full-system simulators and user/application level simulators (see Figure 3).

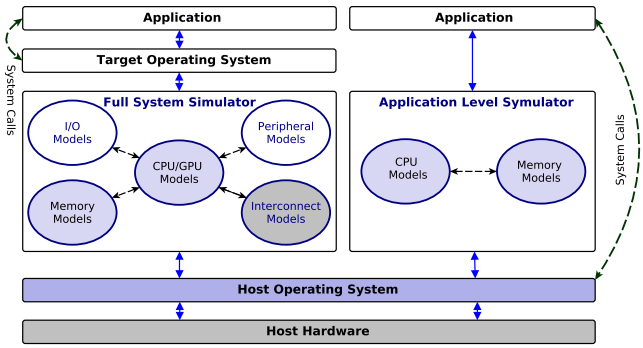


Figure 3: Full System Simulator vs Application Level Simulator

A full system simulator provides capability to run a detailed, complete and real software stack on the target (simulated) system without any modification. The software stack might be OS, complex applications such as multithreaded and multiprocess workloads, or applications that highly exercise system calls, I/O and networking. The full system model generally includes processor cores, memories, network and interconnection, buses, peripheral devices and privileged modes. However, the supported models and the level of details for each model are various for different simulators. A full system simulator, besides timing/performance modeling might also supports simulation of the physical models such as power, energy and thermal. Gem5 [4], Flexus (SimFlex) [5], and MARSS [6] are some examples of full system simulators.

Furthermore, full system simulators can be very important design tools, particularly for System-on-Chip (SoC) simulation where it is necessary to efficiently cope with the huge hardware/software design space. SoC architecture integrates all components of a (electronic/computing) system (including complex hardware/software) into a single integrated concept. This provides capability to scale computing performance / power efficiency through combining (massively parallel and high performance) manycore processors, Network-on-Chip (including several network/interconnect communication protocols) and software (including OS and various computing intensive user applications) [7, 8].

On the other hand, user/application level simulators model only the user side of applications without OS support. They rely on the host machine (host OS) to service the system calls and execute a given user code of a benchmark on top of the simulator. These simulators are easier to develop and use, since they do not boot an OS. But they are also limited to only support specific workloads. In other words, they can not run applications such as multithreaded and JVM workloads that frequently use applications that are sensitive to system time like and client-server workloads. As examples of user/application level simulators we can mention SimpleScalar [9], BigSim [10], Graphite [11], Sniper [12], ZSim [13] and PriME [14].

**Simulator Organization:** Each simulator generally contains functional and timing partitions which interact in order to form a complete simulation. According to whether the functional and the timing partitions are completely separated or not, simulators can be categorized into decoupled and coupled. Coupled simulators potentially can provide flexibility for developing precise simulations (i.e. highly detailed simulation), due to speculative execution modeling (producing all the values and possible side effects) and timing-dependent outcomes. However, this flexibility is reduced when there is need for frequent modifications of functional or performance models. In other words, complexity arises due to new complex devices (functional models), modern performance/timing models and their internal interactions. For coupled simulators, it is challenging to address multiple different conflicting demands (such as simulation precision, accuracy, flexibility and performance) in a single simulator component which integrates both functionality and timing modeling. On the other hand, decoupled simulators aim to reduce this complexity by completely decoupling functional and timing models. Accordingly, they can achieve better flexibility and potentially other advantages such as accuracy (through correctness verification) and modularity (which can lead to faster development and easy modification) [15]. The drawback for this approach is to introduce redundancy which may reduce the simulation performance. Moreover, decoupled simulators typically call an external simulator/emulator to perform functional model-

ing. This might increase the difficulty for modeling interactions among functional and timing models.

Based on the type of interaction or the relationship between the functional model and the timing model, the simulators can use different organizations. A taxonomy of microarchitectural simulator organizations is introduced in [15] and [16]. We extend those taxonomies and classify the simulator organizations into the following five categories (Table 1 presents the advantages and disadvantages for each of the aforementioned simulator organizations):

1- Functional First (Trace Driven): The functional-first is a completely decoupled approach. The functional model is separately executed first, and later, when the functional modeling is finished, the timing model is run. As demonstrated in Figure 4, the functional model executes instructions and produces traces that are streams of information about the execution. Traces are then fed into the timing partition where the microarchitectural simulation is performed. Depending on whether speculative execution is supported or not, the functional-first organization can be categorized into speculative and non-speculative.

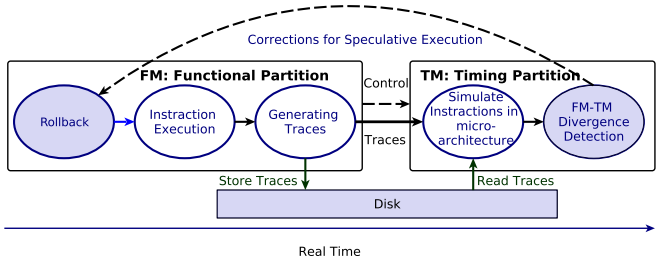


Figure 4: Functional First Organization - The gray elements (rollback and mismatch detection) are only supported by speculative-functional-first organization

The speculative functional-first assumes that all parts of execution (not just for branch control) are speculative. When the timing partition detects that the execution in functional partition has differed from the timing partition, it asks the functional partition to undo or rollback the effects of wrong path for instruction execution, and to redirect fetch. Examples of this approach include UTFast [17], FastSim [18], SimpleScalar [9], Zesto [19], ReSim [20], BigSim [10] and Transformer [21].

Unlike speculative approach, in non-speculative functional-first, rollback mechanism is not supported. This means that the timing partition independently provides the timing model in a highly decoupled fashion and regardless of any potential execution divergence (mismatching functional execution and timing model). Non-speculative approach might be easy for implementation and parallelism through a highly modular and decouple design. However, it potentially suffers from execution divergence issues which results in being inadequate for modeling timing-dependent execution behaviors (such as interactions among threads in a multithreaded application). Examples of this approach include single thread work load simulators (like PTCMP [22]).

2- Functional Directed (Feedback Driven or Event Driven): The functional-directed is a completely decoupled approach where the functional model drives the timing model. The functional model is assumed to be accurate, but timing feedback from the timing model is needed in order to correct the timing behavior. As we can see in Figure 5, using a timing feedback mechanism, the timing model periodically adjusts the speed of functional execution to reflect the timing estimates. This gives the running application a more-precise timing correction. COTSon [23] and Teraflux [24] use this approach.

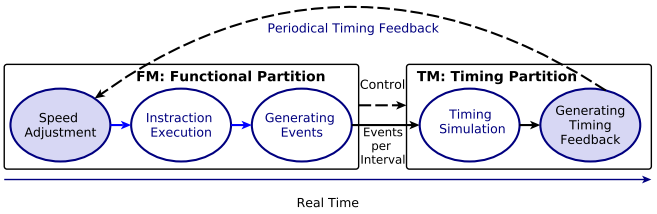


Figure 5: Functional Directed Organization

3- Timing First (Instruction Driven): The timing-first is a decoupled approach which uses the timing model to drive the functional model. Figure 6 demonstrates the timing-functional interactions for this organization. The timing model runs ahead of the functional model and simulates the functional behavior (i.e. instruction execution) in addition to the timing simulation.

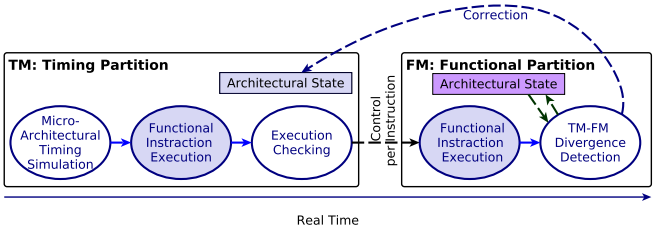


Figure 6: Timing First Organization

But the functional execution (for each instruction) must be checked and verified later by the functional model. This means that the functional model is only used for checking; when a mismatch (execution divergence) is detected, the timing model is notified by the functional model for the correction. In such a case, the architectural state of the timing model is reloaded from the functional partition. Examples of timing-first organization include TFsim [15], FeS2 [25] and GEMS [26].

4- Timing Directed (Execution Driven or Cycle Driven): In timing-directed simulators, functional and timing models are tightly coupled. As we can see in Figure 7, multiple interactions (per instruction) between timing and functional model raise system complexity. The timing model processes the flow of instructions through micro-architectural timing simulation. In each cycle, it directs the functional model with which step (e.g. fetch, decode, operand fetch, memory access, execute, write-back) of which instruction should be executed. Accordingly, the functional model returns the execution information to the timing model for each step. In timing-directed organization, the architectural state in both (timing and functional) partitions are naturally matched with each other, thereby preventing execution divergence. Talisman [27], Graphite [11], ZSim [13] and PriME [14] use this organization.

Table 1: Advantages and disadvantages of various simulator organizations

| Simulator Organization | Pros | Cons |
|---|---|---|
| Functional-First (Non-Speculative) | Easy implementation and parallelization, Highly-decoupled design, On-way data flow, Highly-Modular, Support for binary translation, direct execution and compiled-code simulation | No support for speculative execution, Not able to model timing-dependent outcomes between threads, No corrections for the timing model , Mismatching functional execution and timing simulation (i.e. execution divergence) |
| Functional-First (Speculative) | Matching between functional execution and timing simulator, Support for speculative execution and multithreaded applications, Natural parallelism of the functional and timing simulator, On-way data flow | The functional model must support speculation, The timing simulator needs to implement some of the functionality behaviors |
| Functional-Directed | Efficient trade-off between high speed functional modeling and high accurate timing simulation, Highly-adapted with sampling techniques, Support speculative execution and multithreading | Periodical feedback has impact on the simulation accuracy. |
| Timing-First | Single timing-functional call per instruction, Fast development, Debuggability of the timing simulator, Matching between functional execution and timing simulator. | No support for accurate modeling of interactions between threads, The timing simulator needs to implement almost all functionality behaviors. |
| Timing-Directed | Highly-accurate, Natural support for speculative execution and multithreading in timing partition, Potential support for memory and memory consistency modeling, Easy validation of the timing simulator, Matching between functional execution and timing simulator | Highly-coupled design, Complex functional modeling, Highly frequent inter-partition (timing-functional) communication which impacts the simulation speed, Multiple functional-timing calls per instruction. |

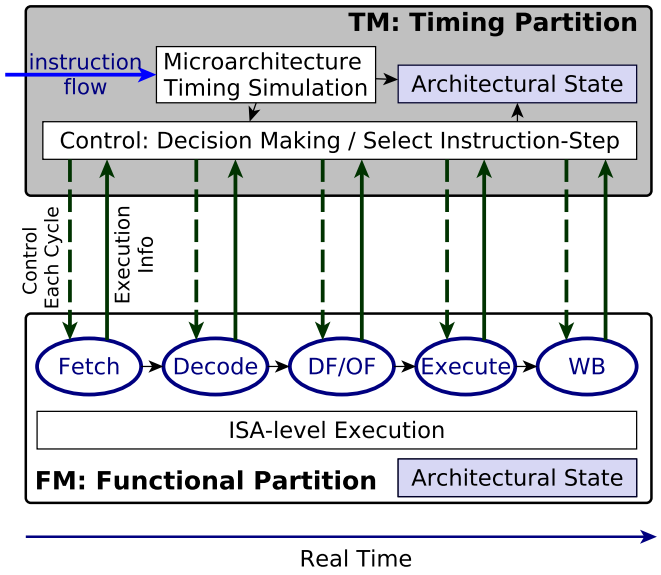


Figure 7: Timing Directed Organization

**Simulator Execution:** The execution of simulations on the host machines can be either sequential or parallel. Sequential simulators are highly accurate but as the complexity of the target architecture increases the simulation speed significantly decreases [28, 29]. There are numerous examples of sequential emulators and simulators in the literature including QEMU [30], Embra [31], Mambo [32], SimNow [33], SimpleScalar [9], Simics [34], SimOS [35], Rsim [36], Gem5 [4] and PROTEUS [37]. While most of these tools are able to model multicore architectures and multi-processors (in parallel chips), they are all limited to only utilizing a single processor on a single host machine which leads to significant slowdown in simulation speed. Parallel emulators (e.g. PQEMU [38], COREMU [39], Parallel Mambo [40] , and Parallel Embra [41]) and simulators (e.g. FastMP [42], SlackSim [43], Wisconsin Wind Tunnel Simulators (WWT) [44, 45], CMPSim [46], ZSim [13], MARSS [6], SimFlex [47], GEMS [26], COTSon [23], Graphite [11], Sniper [12, 48] and BigSim [10]) enhance the performance of simulation (in terms of simulation speed) by dividing and distributing the simulation workload across multiple cores on multiple host machines. However, parallelizing a simulator without sacrificing simulation accuracy is challenging. Furthermore, the execution of functional and timing models might be different. For example, COTSon [23] uses SimNow [33], a sequential emulator, for functional modeling, while it benefits of a parallel timing model.

**Parallelization and Synchronization Strategies:** Synchronization overhead is one of the major challenges for parallel simulation [49]. Different parallelization strategies and synchronization algorithms such as Parallel Discrete Event Simulation (PDES) [50], Lax synchronization [11] and Bound-Weave parallelization [13] have been proposed for parallel architectural simulators. Along this line we can classify parallel simulators based on their parallelization techniques into PDES-based and non-PDES-based simulators.

**I. PDES-based Synchronization:** PDES aims to facilitate fast execution of large simulation programs. It refers to the parallelization strategies for distributed simulation through execution of a single discrete event simulation program on a parallel computer [50]. Many of the conventional parallel micro-architectural simulators in the literature such as HORNET (conservative) [51], BigSim (Optimistic) [10], SimK (conservative) [52], SlackSim (hybrid) [43] and COTSon [23] are based on PDES. Using PDES, the components of the simulator are divided across host threads. The time-stamped simulation events from each component are distributed among multicores in parallel chips of multiple host machines. These events are executed concurrently while maintaining the causality relationship (i.e. cause and effect relationship) between them. This means that some sequencing order

between events executing in two separate processes must be maintained, although maintaining this causality relationship is challenging while exploiting inherent parallelism for faster job scheduling [53].

PDES-based synchronizations can be broadly classified as conservative (pessimistic), optimistic and hybrid. Conservative approaches, using pessimistic estimates, only process events when it is safe to do so. The events are scheduled in time stamp order while avoiding deadlocks efficiently. The synchronization is performed every time an ordering violation (causality error) may happen. On the other hand, in optimistic approaches, all events are executed speculatively and once an ordering violation is detected, the recovery can be performed by invoking a rollback mechanism.

There are several classic algorithms for conservative and optimistic synchronizations including asynchronous-conservative [54], Lower Bound Time Stamp (LBTS) [55] and optimistic time warp [56]. In asynchronous-conservative, there is no need to perform any global synchronization. Instead, various deadlock avoidance algorithms can be applied to ensure that only safe events in the future queue are processed. Using LBTS, in each cycle, the events in the future queue which have the lower bounded time stamp are executed while time progressing is managed by a global reduction and synchronization mechanism. In optimistic time warp, a causality error is detected whenever the time stamp of the received event message by a process is smaller than the process's clock. Many other PDES based synchronization mechanisms have been proposed in the last decade such as conservative null message (or CMB) [57], conservative forecast null message (FNM) [58].

PDES-based parallel simulators might support single or multiple synchronization algorithms. For example, Manifold [59] supports multiple standard PDES algorithms including LBTS [55], CMB [57], FNM [58] and time quantum synchronization [59].

Parallel simulators based on Optimistic-PDES generally suffer from poor scalability due to frequent roll-back and synchronization. On the other hand, simulators based on conservative-PDES provide better scalability due to their simplicity and less synchronization overhead in comparison with optimistic based simulators [51, 60]. Nevertheless, both approaches provide reasonable accuracy [13]. The scalabity for parallel simulators means that the simulation speedup constantly rises when the number of allocated host processors for simulation is increased [61]. Over the decades of research in the field, many optimization methods such as Lazy Cancellation [62], Lazy Reevaluation [63], Direct Cancellation [50], Early Cancellation [64], Space-time Simulation, Optimistic Time Windows [65], Wolf Calls [66] and Time Warp Straggler Message Identification [67] have been proposed to improve the overall performance of PDES based Simulators. However, PDES based parallel simulators are not yet truly competitive with performant sequential simulators with respect to simulation accuracy and scalability.

**II. Non-PDES-based Synchronization:** Non-PDES based parallel simulators such as Sniper [12, 48] and Graphite [11], relax synchronization requirements to obtain scalability by permitting micro-architectural events to occur out of order. However, these approaches provide complexity to model the actual behavior of the target system components such as memory controllers and shared caches while they sacrifice the simulation accuracy. In fact, Graphite leverages lax synchronization models to enable trade-offs between simulation speed and simulation accuracy while Sniper (which is built on Graphite) uses higher levels models by reducing accuracy compromise. ParTejas [68], a more recent work, is a shared memory based parallel simulator written in Java. Unlike Sniper and Graphite, ParTejas doesn't rely on highly relaxed synchronization, but rather it primarily relies on novel concurrent data structures. In fact, it uses a lock free parallel slot scheduler for synchronizing the accesses of multiple threads at a shared resource while uses flexible barriers known as phasers to relax synchronization within bounds. ZSim [13] is one of the latest parallel simulators that provides an alternative approach for synchronization which is called Bound-Weave. ZSim simulation runs in time quanta where each time quanta is defined as a small interval (e.g. 1000 cycles). ZSim divides each time quanta (interval) into two parallel phases: bound phase and weave phase. In the bound phase, similar to the lax synchronization, the cores are simulated without simulation of the interactions among the cores (i.e. unordered simulation), but the core-memory access traces for all the cores are recorded. In the weave phase, parallel event-driven simulation is performed by using the traces to simulate the memory accesses in order. Bound-weave is proposed based on the assumption that path-altering interference is extremely rare. This is a right assumption but only for cores that implicitly communicate through the cache hierarchy. Thus, bound-weave parallelization methods are not applicable to other communication styles (e.g. extremely fine-grained message-passing across whole chip). Moreover, simulating speculation (e.g. transactional memory) and complex workloads (e.g. kernel-intensive applications) would be difficult since ZSim is a user-level simulator. ZSim also provides limitations to model multi-threaded cores, detailed NoC models and virtual memory (TLBs).

In summary, we can claim that non-PDES-based simulators provide better scalability than PDES-based simulators, but they suffer from inaccuracy of simulation.

**Simulation Engine:** The simulation engine specifies the underlying strategy which each simulator uses to perform functional modeling. As we already discussed at the beginning of this section, the main responsibility of the functional model is the correct execution of the simulated ISA (i.e. emulation). But the ISA emulation might not be necessary when simulators directly use the host's ISA, instead, simulators can use instrumentation. This indeed eliminates the need for functional model for such simulators and increases the simulation speed with the potential cost of limiting the ISA of the target machines to only the host's ISA.

Depending on which strategy is used for the simulation engine, we classify the simulators into emulation-based (or interpretation-based) and instrumentation-based. An emulation-based simulator either uses its internal emulator or leverages an external simulator/emulator to model functional behavior. The simulator interprets the instruction and according to the simulator's organization, invokes both functional and timing model

to execute the instruction for the simulated ISA. Emulation becomes the primary and even the optimal option, particularly when the simulated ISAs are supposed to be different from the host's ISA, or when the simulators are expected to be portable. On the other hand, instrumentation, is faster than emulation, since the instructions are directly executed on the host machine. The simulator adds instrumentation calls to the simulated binary in order to interact with the timing model by calling the timing model before each basic block or memory operation. Instrumentation provides facilities to understand the execution behavior of each instruction and measuring the execution performance on the host machine. It returns a set of useful information about the execution which can fed the timing model. This can be done by enabling transparent access to the state of host's processor and memory after each instruction execution. Furthermore, instrumentation can be very efficient, particularly when emulation and functional modeling become difficult for complex ISAs. Most of the current instrumentation-based simulators run on x86 hosts, since x86 is common in both desktop and server segments, and the ISA's impact is less relevant. Examples using emulation strategies include MARSS [6], Gem5 [4], SimFlex, COTSon and BigSim while simulators such as CMPSim [46], Graphite [11], PriME [14], ZSim [13] and McSimA+ [69] benefit from instrumentation strategies.

**Binary (Code) Translation:** Binary translation is the core technology for both emulation and instrumentation strategies through enabling translation techniques to translate binary codes from a simulated architecture to the host architecture. The main difference is that, in emulation, all the functional and execution behavior of the applications for the simulated architecture are modeled through a functional model. In instrumentation, they use instrumentation calls added to the translated binary. Thus, arbitrary statistics about the run-time actions of the executing application can be gathered from the host which can be used to specify the functional behavior of the application for the target architecture without needing a complete functional model.

Two main types of binary translation techniques are static (ahead-of-time translation) and dynamic (translation at run time).

In Static Binary Translation (STB), all the binary code of an executable file is converted into code that can be executed on the host architecture, and after that the translated code is run on the host. This might not be efficient, since discovering some part of the code may depend on the run-time values (e.g. indirect branches, dynamically loaded libraries and self-modifying code). Wisconsin Wind Tunnel (WWT) [70] is a simulator which uses static binary translation.

On the other hand, Dynamic Binary Translation (DBT), relies on on-line code translation which means that that the portions of the binary code (each short sequence of code or single basic block), are translated and executed one after another in the order; the code is only translated as it is discovered. DBT suffers from large amount of overhead during translation which leads to increased execution times. Code cache is a technique which reduces the translation overhead by caching the translated code sequences for later usage when subsequent executions of the same code region can use the already translated code.

Figure 8 presents the general framework of the DBT systems which includes four main components: dispatcher, just-in-time compiler (JIT), emulation unit and software-based code cache. The dispatcher coordinates translation and execution of the code through directing other components. It gets the address of the next program which is a segment of guest binary code and determines whether a translated copy of that code is available in the code cache. If so, the execution is resumed in the code cache, otherwise, the dispatcher kick-starts the JIT compiler to translate the untranslated guest code segment. JIT fetches the code segment, and then optimizes and translates the code to the host binary in the software code cache. Optimization can be performed by adding, removing, inserting or replacing instructions to the code before translation. Using this capability, JIT allows to inject various instrumentation instructions into code. Furthermore, the JIT's granularity to fetch and translate (i.e. amount of code which are proceed at a time) can be specified as a basic block, a trace, a treegion, or an entire procedure. The emulation unit is also responsible to handle exceptions and interrupts (such as I/O interrupts for full-system simulation and system calls for user/application level simulation) during execution.

DBTs enable virtualization across ISAs by emulating a guest binary executable code in one ISA on a host machine with a same or different ISA. Modern DBTs also employ dynamic recompilation techniques (e.g. just-in-time compilation). This way, the translated code is instrumented to return information about the execution of each portion of the code which in turn can be used to optimize the rest of the execution (e.g. incremental optimization of hot regions). DBT has been widely used for many different purposes and applications (such as performance optimization, debugging, profiling, performance motoring and application migration). DBT, depending on the simulated ISAs and the host's ISA, can be classified as simulation engine into three categories: same-ISA, cross-ISA and retargetable (see Figure 8). For the same-ISA, the simulated ISA is identical to the host's ISA. In cross-ISA, guest ISA differs from the host ISA. In retargetable DBT, the guest ISA (simulated ISA) can be retargeted for multiple different ISAs. Both same-ISA and cross-ISA can be considered as dedicated DBTs, since the guest and host architecture are fixed. Dedicated DBTs are limited by assuming that the register set of the host architecture is the same or richer than the guest architecture. This causes lack of translation flexibility and adaptation to highly heterogeneous environments.

Many of the current instrumentation-based simulators such as Graphite [11], PriME [14] and ZSim [13] benefit from DBT tools and libraries which leverage same-ISA DBT on x86 host architecture. Examples of these DBT systems include Pin [71], StarDBT [72], and DynamoRIO [73]. Emulation-based simulator generally leverage cross-ISA and retargetable DBTs to perform virtualization and emulation. IA-32 [74] and FX!32 [75] are examples of cross-ISA DBT systems. There are few DBTs in the state of the art which have been designed for retargetability. Among them, QEMU [30] is a well-known emulator which implements retargetability. It enables binary translation from several different guest ISAs such as x86, PowerPC, ARM and SPARC on multiple common host architectures such as as

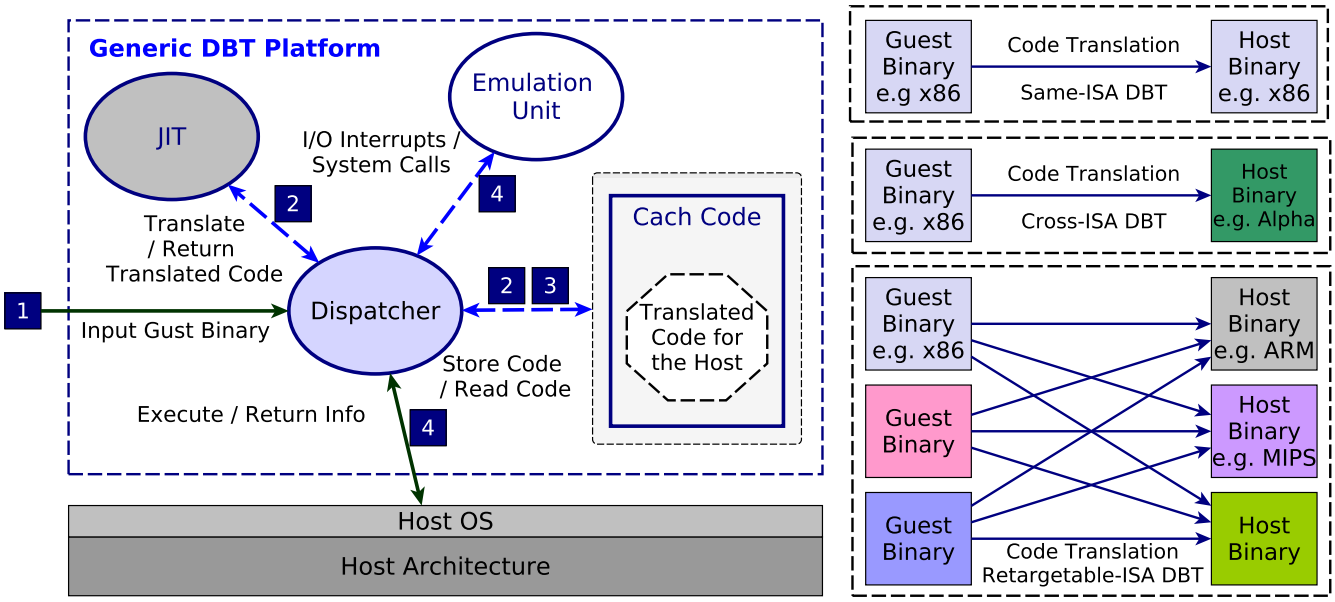


Figure 8: General DBT Framework and DBT Types

ARM, SPARC, Alpha, x86, MIPS and PowerPC. Retargetable DBTs broadly suffers from large emulation overhead ahead of translation, since the guest architecture must be fully virtualized by software in the memory, prior to translation. They also suffer from optimization overhead as well as code optimization overhead. Furthermore, providing near native and high quality translated code is another challenging issue for retargetable DBTs.

**Target Memory Architectures:** Depending on the memory architecture of the target simulated systems, manycore simulators can be categorized into shared memory, distributed memory and distributed shared memory.

In shared memory architecture, all the simulated processors simultaneously have access to a common single memory space in order to avoid data redundancy and facilitating inter-processor communications. In this way, the processors do not need to know where data resides, but they might suffer from race conditions. In distributed memory architecture, each simulated processor has its own private memory. Whenever communication is required (e.g. requesting a remote data), the simulated processors can communicate with each other through simulating message passing techniques on top of the simulated network and interconnect. Thus, it is necessary for every distributed memory simulator, to provide capability for network modeling (such as on-chip network). For example, RAMP-Gold [76] is a shared memory simulator which doesn't provide any network model. But distributed memory simulators such as COTSon [23] and BigSim [10] support network modeling.

Distributed memory simulators are more flexible than shared memory simulators for parallel simulation, since every simulated processor (i.e. a core) can independently be run on a different processor of a different host in the system. In distributed shared memory architecture, each node of a cluster has access to a shared memory in addition to its non-shared, private memory. Distributed shared memory simulators (e.g. SimK [52] and McSimA+ [69]) are generally required to support complex models of memory subsystems (i.e. memory and cache hierarchies) as well as communication networks.

**Implementation:** Depending on the strategy to implement functional and timing models, simulators can be classified into software-assisted simulators (i.e. simulators based on software simulation of the target architectures on CPU/GPU hosts), hardware-assisted simulators (i.e. simulators based on implementation of the target architectures on FPGA hosts) and hybrid simulators (i.e. simulators based on combination of both software and hardware simulation).

Most of the current architectural simulators are software-assisted, since software simulation is low cost, easy to develop, and flexible to explore various target architecture. COTSon, GEMS, SystemC [77], SimFlex and BigSim use software based simulation on CPU for both functional and performance models. In [78], authors introduce a software based simulator which runs on both CPU (by leveraging QEMU for functional modeling) and GPU (by proposing a GPU-based manycore accelerator). The work presented in [79] is another recent example of software based simulators. It leverages QEMU for software based functional modeling on CPU while uses a performance model, written in C and CUDA, designed to execute partly on the host CPU and partly on the host GPU.

On the contrary, hardware-assisted simulators are high cost and not flexible for architectural design exploration. They also take significant amount of time, memory and effort to develop and use, but they provide faster simulation speed compared to software-assisted simulators. Examples include RAMP Gold [76], that uses FPGAs for both functional and timing models. ProtoFlex [80] and FAST [18] are examples of hybrid simulators,

where they use FPGAs to accelerate the performance model while using software based simulation for functional models. Furthermore, hybrid simulators like HASim [81] use FPGAs for functional modeling and CPU based software simulation for performance modeling.

**Sampling:** Sampling and reduction techniques enable faster simulation of large scale execution workloads through simulating representative portions of execution (i.e. samples) [82]. In this way, architectural simulators can be classified based on their sampling strategies. There are various methods for execution sampling in the current literature. Examples include random sampling [10], dynamic sampling [23] and statistical sampling [47, 83] such as SimPoint [82] and Sampling Micro-ARchiTecture Simulation (SMARTS) [83]. Sampling techniques might reduce the simulation accuracy and precision in the estimate.

Assuming that a population is a complete set of elements (e.g. execution instructions of a complete workload) which need to be simulated. Sampling means simulation of a sample workload, instead of a complete workload. Each sample includes a subset of population (i.e. a set of sampling units) where a sampling unit is defined as a quantum of population. An ideal sampling approach must create a sample with choosing a minimal but (highly) representative set of sampling units in order to provide a reliable and quantifiable simulation (in terms of accuracy and speed).

A random sampling is performed by selecting and simulating only a random sample from the entire workload (the whole population) through a random selection of a fixed number of sampling units. The number of selected sampling units (or the sample size) must be sufficiently large for obtaining reliable (in terms of accuracy and precision in the estimate) simulation results (close to complete workload simulation). In a systematic (or periodical) sampling, sampling units are chosen from a population at periodical intervals. A uniform sampling, perform this with equal selection probability for (all) sampling units distributed in the entire population. In a representative sampling, sampling units are chosen from weighed regions of the population. In a dynamic sampling, the sample intervals (and the duration of each interval) are determined dynamically through monitoring the variation of execution behavior for the given benchmark [84].

Statistical sampling attempts to estimate (simulate) the execution behavior and characteristics of a given workload (i.e. a population) by selecting and simulating an optimal sample. This can be done by examining the adequacy of potential samples over the entire population using priori profiling (preprocessing), consisting statistical analysis (for example in terms of total, mean, and proportion) of the code. In other words, an statistical sampling prescribes a specified and constructive mechanism to determine an optimal sample (including a large number of tiny sampling units) in order to obtain a desired confidence interval. The sample must be able to capture the inherent variations of the given stream of execution instructions [85]. Statistical sampling is an efficient approach in capturing average behavior. And also it can specially provide an appropriate sampling when the population benefits from a clear internal control and contains a large number of similar transactions. However, it might be more complex and time consuming compared to non-statistical sampling approaches [86, 87].

### *3.2. Virtual Machines*

Systems based on Virtual Machine (VMs) hypervisors are typical tools to model the hardware. These provide an emulated environment where the guest applications (particularly operating systems) can be executed virtually, very close to the way that they are executed on real hardware. Generally, virtual machines are faster than architecture emulators. This happens because the hypervisors create environments where substantial amount of instructions execute directly on the real underlying hardware without any overhead. VMs also permit execution times near to native speed. However, without support for hardware virtualisation, performance may be severely degraded, and not all currently available processors support virtualisation. They should meet a set of requirements which are elaborated in [88]. Of course, these can only be used when the system to test already exists, precluding the usage of VMs for many design space exploration tasks.

### *3.3. Architecture Emulators*

Architecture emulators are software tools which emulate the behavior and characteristics of a given CPU. They do not directly run the instructions of the emulated processor on the underlying real host processor. Rather, they employ methods of interpretation, or of dynamic translation, to translate the emulated instructions to a corresponding set of instructions for the target platform. Then, they execute the translated instructions on the hardware of the host platform. Therefore, architecture emulators generally decrease execution speed, but dynamic translation, a cache intensive technique, is used to enhance the speed.

Definitely architecture emulators are appropriate tools to be embedded in manycore full simulators for the purpose of functional modelling due to their accuracy, and reasonable speed. Qemu [30] and SimNow [33, 89] are common examples of architecture emulators which have been employed in several full-system simulators.

Qemu is an open source full system emulator which uses the dynamic translation technique. It provides capabilities to emulate several types of CPU architectures (x86, PowerPC, ARM and Sparc) on several different hosts (x86, PowerPC, ARM, Sparc, Alpha and MIPS) to virtually run a complete and unmodified operating system. It also offers three operation modes: full system emulation mode, where the processor architecture and other peripheral devices are emulated; user mode emulation, where Qemu can launch executables compiled for one processor on another host processor which could have a completely different architecture; and finally Qemu acceleration mode, which executes most of the code directly on the hardware without dynamic translation, resorting to different execution rings.

AMD SimNow is a fast cycle, accurate full system emulator, using caching and dynamic compilation techniques. It can support booting an real operating system and launch complex applications over it. The SimNow emulator supports the x86

and x86-64 instruction sets, with support for other devices of a real system. It performs emulation of the real system with (at least) 10x slowdown in comparison with the native execution. SimNow cores generate a series of event that are stored in the asynchronous queues. COTSon provides timing feedback for SimNow instances, it parses asynchronous queues to create higher level objects such as instructions. COTSon is a complete tool which provides timing information back to the functional emulator in order to affect the behavior of the application. It also uses quantum based simulation [23, 59] as synchronization technique. Quantum is the smallest, atomistic timing entity. i.e. the dimension of length ("time") as a single entity is a quantum. In this way, every time a quantum starts, the timing module will get a notification about the staring time and the quantum length. Similarly once a node ends a quantum, the timing module will let the other modules to know about the network timing information which has been calculated during the past quantum. The functional simulation adds extra latency to all packets submitted, because network packets are sent twice. The source Network Interface Card (NIC) sends packets to the mediator timing module and the mediator will send the packets to the target NIC module. Additional time is required to for the packet processing in the mediator, therefore COTSon uses a Quanta (Q) bigger than real latency time between two nodes [90].

### *3.4. Network on Chip, Thermal and Energy Aspects*

Most full-system simulators can simulate the interconnection network (particularly the Network On Chip (NoC)) as well as the entire processor and memory system. However, due to inherent limitations, they are not able to perform NoC simulations, in a very detailed level [91]. ASIM [92] is an example of this case. It is a full-system simulator, used in industrial laboratories, that simulates the processing cores and memory hierarchy. It only models ring interconnects and it cannot model other interconnects such as mesh, hypercube, x-tree, shuffle exchange, fully connected, butterfly, cube connected cycles, etc. RSIM [93] and SESC [94] are other examples of full-system simulators, whose interconnection network is not modeled precisely in these frameworks. On the other hand some of the simulators support modeling of the entire system, including the on-chip network, to a significant degree of detail, such as PharmSim [95].

For the purpose of designing new manycore simulators and particularly with the objective of modeling the future peta-scale parallel machines, we should consider a very efficient and accurate network-on-chip model. NoC is an integral part of the memory system and not modeling its details leads to an unsatisfactory model. Today, there are several NoC only simulators like NOXIM [96] and SICOSYS [93] which are used by the NoC community for experiments. However, these are not able to be used to perform full-system modeling and simulation.

In the current literature, SystemC [77] (refer to Section 4.3), a simulation framework, has been widely used to design full system simulators and particularly for NoC simulators. SystemC-based NoC simulators [97–103] are powerful tools to evaluate different NoC configurations by means of simulation. Xpipes [103] is an example of this type. It provides capability to simulate both homogeneous/heterogeneous NoC architectures for multiprocessor SoCs through using a set of flexible SystemC-based NoC macros, enabling to act as instance-specific network components at instantiation time.

One of the efficient solutions to enhance the capability of a full-system simulator is leveraging other accurate and high-performance network on chip simulators for NoC modeling instead of using a weak built-in network model. It is required for each of the simulator component and tools to provide modularity. SICOSYS [93] is an example which has been plugged into RSIM [36] for simulating symmetric multiprocessor systems.

Energy consumption in uniprocessor architectures grows linearly with the clock rate frequency and quadratically with voltage. Usually, lowering frequency permits operation at smaller voltages, and this has a cubic effect on energy savings. As for multicore and manycore processors, the power consumption increases linearly with the number of cores while clock frequency increases at a much slower pace [104]. Further, the use of identical processing elements in an homogeneous architecture reduces the overall hardware complexity and verification process in the entire development cycle.

Other than power consumption, thermal issues have a significant impact on the performance of manycore architectures. Simulating the thermal behavior of the manycore processors is an important objective in many different aspects. The architecture designers need to know and analyze the impact of their designs on the temperature of the processors. Similarly, thermal modeling is required by the OS and system designers. The problem is that this kind of modeling is computationally expensive. In processors, thermal time changes in time interval of the order of ten milliseconds. Thus, to simulate a very complex thermal impact of a proposed design, it is required to simulate the system in a high level of details at least for several seconds.

In manycore architectures, thermal status of the individual cores are related to the actual computing workloads of each core. Thus, analytical [105] and mathematical models as Fourier expansion [106] can be used to characterize and model manycore processor workloads and foresee the accurate amount of the processors load.

In the current literature, various tools and simulators have been proposed to investigate power consumption and energy efficiency of microprocessors and manycore architectural designs. Examples include Wattch [107], SimplePower [108], SoftWatt [109], XTREM [110], Orion [111], Orion2 [112], McPAT [113], Sim-PowerCMP [114], PrEsto [115], Sniper/McPAT [116], Manifold [59] and [117], providing capability for power or thermal modeling of computing processors. However, since physical modeling (power, energy and thermal) of manycore systems is out of the scope of our interest in this paper, we do not discuss the details of those approaches here.

## 4. Architecture Simulators

Parallel and distributed simulation tools such as SimFlex [47], GEMS [26, 118], FastMP [42], SlackSim [43, 60], BigSim [10] and COTSon framework [23, 89], have been created in order to enhance the performance of simulation by concurrently distributing the simulator workloads among several parallel hosts.

Furthermore, running parallel manycore simulators in accelerated hardware platforms such as FPGAs, and more recently in general purpose GPUs, helps to increase the throughput of the system simulation.

Architectural simulators provide capability to simulate either the target microprocessors, or the full hardware and software functionality of the target machines, as platforms, where platforms may consist of parallel processors, memory hierarchy, storage devices, I/O devices, compiler, OS, etc. Along the line of the growing issues in manycore era architectures, simulators have changed and they used new approaches to solve the problems and improve the system efficiency. They started from a simple approximate uniprocessor simulator, and continually improved to the recent high efficient clustered simulators.

Table 2 provides a generic comparison of architecture simulators and other simulation tools. However, this comparison is very general and depending on each specific tools it might differ. The table shows that simulators (both types) are the more flexible tools to analyze and predict the behavior of future systems. Furthermore, emulation-based simulators are the most powerful tools, since they can potentially simulate any type of architectures. On the other hand, instrumentation-based simulators are faster, since they provide near-native execution speeds.

In the remaining of this section, we discuss some of the most important architecture simulators that have been proposed in the last decade.

### *4.1. SimpleScalar*

SimpleScalar [9] is one of the oldest uniprocessor serialized software simulators, which previously was widely used by the research community focused in processing architectures. As well as other similar simulators, SimpleScalar was limited to only run single-threaded, user-mode workloads. With the advances in manycore micro-architectures, researchers and designers are more interested to use simulators with the capability to run multi-threaded workloads, and to model large number of processing cores along with the memory subsystem and interconnects.

### *4.2. BigSim*

BigSim is a parallel simulator and performance modeling system which is particularly designed to study parallel programming issues [10]. It can predict performance of parallel applications on machines (like IBM Blue-Gene/L systems) [119] with a very large number of processors (i.e. large number of processing nodes). This is done by actual execution of real applications on smaller machines (i.e. small number of processing nodes). Indeed, BigSim can be used for the architecture design of manycore-enabled HPC systems. It has been built on POSE [120] and includes several components. POSE is a general-purpose optimistically synchronized PDES (parallel discrete event simulation) framework which is designed for scalability of fine-grained parallel and distributed large-scale discrete event simulations. However, its load-balancing framework still needs to be improved. BigSim simulates the behaviors of communication and computation separately in two steps. At first, it uses an emulator to execute an application, containing number of virtual processes, on a small number of physical processors to generate trace logs. At the next step, a trace-based simulator uses the log files and simulates activities on a much larger processing system.

BigSim directly executes the application (in small scale) using its emulator and mimics the behavior of the target platform (in large scale). The direct execution creates significant demands on host CPU and memory. For this reason, the simulator allows skipping computations and instead simulates latencies that would be resulted by executing those computations. But the problem is, this works only for data-independent applications, due to the fact that some part of the data are not really computed. The simulator explores the inherent determinacy of several parallel applications. But still it is not an application-independent performance modeling system and its functionality is limited to specific applications. Moreover, tracing in BigSim is specifically designed for its implementation language Charm++, and doesn't support the message passing applications.

BigNetSim, a BigSim's component, uses a simple analytical model (e.g. SimGrid [121]) to simulate interconnection networks by supporting detailed network models of various topologies. However it still doesn't provide capability for packet-level interconnections simulations (e.g. MPI-NetSim [122]) which is the most precise approach (while it is very resource-consuming) for network simulation. This leads to reduction in the overall system accuracy. In fact, the major drawback of BigSim is that it suffers from inaccuracy (with respect to the expected behaviors of the real system) caused by log-based postmortem simulation of the generated traces.

### *4.3. SystemC*

The SystemC language [77] is an extension of C++, providing a cycle-accurate, event-driven, simulation interface for system-level modeling by describing modules of a target architecture as a set of C++ classes. It is a popular framework for SoC architecture simulation, providing a powerful interface to describe HW/SW components as well as interconnections between modules (ports and signals), facilitating description of interconnection between multiple SoC processors. SystemC also provides support for integrating different Instruction Set Simulators (ISS) in a unified system simulation framework, as it is able to plug an independent ISS into the entire simulation framework (as a new system module), where all system modules can be activated and synchronized through a common reference clock. Furthermore, SystemC based simulators (such as [123–127]) can benefit from advantages of C++ language as a hardware description language while bridging the gap between hardware and software description languages [123]. In fact, one of the most important advantages of C/C++ based description languages such as SystemC (or SpecC [128], a similar alternative) is their capability to concurrently specify both hardware and software components in the design (i.e. co-simulation of both hardware and software). This is a necessary requirements for full system simulators (system level).

In the current literature, SystemC has also been extensively used for designing NoC simulators, due to the powerful capa-

Table 2: General Specifications of Simulators, Emulators and Virtual Machines (Execution Info: Information about the execution behaviour, Functional Results: Execution Results)

| Tools | Engine | Code Translation | Output Information |
|---|---|---|---|
| Virtual Machines | Mainly Direct Execution | Same-ISA DBT | Functional Results |
| Emulators | Emulation (Interpretation) | Cross-ISA & Retargetable | Functional Results |
| Emulation-based Simulators | Emulation (Interpretation) | Cross-ISA & Retargetable | Functional Results, Execution Info |
| Instrumentation-based Simulators | Instrumentation | Same-ISA DBT | Functional Results, Execution Info |

bility of SystemC to describe various interconnections between hardware components. Examples of this include [97–103].

SystemC based simulators are accurate and sufficient for validating hardware specifications. However, they might fail to adequately support embedded software (in terms of writing or debugging), which is an important requirement for SoC design [129], are often slow compared to the traditional ISSs like SimpleScalar.

MPARM [123, 124] is an example of such simulators. It is a full-system SystemC-based architecture simulator, enabling to model functional, performance, and power consumption aspects, as well as a complete OS for a Multi-Processor System-on-a-Chip (MP-SoC), in a cycle accurate manner. Using SystemC, MPARM (or MP-ARM) can provide processor models, memory models, the AMBA bus architecture (for communication between models through ports and signals) and support for parallel programming. However, the MPARM simulator is slow and its models for processing cores are very abstract and relatively simple, lacking detailed core modeling [130].

One of the conventional solutions to overcome the speed limitation of SystemC-based simulators is to integrate SystemC with QEMU, making possible to simultaneously benefit from accuracy of SystemC and speed of QEMU. However, interfacing between SystemC and QEMU might be challenging, since their combination must be capable of accessing all the hardware modeled in QEMU and SystemC for co-simulation of HW/SW. For doing this, SystemC needs the QEMU support to provide I/O operations (initiated by the processor), memory access interface, interrupt handling and also peripherals to access memory directly [131]. Furthermore, there are timing aspects which need to be taken into account for synchronization between SystemC and QEMU models [132], making SystemC-QEMU combination a complex task. There are several research works in the current state of the art, which perform this using different approaches [131, 133–136]. Among these types of simulators, Virtualsoc [8, 137] is a recent work for many-core-based accelerators, allowing the execution of a full-fledged Linux operating system.

### *4.4. Graphite*

Graphite [11] was created for the exploration of large-scale manycore environments, as well as for research of isolated applications. It can be used as a distributed, high-level parallel simulator. In order to deliver the high performance and scalability needed for useful evaluations, it uses various methods such as direct execution, multi-machine distribution and analytical modeling. In addition, it benefits from lax synchronization schemes like LaxP2P [138–140] (a distributed synchronization technique, in which the progress of one core is periodically checked against another randomly selected core). Graphite has other important capabilities, such as its flexible and extensible architecture, its compatibility with commodity multicores and clusters, its ability to run off-the-shelf p-threads application binaries, and its support for a single shared simulated address space despite running across several physical host machines.

Graphite, unlike BigSim, FastMP and COTSon, allows the analysis of a much wider category of architectures. While it offers the possibility to model distributed memory architectures, it also provides a coherent shared memory between the simulator threads [11]. In addition, Graphite also models compute cores and interconnected networks while operates transparently through providing a single shared address space to off-the-shelf applications.

Graphite can simulate manycore target architectures with hundreds of cores launched on several parallel hosts, but the problem is that the simulator accuracy resides in the application-level. It is not very successful to deal with speed/accuracy challenges. Graphite has three different methods for synchronization: base model (or Lax synchronization), barrier (LaxBarrier or Lax with quanta-based barrier synchronization) [140] and random-pairs (LaxP2P or Lax with point-to-point synchronization). Lax lets the clocks differ and offers the highest performance and scalability. However, in order to keep the simulated clocks in reasonable agreement, Graphite needs to deploy application events to make them synchronized, otherwise it must let the threads run freely.

LaxBarrier and LaxP2P [58, 140] are the mechanisms on top of Lax to improve its accuracy. LaxBarrier is the most accurate synchronization methods of Graphite where all active threads must wait on a barrier after a configurable number of cycles. It has a relatively poor performance and scalability compared to the other two synchronization models. LaxP2P aims to achieve the accuracy of quanta-based LaxBarrier without reducing the scalability and performance of lax synchronization. Using this scheme each tile periodically chooses another random tile and synchronizes with it. If the clocks in the source and target tiles differ by more than the number of configured cycles then the tile which is ahead goes to sleep for a short period of time. LaxP2P is fully distributed therefore it creates less overhead than LaxBarrier. In comparison to the barrier method it provides more scalability and less accuracy [11].

### *4.5. SimK*

SimK [52] is a framework based on the Parallel Discrete-Event Simulator (PDES) synchronization protocol [141] to develop system simulators. PDES [53] is a well known parallel distributed synchronization technique for parallel simulation.

SimK provides simulation modules that target system components, such as CPUs and memory modules. All modules communicate through message passing methods, which enables them to run concurrently. A dedicated module maintains the time synchronization of all simulation modules. P-GAS [142], HPP-NetSim [143] and G-Cluster [144] are the simulators which have been developed based on the SimK framework.

Since each component has to synchronize the execution state with its peers, continuously at a microsecond rate, SimK employs several optimization strategies and techniques to avoid the severe performance degradation that synchronization would impose. Each node is handled by a single process, which further creates one thread per processor. A user level scheduling scheme is employed where simulation modules are dispatched to each thread. CPU affinity is used to avoid cache related performance penalties. Since the simulation modules run on the same process and share the same memory, SimK employs an asynchronous zero-copy [52] communication mechanism. Further synchronization optimizations are employed at the scheduling level to avoid blocking of the simulation modules. Other optimizations employed include lock-free queues, buffer management and load balance.

SimK requires a host shared-memory multiprocessor system. While it has shown to scale within this system, the lack of cache coherence on manycore systems does not allow the shared-memory dependent approach used by SimK to be effective. Thus, the major bottleneck of this simulation framework, which is synchronization, cannot be solved with the approach taken by SimK on a manycore system. This severely limits the scalability of SimK to multiprocessor systems.

### *4.6. GEMS*

GEMS [26, 118] is a full-system simulation platform capable of capturing detailed aspects of processing cores, cache hierarchy, cache coherence, and memory controllers. The simulation platform consists of a set of nodes connected with links allowing for wide variety of topologies, with each link having a particular latency and bandwidth. This has led to the widespread use of GEMS in the computer architecture research community, with a huge amount of contributions for validating research ideas.

A major limitation of GEMS is its simple interconnection model that serves as a communication fabric between various cache and memory controllers. Messages traverse the network hop by hop, which makes GEMS incapable of modeling a detailed router or a network interface. In fact it does not integrate a real interconnection network model [145]. Because of this limitation, GEMS ignores buffer contention, switch and Virtual Channel (VC) arbitration, realistic link contention and pipeline bubbles. The GEMS interconnect SimpleScalar model also assumes perfect hardware multicast support in routers. However, considering on-chip network designs, supporting fast and low power hardware multicast is currently still a challenge.

The limitations in the interconnect model can significantly affect the results reported by the current GEMS implementation. Thus, GEMS has not been adopted by researchers focusing on low-level interconnection network issues. Researchers instead rely on traffic trace-driven simulation with synthetic or actual traces. In a trace-driven approach, the functional simulation is not impacted by the timing simulator. Timing dependent effects are not captured because the trace is generated a priori on a fixed system, and the timing variation caused in the network does not affect the message trace. Trace-driven techniques also do not capture program variability that a full-system evaluation can.

### *4.7. SimFlex*

SimFlex [5, 47] is a full system, component-based simulator, inspired by the ASIM simulator [92], which enables creation of timing models for uni and multiprocessor systems running unmodified commercial applications. It integrates SMARTS methodology [83] for simulation sampling and Simics [34] for functional modeling with techniques to avoid runtime overheads that arise from component-based software design. Simics is a high configurable simulator with a basic timing model (i.e. uniform timing for all instructions and memory accesses). It can provide functional execution of unmodified commercial OSs and applications for wide variety of systems and ISAs (e.g. x86, SPARC, etc.). SimFlex enables full system simulation by augmenting Simics with a framework for rapidly building complex timing models. SimFlex receives a stream of fetched instructions from Simics, and then models the timing behavior of the system while controls the timing advancement in Simics.

SimFlex is a collection of C++ described components connected together in a hierarchical fashion where each component represents a hardware or software component of a real system. Each component is connected to other components by definition of ports (i.e. unspecified C++ template parameters during component development). The connections between components can be configured and specified in a C++ code which is called wiring description. And accordingly, when these wiring descriptions are fed to the compiler, the interconnection between component is created at compile-time. This way, SimFlex framework can produce a custom timing simulator binary which reflects the desired wiring description. Examples of SimFlex timing models include UniFlex (uniprocessor simulation ), TraceFlex (no timing), CMPFlex (in-order timing) and CMPFlex.OOO (out-of-order timing).

Most of applications exhibit homogeneous execution phases which can include millions of instructions. Statistical sampling reduces the amount of simulation effort required for performance estimation of such applications and leads to increase the simulation speed. SimFlex, using SMARTS rigorous statistical methods, identifies the minimal sample that assesses application performance with a required confidence level.

SimFlex offers detailed multiprocessor memory systems but lacks detailed I/O models and multiple-system capability. Furthermore, SimFlex suffers from the lack of flexibility for development, particularly for non-academics, since it relies on Simics, which is a commercial functional simulator with licensing restrictions.

### *4.8. COTSon*

COTSon is the HP labs' full system simulator based on AMD SimNow. It allows researchers to trade speed for accuracy, depending on the simulation purpose and the user preferences. It

uses a trace driven approach where a single core, full system, simulator generates thread instruction streams. This helps COTSon achieve better simulation speeds, compared to execution-driven simulation, because of the decoupling of functional simulation from detailed simulation. It also is possible to simulate any application that might have been run on different platforms, given the correct tracing infrastructure.

Functional simulation generally adds some extra latency to every transmitted packet. It happens because packets are sent to the mediator and then the mediator sends them to the destination. COTSon performs bandwidth and network simulation in the sender NIC device, but all the network timing characteristics and information is collected at the mediator level. Potentially, COTSon can present reasonable speedups in comparison to some of other simulators, but it assumes an idealized architecture consisting of a perfect memory hierarchy, which is a little far from real architectures.

COTSon leverages some of the other existing simulators and tools for individual sub-components through a robust interface layer, integrated closely with the COTSon timing models to improve simulation accuracy (versus simulation speed). Notable examples are SimNow, for the purpose of functional modeling, and Q-Mediator, for simulation of interconnected network. However, COTSon does suffer from using SimNow, which is a sequential emulator.

SimNow produces a sequential instruction stream as output which is demultiplexed into different threads before timing simulation. It creates a significant drawback for COTSon, limiting parallelism and restricting the simulator to a single host machine for shared-memory simulations. On the other hand, COTSon is able to launch simulations over an overlay network consisting of multiple manycore machines. But, it becomes limited to run applications which are created for distributed memory environments and use a message-passing library (like MPI [146]).

### *4.9. RAMP-Gold*

RAMP-Gold [147] performs system modeling by employing a single timing pipeline, coupled with a single functional pipeline with moderate resource consumption launched on a low-cost mid-size FPGA. The simulator design is constrained in the total amount of cache capacity that we can model by using the BRAM consumption of the timing model [76, 148].

FPGAs include Block RAM (BRAM) and Distributed RAMs (DRAM). Block RAMs or BRAMs are dedicated memory blocks and DRAMs are the RAMs that are constructed using Look-Up-Tables (LUT). LUTs are distributed across the FPGA fabric and they can be used as small blocks of RAM by combining DRAMs. RAMP Gold has the capability to simulate 64 SPARC CPUs over 250 times faster than a regular software-based system simulator launched on a Xilinx Virtex-5 board (which is low-cost in comparison to other FPGA hardware). This demonstrates the cost performance benefit of FPGA-based simulation. The design of RAMP Gold also shows that designing FPGA-based architecture simulators is dramatically different from designing multicore processors in either ASICs or in FPGAs.

### *4.10. Other FPGA-based Simulators*

FAST [149] is a hybrid FPGA-based simulator, whose functional model runs in software and its timing model runs in FPGAs. It needs a significant amount of communication bandwidth between FPGA and CPU, which may result in limited simulation scalability. ProtoFlex [150] and HASim [81] are other simulators which use FPGAs to implement timing models for full-cycle-accurate simulations similar to FAST. Unlike FAST and ProtoFlex, HASim implements its functional model in FPGA and timing model in CPU.

In general, FPGA-based solutions are costly and require the user to buy expensive hardware. In addition, FPGA-based approaches are difficult for development and it is not easy to quickly experiment with various designs while implementing new models in FPGA instead of CPU.

### *4.11. GEM5*

GEM5 [4] is a modular, sequential, full system simulator, based on a combination of both M5 [29] and GEMS [26] simulators. It merges the high configurablity of M5 with high detailed and flexible memory subsystem modeling of GEMS. This leads to the support of a wide range of simulation features ranging from multiple ISAs (such as Alpha, ARM, MIPS, Power, SPARC, and x86), diverse CPU model (non-pipelined and pipelined CPU models such as AtomicSimple, TimingSimple, In-Order, and Out-Of-Order), detailed cache hierarchies (using Ruby memory model) and multiple cache coherence protocols (using SLICC language) to the instantiation of various interconnection networks (Ruby based network models such as Simple network model and Garnet [151] network model), I/O devices (ranging from simple timers to complex network interface controllers) and multiple systems. Despite the high accuracy and flexibility of GEM5, it suffers from very slow simulation speed due to the lack of parallelization in simulation.

### *4.12. McSimA+*

McSimA+ [69] provides a lightweight, flexible, and detailed microarchitecture-level simulator by offering a middle ground between a full-system simulator and an application-level simulator. It can benefit from the light weight of an application-level simulator, while the simulator is able to fully control the threads and processes (similar to a full-system simulator). McSimA+ can simulate x86-based asymmetric manycore systems (up to more than 1,000 cores) for both core and uncore subsystems. The modeling for asymmetric cores can be ranged from single-threaded to multi-threaded workloads and from in-order to out-of-order CPU models. It also supports sophisticated cache hierarchies, coherence hardware, on-chip interconnects, memory controllers, and main memory. Furthermore, using DBT as simulation engine and the ability to support both execution-driven and trace-driven simulations, McSimA+ is able to reasonably improve both simulation speed and accuracy. However, McSimA+ suffers from the inherent limitation of application-level simulators, which is the lack of full support for OSes and applications with complex I/O system calls and extensive system events. Moreover, the simulation workloads for McSimA+ is limited to

only Pthread applications due to the frontend Pthread library of McSimA+. Non-Pthread multithreaded applications cannot be executed on McSimA+ without re-targeting the thread interface. In addition, the simulation accuracy of McSimA+ is most likely suboptimal as compared to most of the cycle accurate emulation based simulators. McSimA+ is also limited to support modeling of speculative wrong path executions due to the inherent limitation of its Pin [71] based functional modeling. In fact, Pin does not provide wrong-path instructions, since wrong-path instructions in Pin are invisible beyond the ISA interface and they are not committed in the native hardware.

### *4.13. ZSim*

The ZSim [13] simulator introduces three techniques in order to enable fast, accurate, and scalable simulation for manycore systems (up to 1024 cores). These techniques are: detailed DBT-accelerated core models, in order to increase sequential simulation speed; bound-weave parallelization, in order to achieve scalable and accurate parallelization; and lightweight user-level virtualization, in order to provide support for simulation of complex workloads. ZSim proposes instruction-driven timing models, based on Pin DBT, to perform most of the core's operations in the timing model during instrumentation (i.e. eliminating the need for functional modeling of x86). This reduces most of the FM-TM overheads compared to conventional cycle-driven or event-driven core models. It also leads to fast sequential simulation (between 20-90 MIPS) while the core modeling is detailed enough to allow simulation of detailed core models like Out-Of-Order (OOO), including features such as branch prediction, limited fetch and issue widths, and $\mu$op fission.

ZSim also introduces the bound-weave algorithm which is a two phase event-driven parallelization technique that scales parallel simulation without increasing overheads or loosing accuracy. The main insight behind this algorithm is that, at a small time scale, most concurrent core-memory accesses occur to different unrelated cache lines. This means that out of order simulation of these accesses at first and then simulating their detailed timing in order, can be equivalent as simulating them completely in order. This allows reorderings of instructions only within a small interval (e.g. within 1,000 cycles) with assumption that, in such case, path-altering interference is exceedingly rare.

A two core-memory accesses might suffer from path-altering interference, if out of order simulation of those accesses, modifies their paths through the memory hierarchy (e.g. two write for the same cache line from two different cores). ZSim divides the simulation into small intervals of a few thousand cycles each. As we already mentioned in Section 3.1, for each interval the simulator proceeds bound phase and weave phase. In bound phase, cores are simulated in parallel with assumption that all memory accesses have initially zero latencies and for each memory access, the path through memory hierarchy is recorded. The bound phase also puts a lower bound on the cycle of each microarchitectural event. Accordingly per-core event traces are generated through instrumenting all loads, stores and basic blocks of the host thread which simulates a core. In the weave phase, these traces are used to perform parallel microarchitectural event-driven timing simulation of the aforementioned memory accesses by dividing events among parallel event queues, and simulating them in full order.

While bound-weave parallelization is efficient to provide scalability for parallel simulation, it is only applicable for shared memory target systems. The reason is that the bound-weave approach relies on the assumption that path-altering interference being rare, which is true but only for the cores that communicate implicitly through the cache hierarchy. This means that bound-weave mechanism is not applicable for other target systems like distributed systems with message passing inter-core communications. Moreover, ZSim is a user-level simulator but since it is using a set of lightweight user-level virtualization mechanisms, it can run most modern workloads (such as client-server workloads and multiprocess applications) without any modifications.

### *4.14. PriME*

PriME [14] is an MPI-based, manycore, x86 simulator. It uses Pin as simulation engine and combines both shared memory and message passing techniques in order to achieve high parllelization and distribution of simulation workloads through running simulation across any MPI-enabled cluster of multiprocessor and multicore machines. This allows PriME to support multi-threaded workloads as well as multi-programmed workloads through benefiting from two levels of parallelization; within a host machine and across host machines. The simulation of multi-threaded workloads can be parallelized inside of a host machine while the simulation of multi-programmed workloads can be parallelized across several host machines. This happens by utilizing MPI to communicate among different PriME modules.

PriME demonstrates reasonable performance and speed for manycore simulation (for example, it can simulate more than 2000 cores on 9 machines with a total number of 108 cores). However, similar to other simulators, it also suffers from drawbacks. While the simulator supports detailed modeling of uncore components such as the memory subsystem and NoCs, it can only simulate simple in-order core models with a constant cycles-per-instruction (CPI) for non-memory instructions. Therefore, using PriMe, simulation of the detailed core models, like modern out-of-order processors, is not feasible. While PriME is fast enough to explore thousand-core architectures, it is not cycle-accurate. While PriME offers high configurability for uncore models such as memory subsystem, cache coherence and interconnect models, the configurability of the simulator for the core models is poor. For example, the simulated ISA is limited to x86 architectures.

### *4.15. Comparison*

Tables 3 and 4 provide comparative analysis of some of the most important multiprocessor and manycore architecture simulators with respect to the taxonomies presented in Section 3.1.

**Table 3.** A comparative analysis of various many-core architecture simulators.

| Characteristics | SimpleScalar (2002) | BigSim (2004) | GEMS (2005) | SimFlex (2006) | SimK (2009) | COTSon (2009) |
|---|---|---|---|---|---|---|
| Simulation Execution | Sequential | Parallel | Parallel (sequential functional simulators) | Parallel (parallel simulation of live points or flex points, sequential functional simulators) | Parallel, distributed | Parallel (sequential functional simulators) |
| Simulator Organization | Timing directed (execution driven, interpreters) and functional-first (trace driven) | Functional-first (trace-driven) | Timing-first and timing directed (execution driven) | Cycle driven execution model, timing-first for out-of-order multiprocessor simulation | Functional-first (trace driven) and Timing directed (execution driven) | Functional-directed (feedback-driven) |
| Simulation Scope | User/Application-level simulator | User/Application-level simulator | Full system simulator | Full system simulator | Full system simulator | Full system simulator |
| Simulation Engine | Emulation | Emulation | Emulation | Emulation | Emulation | Emulation |
| Target Memory Architectures | Shared memory | Distributed memory (cluster-based simulation) | Shared memory (with cache coherence support) | Shared memory | Distributed shared memory | Shared memory, distributed memory |
| Scalability | Very Poor | Poor | Poor | Moderate | Moderate | High |
| Accuracy | Cycle accurate (cycle timers) | Instruction-level/cycle-accurate | Cycle Accurate | Cycle accurate | Functional accurate | Cycle accurate |
| Complexity of Setup | Moderate | Moderate | High | Low | Moderate | Moderate |
| Cost | Low | Low | Low | Low | Low | Low |
| Many-Core Support | Uniprocessor | Multiprocessor is supported | Multiprocessor is supported | Multiprocessor and multi-core are supported | Multiprocessor and multi-core are supported | Supported (up to 1024 core) |
| Implementation | Software simulation | Software simulation | Software simulation | Software simulation | Software simulation | Software simulation |
| Basis | Embedded (functional: sim-fast, sim-safe, sim-profile, sim-cache, sim-cheetah, sim-BPred, functional with timing: sim-outorder) | BigSim emulator (functional simulation), BigNetSim (performance simulation) | Simics (functional simulation), Rubby (timing simulation) | Simics (functional simulation), SMART (performance simulation) | HppSim (performance simulation), HppSwSim (functional simulation), HppNetSim (performance simulation for interconnection) | SimNow |
| Modeling Features | Single-core CPU models from simple unpipelined processors to detailed dynamically scheduled micro-architectures with multiple-level memory hierarchies | Network models (detailed models of communication networks e.g., network modeling for large message-passing machines such as Blue Waters and BlueGene), uni-processor models | Memory (Ruby model for caches, cache controllers, memory controllers, and banks of main memory), Ruby model for interconnection network, Opal CPU model | CPU models (uni- and multiprocessor systems), CMP and DSM system models, aggressive out-of-order core tuned to produce high memory parallelism | CPU cores, memory without hardware cache coherence, network adapters, routers, switches, interconnection network | Compute cores, networks, memory, the OS and common devices such as disks, video, or network interfaces |
| Event Sampling | Not Specified | Random sampling of sequential execution blocks, Slicing, Miniaturization | Not Specified | Multi-threaded rigorous statistical sampling using Flex Points and SMARTS | Not Specified | Single-threaded dynamic sampling |
| Synchronization | Not Specified (only inter-stage latch synchronization for register update and memory load/store in sim-outorder) | Optimistic PDES (taking advantage of the parallel programs' inherent determinacy to reduce synchronization overhead) | Not Specified | SimFlex does not have an explicit synchronization component but synchronization works exactly as it would on a real SPARC machine. | Conservative PDES (fine-grained block/unblock-based synchronization) | PDES |
| Supported ISAs | ALPHA, PISA, ARM, PowerPC and x86 | x86 | SPARC-v9, x86, etc. | ALPHA, x86-64, IA-64, ARM, MIPS, MSP430, PowerPC, POWER, SPARC-v8 and v9, etc. | (on CPU)MIPS64, MIPS III | x86-64 |
| Workloads | Single-threaded workloads (realistic applications with relative simplicity and less efforts) | MPI, CHARM++ and Adaptive MPI applications | Not Specified | Not Specified | Not Specified | MPI applications |

**Table 4.** A comparative analysis of various many-core architecture simulators.

| Characteristics | Graphite (2010) | RAMP-Gold (2010) | GEM5 (2011) | McSimA+ (2013) | ZSim (2013) | PriME (2014) |
|---|---|---|---|---|---|---|
| Simulation Execution | Parallel, distributed | Parallel | Sequential | Parallel | Parallel (Bound-weave) | Parallel, distributed |
| Simulator Organization | Timing directed (execution driven) | Multithreaded FAME (FPGA Architecture Model Execution) | Timing directed (execution driven) | Supports both execution-driven and trace-driven, event-driven backend simulator | Timing directed (execution-driven), (instruction-driven core, event-driven uncore) | Timing directed (execution-driven) |
| Simulation Scope | User/Application-level simulator | Full system simulator | Full system simulator | A middle ground between a user/application-level simulator and a full system simulator | User/Application-level simulator | User/Application-level simulator |
| Simulation Engine | DBT | Emulation | Emulation | DBT | DBT | DBT |
| Target Memory Architectures | Coherent shared memory across a cluster of machines, distributed memory | Shared memory (no coherence) | Shared memory (with cache coherence support) | Shared memory (with cache coherence support), distributed memory (cluster-based simulation) | Shared memory (with cache coherence support) | Shared memory (with cache coherence support), distributed memory (across a cluster of machines) |
| Scalability | High | Moderate | Poor | Moderate | High | High |
| Speed | Very Fast | Fast | Very Slow | Moderate | Fast | Fast |
| Accuracy | Not cycle accurate | Cycle accurate | Cycle accurate | Cycle accurate | Not strictly cycle accurate | Not cycle accurate |
| Complexity of Setup | High | High | Low | Moderate | High | Moderate |
| Cost | Moderate | High | Low | Low | Low | Low |
| Many-Core Support | Supported (up to 1024 cores) | Supported (up to 64 cores) | Supported (up to 64 cores) | Supported (up to 1024 cores) | Supported (up to 1024 cores) | Supported (up to 2048 cores) |
| Implementation | Software simulation | FPGA implementation | Software simulation | Software simulation | Software simulation | Software simulation |
| Basis | Pin | Embedded | M5 (processor simulator), GEMS (network and memory simulator)-> Ruby (cache and simple networks), Garnet (complex networks), SLICC (coherence protocols) | Pin based frontend simulator for functional simulations and the event-driven backend simulator for timing simulations | Pin | Pin based frontend simulator for functional simulations and the event-driven backend simulator for timing simulations |
| Modeling Features | Core models, memory subsystems (cache hierarchies with full cache coherence), on-chip networks | Cache models, abstract core models, CMP models, no network models | Core models, pipelined model, memory subsystems , networks models, system execution modes | Asymmetric core models, memory subsystems and network models | Detailed DBT-accelerated core models, memory subsystems (cache hierarchies) | One-CPI core models, profiling-based core models, memory subsystem, other uncore models |
| Event Sampling | Statistical sampling | Not Specified | Not Specified | Not Specified | Not Specified | Not Specified |
| Synchronization | Lax, lax with barrier and lax with P2P synchronization | Cycle-level synchronization | Not Specified | Pthread scheduling | Bound-Weave parallelization | Thread-level and process-level barrier synchronization |
| Supported ISAs | x86 | SPARC-v8 | ARM, ALPHA, MIPS, Power, SPARC, and x86 | x86 | x86 | x86 |
| Workloads | Unmodified Pthread applications | Not Specified | Multi-threaded applications in System-call emulation mode and variety of workloads in Full-system mode. | Unmodified Pthread applications (single-threaded and multi-threaded workloads) | Most modern and complex workloads (such as multithreaded applications, JVM and client-server workload) | MPI applications |

The results of these comparative analysis, presented in the tables, also demonstrate the qualitative ideas expressed on Section 3.1. The discussion of the consequences of these analysis will be deeply used in the Sections 5 and 6 in order to discuss the status of manycore future simulators.

## 5. Fundamental Simulation Challenges

Two important features of future peta-scale systems are high-heterogeneity of cores and, very large number of cores. Due to these features, the requirements for simulation of future peta-scale manycore systems include supporting:

- Scalability, in terms of number of cores simulated;
- Heterogeneity, in terms of diversity of cores;
- High speed simulation, due to the complexity and large number of components of such systems;
- Accuracy, in terms of level of detail and similarity to the target systems;
- Flexibility to rapidly explore a very large design space, due to the high diversity of potential architectures for future peta-scale systems;
- And low cost simulation, in terms of hardware required as simulation host.

Efficiently designing future peta-scale manycore architectures is not achievable unless we use powerful simulation tools, fulfilling the abovementioned requirements, and able to drive micro-architecture exploration, evaluate novel systems and design decisions. It is obviously desirable to have a simulator with the highest level of accuracy and scalability, while providing a very fast simulation speed. Along this line, the typical full-cycle-accurate simulators cannot be used anymore, since in some cases, simulating a single second of execution can take between 1 to 12 days. We expect simulators to deal with a lot of design space variables, such as heterogeneity, scalability, modularity, in order or out-of-order core models, cache coherency, memory hierarchy, distributed or shared memory, accelerators, etc. However, there are critical challenges that create difficulties and obstacles to produce such a desirably simulator.

Improving large manycore architectures with large amount of cache memory led simulators to employ highly parallel methods to distribute simulation workloads (as it is the case in current parallel simulators such as PriME and Graphite). However, due to data dependency between parts of the simulated environment, full parallelization is not easy to achieve and we return to the problem of concurrency in current architecture models. Additionally, scale-out from core/tile to die, socket, rack and data-center is really challenging, although this is a necessary requirement for evaluation of future peta-scale machines.

In the remaining of this section, we discuss the major challenges for simulation of future peta-scale systems. We must also note that, in this paper, due to the space limitations, and to avoid losing focus, we avoid the discussion on the issues related to implementation details of the architectural simulators (such as scheduling, synchronization, etc.), since most of these issues are not particularly specific to the area of microprocessor and manycore simulation, but instead generally apply to many different areas in computing.

### *5.1. System Complexity vs Simulation Capacity*

Complexity arises in manycore systems by having more and more diversity and intricacy in the computing architecture and interconnects. For future peta-scale systems, we expect simulation tools to provide functionalities to evaluate the complexity of such nonlinear systems by modeling properties and behavior of each individual component, showing ways where the system behavior cannot be assumed as the sum of the behavior of its parts. For instance, a minor manipulation on the processing/communication properties/behaviors of one system element may cause a significant impact, a proportional impact, or even no impact on other elements or the whole system. However, managing such complexity is a critical challenge in manycore system simulators which are constrained by limited capacity to deal with the system complexity like coupling timing models with functional models. For example, timing models can be multiple **orders of magnitude** slower than real time.

### *5.2. Performance*

Execution performance for manycore simulators is pushed to its limits when a parallel simulator divides its main sequential instruction stream in a set of segmented simulation workloads which have to be executed on a group of individual processors. Basically, it is not just the problem of simulators, but rather it is a challenge in the upcoming manycore architectures, which obligate a paradigm shift in algorithmic design to effectively fully unlock manycore capabilities to achieve maximum performance. Thus, simulators have to deal with challenges such as increasing the level of parallelism, multi-scaling, hardware acceleration, etc., both during development and implementation phases.

### *5.3. Speed vs Accuracy*

Interconnection networks and hardware details of manycore machines can be modeled at very different levels of abstraction depending on the target usage of the simulator. So there is a trade-off between simulation detail and simulation speed. The level of detail of the models used for describing the resources employed within the simulator has a direct impact on the final behavior and characteristics of the simulator. Hardware resource Description Languages (HDLs) are essential tools to build an architecture simulator, and to build an efficient simulator we must use a powerful, accurate and appropriate resource description model. While classical HDLs, like Verilog and VHDL [152], are very good at describing detailed hardware characteristics and behaviors (such as timing behavior), they are generally inadequate for expressing the higher-level abstractions required for (today and) future large and complex micro-architectural architecture designs.

The Architecture Description Language (ADL) [153] and the XML-Based ADL [154] are examples of the higher-level

resource description model, which have been used to build many-core simulators such as Mhetero [155] and M3C [156]. C$\lambda$aSH [157–159] is another example of functional resource description which is based on Haskell. Using an efficient resource description language might help to increase the total performance of the constructed simulators.

Ideally, in order to achieve a proper peta-scale-level simulation platform it is required to describe the hardware details and interconnections as detailed as possible while keeping the total throughput of the system within an acceptable level. This requires the (simulator) designers to build methodologies for constructing calibrated models. Unfortunately, this is not easy to perform, and a common strategy in current simulators is to balance the need for simulation accuracy versus the desire for good simulation performance. Better modelling of target system details by a simulator will produce more accurate results but will result in a slower simulation. This trade-off is particularly important when modeling the interconnection network of a parallel computer.

### *5.4. Development Cost*

The cost of developing a new validated and useful simulator refers to two different aspects. On one hand, the developing time of a simulator should not exceed a reasonable duration. For example if we go to design a more detailed cycle-accurate modeling system, this results in consumption of more time on development, which is not desired. On the other hand the designed simulator should be modular, pluggable and able to reuse in future related works. In addition, new methodologies for constructing simulators have to show an acceptable performance while executing on the underlying hardware, and should not be dependent of some specific-purpose hardware infrastructure.

However, sticking to the objective of cost efficient modeling solutions in case of either the cost of the required hardware (e.g. current software-based simulators like GEM5), or the cost of time consuming for development (e.g. SimK), creates substantial constraints on implementation of some efficient solutions in aspects other than cost.

### *5.5. Design Space*

Efficient design space exploration and modeling of manycore environments could be desired from different research communities with focusing on different metrics, design aspects, applications and requirements. This is a major research challenge for manycore modeling and simulation. On one hand, a diversity of metrics comes from differentiated market segments and metric emphases, such as power, temperature, latency, or throughput. On the other hand, a diversity of variable multi-processor designs (even designs which are not existing yet) are of interest by users. A comprehensive design exploration model has to locate the optimal amounts of results for different metrics and workloads in a large and high-resolution design space. It should consider all design space parameters simultaneously, while enabling predictions for metrics of interest.

Full design space exploration (e.g. the case in current full system simulators such as SimFlex and GEM5) is constrained by the substantial costs of cycle-accurate simulators, which provide very detailed view into system modeling for a wide range of manycore micro-architectural configurations. We can expose specific trends or interested metrics in design space. However, due to issues as long simulation times, we are limited to constrain design scenarios and consider only small subsets of the full design space (as it is the case in current instrumentation-based simulators such as ZSim and McSimA+). Unfortunately, implying these limitations in design space may lead to results that may not be acceptable for the larger space (e.g. future peta-scale systems with high diversity of processors).

### *5.6. Simulation Time*

Designing a high-performance manycore system is extremely time-consuming. It involves exploring and analyzing a huge number of input parameters and configuration elements. Thus, it would be expectable that, in most cases, general simulation methods become infeasible or inefficient. The problem is that architectural simulation is very time consuming. In order to design a microprocessor with optimal characteristics, the simulator must explore and evaluate all possible configurations and find the optimal one. Note that finding the optimum set of configuration values can be different depending on the target and the design criteria. In fact we can say that the total simulation time is directly proportional to the number of configuration parameters which are required to be evaluated. The simulation time is also proportional to the size of the workload space, the average number of instructions in each application-input pair in the workload space, and the simulator slowdown factor [160].

During simulation, the performance evaluation of the system configuration is done through running a software program or a benchmark with a suitable set of inputs. In other words a collection of computer programs (workload) is used for the evaluation process. The simulator must provide capabilities to compose required workloads (the benchmarks with appropriate inputs) which are quite specific depending on the target operation domain of simulated system. Exploration of workload space along with design space has direct impact on the simulation time. Furthermore, the size of workload and extra features of the manycore processor architecture increase the timing cost of the simulation.

It is obvious, that to reduce the total simulation time, we need to reduce the size of any or all of the aforementioned factors (e.g. COTSon has reduced its workload space to only support MPI applications, ZSim has also reduced its design space by only supporting x86 as target ISA). But this might have greater costs in other aspects of simulator performance. Along this line, techniques such as selecting a region of interest through statistical simulation [161, 162] (as used in Graphite, LiveSim [163] and BigHouse [164]), choosing a limited but representative set of program-input pairs [165–167], reduced input sets [168], trace sampling [169] (as used in TQSIM [170]), barrier interval time parallelism [171] and simulation optimization [172, 173] (as these are partially used in BigSim and COTSon) might be taken into account.

### 5.7. Multi-Model Simulations

We must consider modeling manycore systems in different scenarios, as for some studies and analyses we need to have a simulator with capability of multi-dimensional modeling. It means that the simulator must provide an accurate modeling of the system and processor architecture while it offers functionalities to analyze the impact of interconnect networks, thermal and power consumption modeling. It is a challenging task, since we need to put several modeling mechanisms that communicate and work together.

### 5.8. Scalability

In order to model future peta-scale manycore systems and architectures, the current parallel software simulators are supposed to exploit the highest level of parallelism from the underlying parallel architectural host platforms. The obvious big challenge of such simulation is dealing with the rapid increasing number of cores and threads. We expect modeling and analyzing tools to keep a constant amount of overhead, while coping with the exponentially increasing number of the cores. In other words, overhead must be constant in time and independent from the system size. Handling and managing the intercommunication between threads/cores, resources and threads, which are involved in synchronization and contention points between threads and cores, in both application and architecture level is critical with respect to the synchronization bottlenecks. The problem of synchronization in manycore era will result in large scalability issues.

### 5.9. Productivity

Researchers and system designers expect simulators to be more productive and come with a set of requirements, features and characteristics such as ease of use, management tools, documentation and deploy-ability, visualization, deployment, debugging, etc. which could help them to perform modeling, designing and evaluating of the future systems in a better way. However, these requirements are changing, and sometimes are unclear. In addition it increases the cost of simulator construction.

## 6. Discussion and Future Directions

In order to produce peta-scale manycore simulators with capability to explore a wide range of micro-architectural design space of future parallel systems and architecture we have to use the capabilities of the current parallel hardware. This means that we must be able to fully exploit the current hardware architecture to build future parallel hardware and platforms. This direction, the only that will follow current hardware evolution, brings all of the challenges in application concurrency, manycore and hardware parallelism, to simulator designs, while they face some specific challenges which particularly belongs to the scope of distributed and parallel simulations. We also need to identify the most important capabilities of the current simulators.

The followings summarize the key information on the capabilities of the current architecture simulators (as already discussed in Sections 3 and 4):

- Instrumentation (DBT) based simulators provide faster and more scalable simulation capability compared to emulation based simulators, while the emulation based simulators provide better simulation accuracy to general architectures.
- The configurability (i.e. the ability to explore various ISAs and system architecture) of instrumentation based simulators is limited. Moreover, most of the instrumentation based simulators are user/application level simulators.
- Sampling based simulators are faster and more scalable compared to non-sampling based simulators with the cost of reducing simulation accuracy.
- Hardware-assisted simulators are faster than software-assisted simulators. However, they create complexity for development and they are not flexible for design space exploration. Furthermore, hardware-assisted simulators are more expensive than software-assisted simulators.
- Network/interconnect modeling is a requirement for distributed-memory simulation. Furthermore, distributed memory simulators are more scalable than shared memory simulators and they can provide faster simulation by enabling cluster based simulation.
- Parallel simulators are faster than sequential simulators. However, they generally suffer from drawbacks, such as complicated and inefficient scheduling of simulation segments and high communication overhead. Synchronization techniques improve the performance of the parallel simulators. But still efficient synchronization is a challenging issue for parallel simulation.
- Modularity is a very important feature for simulators, since it provides fast development, ease of use and flexibility for simulators. Accordingly, for most current simulators, efforts have been made to apply modularity through creating component based architectures which enables leveraging of existing tools and components.

We can conclude that, due to the large variety of simulator applications and methodologies trade-offs, it is not feasible to expect a comprehensive simulator to simultaneously satisfy the requirements of researchers in different communities. Therefore simulator designers must clarify the specific target usage of their simulation tools at the first step. Furthermore, due to the aforementioned challenges, we should make efforts towards modular simulation platforms that are able to leverage existing high throughput modules, methods and solutions to build the new tools. Here, base-simulation techniques such as synchronization, sampling, scheduling, etc. are required either to be improved or recreated.

In this paper, we have presented the characteristics and capabilities of the current simulations and modeling tools, aiming to simulate future peta-scale manycore systems and architectures. We have extracted and demonstrated a set of most significant problems and challenging issues which the simulator designers have to deal with. In order to cope with the aforementioned

challenges, and to respond to the needs and concerns of different design space exploration and multi/many core architecture simulation requirements, we argue that, several techniques appear to be more promising to develop. Thus, we expect that the future direction of research in modeling and simulation of manycore systems will expand upon the following lines, relying on existing best of breed features currently incipient in different simulators:

- Regression and Analytic Models
- Statistical Simulation
- Acceleration and FPGA-Based Prototyping
- Modularity, Integrability and Aspect-oriented Simulation
- Parallel Simulation
- Cloud-based Simulation
- Raising Level of Abstraction
- Model-driven Simulation

### *6.1. Regression and Analytic Models*

Regression and analytical modeling is a statistical tool for the investigation of relationships between design space metrics and variables and it is an efficient approach for accurately predicting different metrics and parameters in a large micro-architectural design space. Regularly, in manycore modeling we seek to clarify the causal effect of one metrics upon another metrics and parameters. To explore such issues, we can apply regression techniques to simulators, enabling them to efficiently obtain approximates of the design metrics. Simulators can collect data on the interested underlying metrics and employ regression analytic approaches to foresee the quantitative effect of the causal metrics upon the metric that they influence. For example, the work presented in [184] uses this approach for design space exploration by applying a class of models where, in each model, the response time is modeled as a weighted sum of foreseeing metrics plus a random noise (a noise is defined as the effect of the other metrics which are not considered in the prediction model).

### *6.2. Statistical Simulation*

Analyzing all the functional events and creating timing models for each component is costly due to the simulation time. Sampling and statistical simulation techniques are solutions to decrease the time of simulation. These approaches rely on collecting and pre-calculating certain architecture-dependent performance factors. The analysis techniques periodically sample and statistic on some characteristics of the running application in order to accelerate subsequent simulations. This solution proved to be highly accurate while having low overhead. However, the statistical simulation is constrained in the number of architectures that it can support. Furthermore, it is not clear whether these techniques can tolerate significant modifications in the number of cores or the interconnect network topology.

### *6.3. Acceleration and FPGA-Based Prototyping*

Architecture research already has started to concentrate more on implementation and less on design of instruction sets. Along this line, FPGAs have been employed at different levels of abstraction in the design for the purpose of simulation, implementation and evaluation of current and future computer architectures. Architecture simulators can achieve benefits from increasing the speed of register transfer level simulation by performing logical simulation and evaluation based on FPGA-accelerated hardware platforms. As example of these range of simulators, we can mention Quick-turn [179] (an old product) and CadencePalladium [180] (a more recent one). [181] is another example which integrated FPGA technologies into a tool for exploring and evaluating microarchitectural designs especially for newly proposed architectures.

The simple idea behind FPGA-based simulators is using FPGAs to simulate Register Transfer Level (RTL) (i.e. a design abstraction which models a synchronous digital circuit), because a RTL simulation typically is very slow. In addition, we can map processors directly to FPGAs, which results to enable using higher clock rates. According to this concept the Research Accelerator for Multiple Processors (RAMP) [182] provides a set of research efforts such as RAMP Blue [183], RAMP Red and RAMP Gold [147]. RAMP Blue is a simulated machine, with 1008 cores, which are inter-operated by message passing and distributed memory. RAMP Red is another many-core simulator designed to investigate the issues of transactional memory (TM). Both of these projects used Xilinx FPGAs on Berkeley Emulation Engine 2 (BEE2) boards.

The most important problem of direct-RTL-mapping is that timing models is a property of the functional modeling. RAMP Blue tried to solve this problem by employing a separate simple timing model, which uses clock-gating of various components to get correct functional behavior while faking the desired timing model. This technique implies a tight coupling between the timing and functional simulations because the timing model is a wrapper around each functional component. Direct RTL-mapped systems do not support modularity and they are also suffering from timing wrappers. This leads FPGA-based simulators to completely split the timing and functional modeling of the target system.

### *6.4. Modularity, Integrability and Aspect-oriented Simulation*

There are islands of simulation and modeling tools with various focus on different domains, which are completely isolated from each other. Modularity is an important requirement for simulation tools in order to achieve integrability and reusability. Modularity can be obtained in different levels. Implementation-level (object-level or component-level) modularity is a common approach to design modular simulators, where a simulator composed of a set of interrelated components/objects working together toward a common objective. This even makes simulators more flexible to interoperate with other external simulators or simulator components. In fact, a powerful and comprehensive solution has to benefit from existing tools and individual investments as much as possible. In addition, a proper simulation approach must provide an infrastructure to leverage mature tools via standardized APIs for common simulator services and functionalities (such as time modeling, synchronization, sampling, event modeling, interconnects simulation, bandwidth and latency modeling, thermal and energy consumption modeling). It could support a very large design space exploration in various aspects and objectives. Furthermore, it catalyzes the development process of the new simulator platforms while the

resulted tools would be substantially efficient. There are several examples of successful (implementation-level) modular integrations in the current literature, including Qemu-COTSon [30], NOXIM-COTSon (NOXIM is a network on chip simulator), and SystemC-Qemu [131, 133–136].

Modularity can also be applied in a more conceptual level by advocating aspect-oriented paradigms into the area of manycore simulation. This type of modularity can be described as model-level (or aspect-level) modularity, where simulation of a target (manycore) system can be performed considering different viewpoints (aspects or simulation aspects). Each viewpoint can be defined as a representation (specification) of a target system (simulated system) in a certain view or aspect (i.e. simulation aspect). In fact, a viewpoint (or an aspect) specifies the simulation focus (e.g. memory and cache, processing cores, network on chip, power and energy, OS and application execution). For example, in NoC simulators, it might be necessary to provide more detailed interconnection while describing cores in more abstract level, since the focus of such simulators is on interconnection networks.

In this context, modularity refers to separating the simulation of different concerns of a target system and decomposing the entire simulation into a set of simulation aspects. Each simulation aspect can simulate the system under study from a particular point of view. The simulation aspects can be recomposed through a weaving mechanism (e.g. [185]), providing facility to simulate a target system from all aspects or only certain aspects of interest. These weaved modular aspects can flexibly communicate to each other in order to provide a detailed model of a target system with respect to desired aspects. This can provide simulation efficiency specially when simulation is required for systematic optimization. In such a case, the optimizer aims to optimize the system under study from a certain point of view, therefore it needs to evaluate the system from a particular view by simulating the certain aspects of the system. There are some works that already introduced aspect-oriented simulators [186–189] in the field of discrete event simulation. However, none of them are specifically designed for the purpose of manycore architecture simulation.

### 6.5. Parallel Simulation

The traditional sequential simulators already began to be converted to parallel simulators. However, there are still major challenges that adversely affect the performance of parallel simulations. These issues include segmentation of simulation workloads, dynamic scheduling, communications between simulator instances, time management and synchronization. Among them, synchronization is a key issue, since lacking an adequate synchronization may result in a time causality error (i.e. violation of the time-stamp order), imposing overhead by periodical synchronization in order to achieve consistency among logical clocks of different host nodes [174]. For example, a high speed node might receive a straggler event with an overdated time-stamp from a low speed node (an straggler event is an event which its time-stamp is less than the local clock value). A potential direction to cope with the synchronization problem might be to use multilevel lax(relax)-based synchronization approaches, as it is getting common for recent parallel (manycore) simulators [174, 175] due to their capability to significantly reducing the synchronization frequency and in turn, reducing synchronization overhead and increasing parallelism. These strategies work through efficiently adapting a lax-based technique (e.g. by lengthening of synchronous periods in occurrence of straggler events [11]) in one level, while correcting the simulation accuracy in another level (note that traditional lax-based synchronization suffer from the lack of simulation accuracy).

Overall, regardless of the aforementioned parallelism issues, we propose two future directions for simulation-level parallelism along the thematics of this paper: component-level (object-level) and aspect-level (model-level) parallelism. In component-level parallelism, a simulator contains a set of stand-alone components (a component may include sub-components/objects) and a target system can be simulated by instantiation of components required and definition of inter-component processes (e.g. communication between component instances). Component-level parallelism [176, 177] can be achieved by parallel execution of component instances on different host nodes (processors). In aspect-level parallelism, the simulation of a target system includes a set of stand-alone aspects where each aspect provides a detailed representation of a system from the prospective of a certain aspect. More complete (global) simulation of a target system can be extracted through merging a set of aspects desired. We further discuss this in the remaining of this section (refer to aspect-oriented simulators).

### 6.6. Cloud-based Simulation

The Cloud service models such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) facilitate doing research in all the related fields of distributed/parallel computing. Using Cloud-based systems offer the end users a completely remote, self-organized, load balanced and distributed environment as a single service/single system which is more efficient and easy to work with. Definitely, the Cloud-based software services are able to support the whole rage of softwares as services even simulator softwares. Currently there exist some research works that provide system simulation service for Cloud as part of SaaS [198]. However, all the software (such as computation centric and real time applications) and particularly simulators are not the same and they have different operational behaviors and characteristics, so we might consider that using a generic SaaS would not be enough to satisfy all the requirements of the future Peta-scale system simulators. Deploying simulators using SaaS [199] will arise some complexities in different aspects such as configuration, synchronization, workload distribution, etc. In other words it doesn't provide the performance and ease of use as the simulator users are expecting.

A possible approach to improve the performance of the Cloud-based simulators [200–202] is developing the Cloud service model as Simulation-as-a-Service(SiaS), which means that SaaS have to be adapted and customized precisely to run simulators. So by using such a simulation service, the simulator runs on the Cloud servers while leaving the user's local resources free. A large scale manycore simulator explores a very large

model/experiment space where the behavior of each model is explored during its creation. Furthermore, in runtime each experiment model in the design space will find emergent properties which can not be deducted from the model and they can only be observed during simulation. Simulation in large scale involves a high number of parameters, values and settings and we can say that it is a resource intensive application which most of its behaviors can be only identified during runtime. Such a Cloud-based simulation service provides a service based on the amount of simulated time for the end user experiment. It takes care of the distribution of the experiment work load on several machines in the Cloud, and finally it will collect all the results from the worker machines.

### 6.7. Raising Level of Abstraction

Generally, architecture simulators suffer from two major timing limitations, which are simulation time and development time of simulators. Full-cycle-accurate simulators are examples of this case. They are accurate and fulfill the user requirements for correctness of the experiments, but they are highly time consuming and result from a complex development process which takes time. But this detailed modeling is not necessary for many kinds of design space exploration and just makes the development of new models more difficult. For example, we do not usually need to model the details of cache coherence protocols or many-core interconnection networks when investigating trade-offs in the memory system hierarchy.

Raising the abstraction level of the simulation will help simulators to be faster and easier to use, while remaining accurate. The problem is finding out what level of abstraction is appropriate and how to deal with the tight performance among the co-executing threads and the micro-architectures. Interval simulation [178] is a possible solution along this line. It raises the level of abstraction in the core-level compared with the typical detailed simulation. It uses a mechanistic analytical model [12, 178] which drives the timing models of each individual cores with sampling of some set of the instructions through the cores' pipeline stages. Each interval is defined as the distance between two miss events (branch mis-predictions, cache L1 I-cache miss, long-latency load miss or TLB misses) (see Figure 9) through the cores' pipeline stage which identifies a part of instructions stream. The miss events can be determined by employing branch predictor, memory hierarchy, cache coherence and interconnection network simulators.Thus, the mechanistic analytical model drives the timing models for each intervals instead of the whole instruction set. The integration between miss event simulators and the analytical models facilitate the modeling of the tight performance entanglement between co-executing threads on manycore architectural processors. This approach is a promising technique particularly for system level exploration at the early design stage. Sampling the simulated instruction stream, using host multi-threading and mapping the simulations workloads on FPGAs or GPU are the other alternative approaches that achieve considerable simulation speedups while maintain the performance in a cycle-accurate manner.

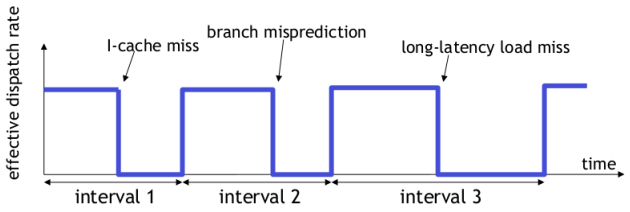


Figure 9: Interval simulation according to miss events (extracted from [178]).

### 6.8. Model-driven Simulation

MDE based simulation (simulation based on Model Driven Engineering) can be used as a powerful alternative for manycore simulations in future instead of instrumentation-based simulation. Model-Driven Engineering (MDE) is a software development methodology that evolved as a paradigm shift from the object-oriented paradigm (everything is an object), into the model engineering paradigm (everything is a model) [190]. Currently, this method is used mostly in the domain of software engineering, programming languages and domain specific languages, but not in manycore simulation. Applying MDE concepts in the area of simulation, may lead to a new generation of manycore simulators which can efficiently cope with multiple challenging issues, including interoperability (e.g. ability to simulate various target systems on a fixed host system), multi-model simulation, simulation speed, design space exploration, etc.

In MDE, everything is defined as model and a model basically is a description of a real system (like a manycore system including different components such as memory, processing cores, network and interconnect). In fact, a (real) system is an instantiation of a model. Each model, in turn, is defined by another model, so-called metamodel (i.e. the abstract syntax of a modeling language is specified using another model, describing the syntactic elements and the relationships existing between those elements). The Object Management Group (OMG) proposed a four-level metamodeling framework as a standard to develop modeling languages. In each level, except the bottom level (M0), there is a model that specifies a set of other models at the lower level in a recursive way. The M0 is the bottom level of this hierarchy, specifying various real systems (e.g. a memory component, a processor, different manycore systems). At the higher level (M1), models (e.g. UML class diagrams) represent (abstract) these systems. Each model conforms to its metamodel defined at the upper level (M2). And similarly, metamodels in level M2 conform to another models, defined at the highest level (M3), so-called meta-metamodels. Meta-metamodels (e.g. OMG's Meta Object Facility (MOF) [191]) are self-descriptive entities, enabling to confirm to theirself. MDE is able to raise the level of abstraction in system description/specification (by using models at the different levels of abstraction) and increase automation in running (execution or simulation of) a system (by using code generation and model transformations mechanism). In other words, the MDE approach promotes the use of models as first-class entities that need to be constructed, maintained, executed, and mapped into other models or artifacts by model transformations.

Overall, MDE covers aspects such as architecture design, code generation, model transformation and model checking. Model checking includes techniques to check and ensure the quality (performance) of the models (e.g. model validation using behavioral properties). Referring to the above-mentioned metamodel hierarchy, a model-driven manycore simulator can flexibly explore the design space through specification of unlimited target systems as different models in different levels of metamodel hierarchy. Since the descriptions of target models in metamodel hierarchy is very abstract, the simulator can swiftly check the functional accuracy of different designed target models, before going to the detailed simulation of a specific target system. This can be done through using model checking/validation mechanism provided by MDE approach. Furthermore, using model transformation languages [192–196], a model-driven simulator can provide interoperability by means of capability to convert any desired target models to codes which can be directly executed on the host machine. This means that a MDE-based simulator can flexibly address platform complexities through simulation of any desired target system on a fixed host machine (similar to retargetable DBT, discussed in Section 3.1).

As we discussed above, current Model-Driven Architectures (MDAs) provide capability for structural metamodeling which can efficiently be used for functional modeling in manycore simulators. A recent work [197] in this area introduces the behavioral metamodeling to complement the structural metamodeling (see Figure 10). This behavioral metamodeling (including various behaviors described in different levels of metamodel hierarchy) facilitates to extract behavioral properties of a target model (target system) during run-time. This allows performance modeling while functional modeling is performed (see Figure 11). Model-driven simulation can be used as an alternative for instrumentation-based simulation which extracts the performance behavior of the target system by injecting code into the simulated binary, running on the host machine.

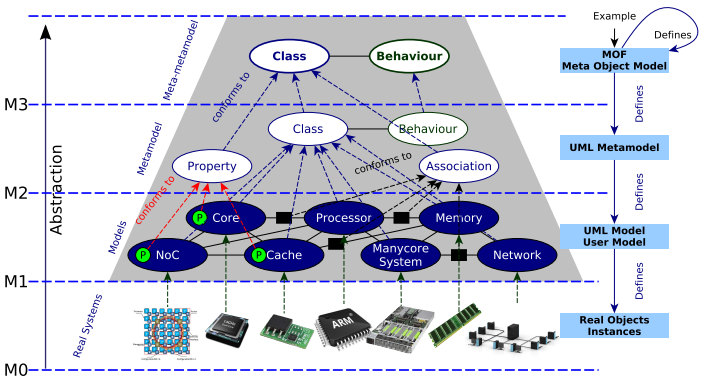


Figure 10: An example of both structural and behavioral metamodel hierarchy for model-driven manycore simulation (note that "Behaviour" entities are specific for behavioral metamodeling).

### 6.9. Summary

Overall, the areas open to innovation are manyfold. We do expect next generation manycore simulators to rely on several of the aforementioned techniques to reach the required performance and complexity management of future simulations. Global technology development, and specially the specific development we will see in manycore systems, and its associated market, will determine the relevance of each of these simulation techniques in the future manycore simulations that will handle peta-scale computing systems.

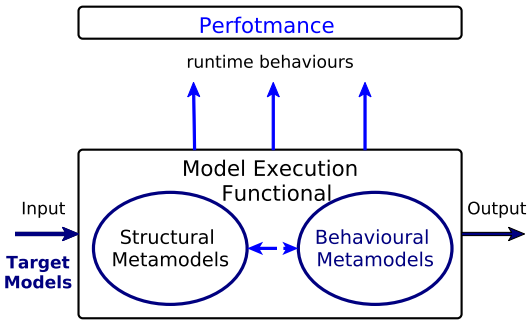


Figure 11: Extracting behavioral properties of a target model (target system) during run-time.

## 7. Acknowledgment

The authors acknowledge the support of project FP7-ICT-2009.8.1, Grant Agreement No.248465, Service-oriented Operating Systems (S[o]OS, 2010-2013) [203–211] and of project Cloud Thinking (2013-2015), CENTRO-07-ST24-FEDER-002031 [212]. We advocate for the research community to develop a collective effort for a community integrated S[o]OS environment, along the lines here discussed.

## References

[1] Toshiyuki Imamura, Susumu Yamada, and Masahiko Machida. Development of a high performance eigensolver on the petascale next generation supercomputer system. In *Proceedings of Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2010 (SNA+ MC2010)*, 2010.

[2] Can-qun Yang, Qiang Wu, Tao Tang, Feng Wang, and Jing-ling Xue. Programming for scientific computing on peta-scale heterogeneous parallel systems. *Journal of Central South University*, 20(5):1189–1203, 2013.

[3] M. Pellauer, M. Vijayaraghavan, M. Adler, Arvind, and J. Emer. Quick performance models quickly: Closely-coupled partitioned simulation on fpgas. In *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on*, pages 1–10, April 2008.

[4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, and Somayeh Sardashti. The gem5 simulator. 39:1–7, 2011.

[5] Nikolaos Hardavellas, Stephen Somogyi, Thomas F Wenisch, Roland E Wunderlich, Shelley Chen, Jangwoo Kim, Babak Falsafi, James C Hoe, and Andreas G Nowatzyk. Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. *ACM SIGMETRICS Performance Evaluation Review*, 31(4):31–34, 2004.

[6] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. Marss: a full system simulator for multicore x86 cpus. In *Proceedings of the 48th Design Automation Conference*, pages 1050–1055. ACM, 2011.

[7] Jian Lu, Hongwei Jia, Andres Arias, Xun Gong, and Z John Shen. On-chip bondwire transformers for power soc applications. In *Applied Power Electronics Conference and Exposition, 2008. APEC 2008. Twenty-Third Annual IEEE*, pages 199–204. IEEE, 2008.

[8] Daniele Bortolotti, Christian Pinto, Andrea Marongiu, Martino Ruggiero, and Luca Benini. Virtualsoc: A full-system simulation environment for massively parallel heterogeneous system-on-chip. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 2182–2187. IEEE, 2013.

[9] T. Austin, E. Larson, and D. Ernst. Simplescalar: an infrastructure for computer system modeling. *Computer*, 35(2):59–67, feb 2002.

[10] Gengbin Zheng, Gunavardhan Kakulapati, and L.V. Kale. Bigsim: a parallel simulator for performance prediction of extremely large parallel machines. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 78–, April 2004.

[11] J.E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. Graphite: A distributed parallel simulator for multicores. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12, jan. 2010.

[12] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 52. ACM, 2011.

[13] Daniel Sanchez and Christos Kozyrakis. Zsim: fast and accurate microarchitectural simulation of thousand-core systems. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 475–486. ACM, 2013.

[14] Yaosheng Fu and D. Wentzlaff. Prime: A parallel and distributed simulator for thousand-core chips. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 116–125, March 2014.

[15] Carl J. Mauer, Mark D. Hill, and David A. Wood. Full-system timing-first simulation. *SIGMETRICS Perform. Eval. Rev.*, 30(1):108–116, June 2002.

[16] D.A. Penry. A single-specification principle for functional-to-timing simulator interface design. In *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, pages 186–196, April 2011.

[17] Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil Patil, William H. Reinhart, D. Eric Johnson, and Zheng Xu. The fast methodology for high-speed soc/computer simulation. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '07, pages 295–302, Piscataway, NJ, USA, 2007. IEEE Press.

[18] Eric Schnarr and James R. Larus. Fast out-of-order processor simulation using memoization. *SIGPLAN Not.*, 33(11):283–294, October 1998.

[19] G.H. Loh, S. Subramaniam, and Yuejian Xie. Zesto: A cycle-level simulator for highly detailed microarchitecture exploration. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 53–64, April 2009.

[20] S. Fytraki and D. Pnevmatikatos. Resim, a trace-driven, reconfigurable ilp processor simulator. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 536–541, April 2009.

[21] Zhenman Fang, Qinghao Min, Keyong Zhou, Yi Lu, Yibin Hu, Weihua Zhang, Haibo Chen, Jian Li, and Binyu Zang. Transformer: A functional-driven cycle-accurate multicore simulator. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 106–114, New York, NY, USA, 2012. ACM.

[22] J. Donald and M. Martonosi. An efficient, practical parallelization methodology for multicore architecture simulation. *IEEE Computer Architecture Letters*, 5(2):14–14, July 2006.

[23] Eduardo Argollo, Ayose Falcón, Paolo Faraboschi, Matteo Monchiero, and Daniel Ortega. Cotson: infrastructure for full system simulation. *SIGOPS Oper. Syst. Rev.*, 43(1):52–61, January 2009.

[24] Roberto Giorgi. Teraflux: Exploiting dataflow parallelism in teradevices. In *Proceedings of the 9th Conference on Computing Frontiers*, CF '12, pages 303–304, New York, NY, USA, 2012. ACM.

[25] Naveen Neelakantam, Colin Blundell, Joe Devietti, Milo MK Martin, and Craig Zilles. Fes2: A full-system execution-driven simulator for x86. *Poster presented at ASPLOS*, 2008:6, 2008.

[26] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, November 2005.

[27] Robert C. Bedichek. Talisman: Fast and accurate multicomputer simulation. *SIGMETRICS Perform. Eval. Rev.*, 23(1):14–24, May 1995.

[28] Pablo Montesinos Ortego and Paul Sack. Sesc: Superescalar simulator. In *17 th Euro micro conference on real time systems (ECRTS'05)*, pages 1–4, 2004.

[29] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K. Reinhardt. The m5 simulator: Modeling networked systems. *Micro, IEEE*, 26(4):52–60, July 2006.

[30] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[31] Emmett Witchel and Mendel Rosenblum. Embra: Fast and flexible machine simulation. *SIGMETRICS Perform. Eval. Rev.*, 24(1):68–79, May 1996.

[32] Patrick Bohrer, James Peterson, Mootaz Elnozahy, Ram Rajamony, Ahmed Gheith, Ron Rockhold, Charles Lefurgy, Hazim Shafi, Tarun Nakra, Rick Simpson, Evan Speight, Kartik Sudeep, Eric Van Hensbergen, and Lixin Zhang. Mambo: A full system simulator for the powerpc architecture. *SIGMETRICS Perform. Eval. Rev.*, 31(4):8–12, March 2004.

[33] Robert Bedichek. Simnow: Fast platform simulation purely in software. In *Hot Chips*, volume 16, 2004.

[34] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, Feb 2002.

[35] Mendel Rosenblum, Edouard Bugnion, Scott Devine, and Stephen A. Herrod. Using the simos machine simulator to study complex computer systems. *ACM Trans. Model. Comput. Simul.*, 7(1):78–103, January 1997.

[36] C.J. Hughes, V.S. Pai, P. Ranganathan, and S.V. Adve. Rsim: simulating shared-memory multiprocessors with ilp processors. *Computer*, 35(2):40–49, feb 2002.

[37] Eric A. Brewer, Chrysanthos N. Dellarocas, Adrian Colbrook, and William E. Weihl. Proteus: A high-performance parallel-architecture simulator. *SIGMETRICS Perform. Eval. Rev.*, 20(1):247–248, June 1992.

[38] Jiun-Hung Ding, Po-Chun Chang, Wei-Chung Hsu, and Yeh-Ching Chung. Pqemu: A parallel system emulator based on qemu. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 276–283, Dec 2011.

[39] Zhaoguo Wang, Ran Liu, Yufei Chen, Xi Wu, Haibo Chen, Weihua Zhang, and Binyu Zang. Coremu: A scalable and portable parallel full-system emulator. *SIGPLAN Not.*, 46(8):213–222, February 2011.

[40] Kun Wang, Yu Zhang, Huayong Wang, and Xiaowei Shen. Parallelization of ibm mambo system simulator in functional modes. *SIGOPS Oper. Syst. Rev.*, 42(1):71–76, January 2008.

[41] R Lantz. Fast functional simulation with parallel embra. In *Proceedings of the 4th Annual Workshop on Modeling, Benchmarking and Simulation*. Citeseer, 2008.

[42] Shobhit Kanaujia, Irma Esmer Papazian, Jeff Chamberlain, and Jeff Baxter. Fastmp: A multi-core simulation methodology. In *Workshop on Modeling, Benchmarking and Simulation (MoBS 2006)*, 2006.

[43] Jianwei Chen, Murali Annavaram, and Michel Dubois. Slacksim: a platform for parallel simulations of cmps on cmps. *SIGARCH Comput. Archit. News*, 37(2):20–29, July 2009.

[44] Shubhendu S Mukherjee, Steven K Reinhardt, Babak Falsafi, Mike Litzkow, Steven Huss-Lederman, Mark D Hill, James R Larus, and David A Wood. Fast and portable parallel architecture simulators: Wisconsin wind tunnel ii. *IEEE Concurrency*, 8(4):12–20, 2000.

[45] Shubhendu S. Mukherjee, Steven K. Reinhardt, Babak Falsafi, Mike Litzkow, Steve Huss-Lederman, Mark D. Hill, James R. Larus, and David A. Wood. Wisconsin wind tunnel ii: A fast and portable parallel architecture simulator. In *Workshop on Performance Analysis and Its Impact on Design*, Denver, Co, June 1997. ACM.

[46] Aamer Jaleel, Robert S Cohn, Chi-Keung Luk, and Bruce Jacob. Cmp $ im: A pin-based on-the-fly multi-core cache simulator. In *Proceedings of the Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA*, pages 28–36, 2008.

[47] T.F. Wenisch, R.E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J.C. Hoe. Simflex: Statistical sampling of computer system simulation. *Micro, IEEE*, 26(4):18–31, July 2006.

[48] Wim Heirman, Trevor Carlson, and Lieven Eeckhout. Sniper: scalable and accurate parallel multi-core simulation. In *8th International*

*Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES-2012)*, pages 91–94. High-Performance and Embedded Architecture and Compilation Network of Excellence (HiPEAC), 2012.

[49] Gabriele D'Angelo and Moreno Marzolla. New trends in parallel and distributed simulation: From many-cores to cloud computing. *Simulation Modelling Practice and Theory*, 49:320 – 335, 2014.

[50] Richard M Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[51] Pengju Ren, M. Lis, Myong Hyon Cho, Keun Sup Shim, C.W. Fletcher, O. Khan, Nanning Zheng, and S. Devadas. Hornet: A cycle-level multicore simulator. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(6):890–903, June 2012.

[52] Jianwei Xu, Mingyu Chen, Gui Zheng, Zheng Cao, Huiwei Lv, and Ninghun Sun. Simk: a parallel simulation engine towards shared-memory multiprocessor. *Journal of Computer Science and Technology*, 24(6):1048–1060, 2009.

[53] Richard M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):30–53, October 1990.

[54] R. Curry, C. Kiddle, R. Simmonds, and B. Unger. Sequential performance of asynchronous conservative pdes algorithms. In *Principles of Advanced and Distributed Simulation, 2005. PADS 2005. Workshop on*, pages 217–226, June 2005.

[55] Richard M. Fujimoto. *Parallel and Distribution Simulation Systems*. John Wiley and Sons, Inc., New York, NY, USA, 1st edition, 1999.

[56] David R. Jefferson. Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3):404–425, July 1985.

[57] K.M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *Software Engineering, IEEE Transactions on*, SE-5(5):440–452, Sept 1979.

[58] Jun Wang, Zhenjiang Dong, Sudhakar Yalamanchili, and George Riley. Optimizing parallel simulation of multicore systems using domain-specific knowledge. In *Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation*, pages 127–136. ACM, 2013.

[59] Jun Wang, Jesse Beu, Rishiraj Bheda, Tayana Conte, Zhenjiang Dong, Chad Kersey, Mitchelle Rasquinha, George Riley, Wanjuan Song, He Xiao, et al. Manifold: A parallel simulation framework for multicore systems. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 106–115. IEEE, 2014.

[60] Jainwei Chen, L. Kumar Dabbiru, D. Wong, M. Annavaram, and Michel Dubois. Adaptive and speculative slack simulations of cmps on cmps. In *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, pages 523–534, Dec 2010.

[61] Yanyong Mongkolsin and Worawan Marurngsith. P-hase: An efficient synchronous pdes tool for creating scalable simulations. In Tianyuan Xiao, Lin Zhang, and Minrui Fei, editors, *AsiaSim 2012*, Communications in Computer and Information Science, pages 231–245. Springer Berlin Heidelberg, 2012.

[62] R. Rajan and P.A. Wilsey. Dynamically switching between lazy and aggressive cancellation in a time warp parallel simulator. In *Simulation Symposium, 1995., Proceedings of the 28th Annual*, pages 22–30, Apr 1995.

[63] Avinash C Palaniswamy, Sandeep Aji, and Philip A Wilsey. An efficient implementation of lazy reevaluation. In *Proceedings of the 25th annual symposium on Simulation*, pages 140–146. IEEE Computer Society Press, 1992.

[64] Ranjit Noronha and Nael B. Abu-Ghazaleh. Early cancellation: An active nic optimization for time-warp. In *Proceedings of the Sixteenth Workshop on Parallel and Distributed Simulation*, PADS '02, pages 43–50, Washington, DC, USA, 2002. IEEE Computer Society.

[65] Robert Pavel. *Simulation methodology and tools for the development of novel program execution models and architectures*. PhD thesis, University of Delaware, 2015.

[66] Duane Ball and Susan Hoyt. The adaptive time-warp concurrency control algorithm. In *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22, pages 174–177, 1990.

[67] A.W. Malik, A. Park, and R.M. Fujimoto. Optimistic synchronization of parallel simulations in cloud computing environments. In *Cloud Computing, 2009. CLOUD '09. IEEE International Conference on*, pages 49–56, Sept 2009.

[68] G. Malhotra, P. Aggarwal, A. Sagar, and S.R. Sarangi. Partejas: A parallel simulator for multicore processors. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 130–131, March 2014.

[69] Jung Ho Ahn, Sheng Li, O. Seongil, and N.P. Jouppi. Mcsima+: A many-core simulator with application-level+ simulation and detailed microarchitecture modeling. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pages 74–85, April 2013.

[70] Steven K. Reinhardt, Mark D. Hill, James R. Larus, Alvin R. Lebeck, James C. Lewis, and David A. Wood. The wisconsin wind tunnel: Virtual prototyping of parallel computers. *SIGMETRICS Perform. Eval. Rev.*, 21(1):48–60, June 1993.

[71] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, 40(6):190–200, June 2005.

[72] Cheng Wang, Shiliang Hu, Ho-seop Kim, Sreekumar R. Nair, Mauricio Breternitz, Zhiwei Ying, and Youfeng Wu. Stardbt: An efficient multi-platform dynamic binary translation system. In *Proceedings of the 12th Asia-Pacific Conference on Advances in Computer Systems Architecture*, ACSAC'07, pages 4–15, Berlin, Heidelberg, 2007. Springer-Verlag.

[73] Derek Bruening, Timothy Garnett, and Saman Amarasinghe. An infrastructure for adaptive dynamic optimization. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '03, pages 265–275, Washington, DC, USA, 2003. IEEE Computer Society.

[74] Leonid Baraz, Tevi Devor, Orna Etzion, Shalom Goldenberg, Alex Skaletsky, Yun Wang, and Yigel Zemach. Ia-32 execution layer: A two-phase dynamic translator designed to support ia-32 applications on itanium®-based systems. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 191–, Washington, DC, USA, 2003. IEEE Computer Society.

[75] Anton Chernoff, Mark Herdeg, Ray Hookway, Chris Reeve, Norman Rubin, Tony Tye, S. Bharadwaj Yadavalli, and John Yates. Fx!32: A profile-directed binary translator. *IEEE Micro*, 18(2):56–64, March 1998.

[76] Rimas Avizienis, Yunsup Lee, and Andrew Waterman. Ramp gold: A high-throughput fpga-based manycore simulator. In *The 4th Workshop on Architectural Research Prototyping, Austin, TX*, 2009.

[77] Thorsten Grotker. *System Design with SystemC*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[78] Shivani Raghav, Andrea Marongiu, Christian Pinto, David Atienza, Martino Ruggiero, and Luca Benini. Full system simulation of many-core heterogeneous socs using gpu and qemu semihosting. In *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units*, GPGPU-5, pages 101–109, New York, NY, USA, 2012. ACM.

[79] S. Raghav, M. Ruggiero, A. Marongiu, C. Pinto, D. Atienza, and L. Benini. Gpu acceleration for simulating massively parallel many-core platforms. *Parallel and Distributed Systems, IEEE Transactions on*, 26(5):1336–1349, May 2015.

[80] E.S. Chung, E. Nurvitadhi, J.C. Hoe, B. Falsafi, and Ken Mai. Protoflex: Fpga-accelerated hybrid functional simulator. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–6, March 2007.

[81] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer. Hasim: Fpga-based high-detail multicore simulation using time-division multiplexing. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pages 406–417, feb. 2011.

[82] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. *SIGPLAN Not.*, 37(10):45–57, October 2002.

[83] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling. *SIGARCH Comput. Archit. News*, 31(2):84–97, May 2003.

[84] Ayose Falcón, Paolo Faraboschi, and Daniel Ortega. Combining simulation and virtualization through dynamic sampling. In *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 72–83. IEEE, 2007.

[85] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe. Statistical

sampling of microarchitecture simulation. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pages 8 pp.–, April 2006.

[86] Joshua J Yi, Lieven Eeckhout, David J Lilja, Brad Calder, Lizy Kurian John, and James E Smith. The future of simulation: A field of dreams. *Computer*, 39(11):22–29, 2006.

[87] Joshua J Yi, Sreekumar V Kodakara, Resit Sendag, David J Lilja, and Douglas M Hawkins. Characterizing and comparing prevailing simulation techniques. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 266–277. IEEE, 2005.

[88] John Scott Robin and Cynthia E. Irvine. Analysis of the intel pentium's ability to support a secure virtual machine monitor. In *Proceedings of the 9th conference on USENIX Security Symposium - Volume 9*, SSYM'00, pages 10–10, Berkeley, CA, USA, 2000. USENIX Association.

[89] Diego Lugones, Emilio Luque, Daniel Franco, Juan C Moure, Dolores Rexachs, Paolo Faraboschi, Daniel Ortega, Galo Gimenez, and Ayose Falcon. Initial studies of networking simulation on cotson, 2009. Accessed: 2013-07-10.

[90] Ayose Falcon, Paolo Faraboschi, and Daniel Ortega. An adaptive synchronization technique for parallel simulation of networked clusters. In *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on*, pages 22–31. IEEE, 2008.

[91] Marinho P. Barcellos, Rodolfo S. Antunes, Hisham H. Muhammad, and Ruthiano S. Munaretti. Beyond network simulators: Fostering novel distributed applications and protocols through extendible design. *Journal of Network and Computer Applications*, 35(1):328 – 339, 2012. Collaborative Computing and Applications.

[92] Dave Nellans, Vamshi Krishna Kadaru, and Erik Brunv. Asim- an asynchronous architectural level simulator abstract, 2004.

[93] V. Puente, J.A. Gregorio, and R. Beivide. Sicosys: an integrated framework for studying interconnection network performance in multiprocessor systems. In *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on*, pages 15–22, 2002.

[94] Pablo Montesinos Ortego and Paul Sack. Sesc: Superescalar simulator. Technical report, 2004.

[95] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. Statistical sampling of microarchitecture simulation. *ACM Trans. Model. Comput. Simul.*, 16(3):197–224, July 2006.

[96] Fabrizio Fazzino, Maurizio Palesi, and Davide Patti. Noxim-noc simulator. *Online:http://noxim. sourceforge. net*, 2010.

[97] Keita Nakajima, Takuji Hieda, Ittetsu Taniguchi, Hiroyuki Tomiyama, and Hiroaki Takada. A fast network-on-chip simulator with qemu and systemc. In *Networking and Computing (ICNC), 2012 Third International Conference on*, pages 298–301. IEEE, 2012.

[98] Antoni Portero, Ramon Pla, and Jordi Carrabina. Systemc implementation of a noc. In *2005 IEEE International Conference on Industrial Technology*, pages 1132–1135. IEEE, 2005.

[99] Song Chai, Chang Wu, Yubai Li, and Zhongming Yang. A noc simulation and verification platform based on systemc. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 3, pages 423–426. IEEE, 2008.

[100] Ye-ting Li and Li-ping Liang. A noc modeling and simulating method with systemc. *Microelectronic & Computer*, pages 78–82, 2010.

[101] Xu Ningyi, Leng Xianglun, Liu Renfei, and Zhou Zucheng. A systemc-based noc simulation framework supporting heterogeneous communicators. In *2005 6th International Conference on ASIC*, volume 2, pages 1032–1035. IEEE, 2005.

[102] Jaison Valmor Bruch, Magnos Roberto Pizzoni, and Cesar Albenes Zeferino. Brownpepper: A systemc-based simulator for performance evaluation of networks-on-chip. In *2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 223–226. IEEE, 2009.

[103] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini. Xpipes: A latency insensitive parameterized network-on-chip architecture for multi-processor socs. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pages 45–48, Sept 2012.

[104] Shajulin Benedict. Energy-aware performance analysis methodologies for {HPC} architectures - an exploratory study. *Journal of Network and Computer Applications*, 35(6):1709 – 1719, 2012.

[105] R. Rao, S. Vrudhula, and K. Berezowski. Analytical results for design space exploration of multi-core processors employing thread migration. In *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*, pages 229–232, aug. 2008.

[106] B. Wojciechowski, K.S. Berezowski, P. Patronik, and J. Biernat. Fast and accurate thermal simulation and modelling of workloads of many-core processors. In *Thermal Investigations of ICs and Systems (THERMINIC), 2011 17th International Workshop on*, pages 1–6, sept. 2011.

[107] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94, May 2000.

[108] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. *SIGARCH Comput. Archit. News*, 28(2):95–106, May 2000.

[109] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mary Jane Irwin, N. Vijaykrishnan, Mahmut Kandemir, Tao Li, and Lizy Kurian John. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, pages 141–, Washington, DC, USA, 2002. IEEE Computer Society.

[110] Gilberto Contreras, Margaret Martonosi, Jinzhang Peng, Guei-Yuan Lueh, and Roy Ju. The xtrem power and performance simulator for the intel xscale core: Design and experiences. *ACM Trans. Embed. Comput. Syst.*, 6(1), February 2007.

[111] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*, pages 294–305, 2002.

[112] A.B. Kahng, Bin Li, Li-Shiuan Peh, and K. Samadi. Orion 2.0: A power-area simulator for interconnection networks. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(1):191–196, Jan 2012.

[113] Sheng Li, Jung Ho Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480, Dec 2009.

[114] A. Flores, J.L. Aragon, and M.E. Acacio. Sim-powercmp: A detailed simulator for energy consumption analysis in future embedded cmp architectures. In *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, volume 1, pages 752–757, may 2007.

[115] Dam Sunwoo, G.Y. Wu, N.A. Patil, and D. Chiou. Presto: An fpga-accelerated power estimation methodology for complex systems. In *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, pages 310–317, Aug 2010.

[116] Wim Heirman, Souradip Sarkar, Trevor E. Carlson, Ibrahim Hur, and Lieven Eeckhout. Power-aware multi-core simulation for early design stage hardware/software co-optimization. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 3–12, New York, NY, USA, 2012. ACM.

[117] Shah Mohammad Faizur Rahman, Qing Yi, and Houman Homayoun. Just-in-time component-wise power and thermal modeling. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, CF '15, pages 2:1–2:8, New York, NY, USA, 2015. ACM.

[118] Antonio Laganà, Alessandro Costantini, Osvaldo Gervasi, Noelia Faginas Lago, Carlo Manuali, and Sergio Rampino. Compchem: progress towards gems a grid empowered molecular simulator and beyond. *Journal of Grid Computing*, 8(4):571–586, 2010.

[119] Neelam Saboo, Arun Kumar Singla, Joshua Mostkoff Unger, and Laxmikant V. Kalé. Emulating petaflops machines and blue gene. In *Proceedings of the 15th International Parallel &Amp; Distributed Processing Symposium*, IPDPS '01, pages 195–, Washington, DC, USA, 2001. IEEE Computer Society.

[120] Terry L Wilmarth and Laxmikant V Kale. Pose: Getting over grainsize in parallel discrete event simulation. In *Parallel Processing, 2004. ICPP 2004. International Conference on*, pages 12–19. IEEE, 2004.

[121] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, pages 126–131. IEEE, 2008.

[122] Brad Penoff, Alan Wagner, Michael Tuxen, and Irene Rungeler. Mpi-netsim: A network simulation module for mpi. In *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*, pages 464–471. IEEE, 2009.

[123] Luca Benini, Davide Bertozzi, Alessandro Bogliolo, Francesco Menichelli, and Mauro Olivieri. Mparm: Exploring the multi-processor soc design space with systemc. *Journal of VLSI signal processing systems for signal, image and video technology*, 41(2):169–182, 2005.

[124] Jason Cong, Karthik Gururaj, Guoling Han, Adam Kaplan, Mishali Naik, and Glenn Reinman. Mc-sim: An efficient simulation tool for mpsoc designs. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 364–371. IEEE Press, 2008.

[125] Jerome Chevalier, Olivier Benny, Mathieu Rondonneau, Guy Bois, El Mostapha Aboulhamid, and F-R Boyer. Space: a hardware/software systemc modeling platform including an rtos. 2003.

[126] Marius Monton, Antoni Portero, Marc Moreno, Borja Martinez, and Jordi Carrabina. Mixed sw/systemc soc emulation framework. In *2007 IEEE International Symposium on Industrial Electronics*, pages 2338–2341. IEEE, 2007.

[127] Claude Helmstetter and Vania Joloboff. Simsoc: A systemc tlm integrated iss for full system simulation. In *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, pages 1759–1762. IEEE, 2008.

[128] Daniel D Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, and Shuqing Zhao. *SpecC: specification language and methodology*. Springer Science & Business Media, 2012.

[129] Richard Buchmann, F Pétrot, and A Greiner. Fast cycle accurate simulator to simulate event-driven behavior. In *Electrical, Electronic and Computer Engineering, 2004. ICEEC'04. 2004 International Conference on*, pages 35–38. IEEE, 2004.

[130] Jason Cong, Karthik Gururaj, Guoling Han, Adam Kaplan, Mishali Naik, and Glenn Reinman. Mc-sim: An efficient simulation tool for mpsoc designs. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '08, pages 364–371, Piscataway, NJ, USA, 2008. IEEE Press.

[131] Tse-Chen Yeh and Ming-Chao Chiang. On the interfacing between qemu and systemc for virtual platform construction: Using dma as a case. *Journal of Systems Architecture*, 58(3):99–111, 2012.

[132] Davide Quaglia, Franco Fummi, Maurizio Macrina, and Saul Saggin. Timing aspects in qemu/systemc synchronization. In *Proc. of the Int. QEMU Users' Forum*, pages 11–14, 2011.

[133] Ming-Chao Chiang, Tse-Chen Yeh, and Guo-Fu Tseng. A qemu and systemc-based cycle-accurate iss for performance estimation on soc development. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):593–606, 2011.

[134] Tse-Chen Yeh, Guo-Fu Tseng, and Ming-Chao Chiang. A fast cycle-accurate instruction set simulator based on qemu and systemc for soc development. In *MELECON 2010-2010 15th IEEE Mediterranean Electrotechnical Conference*, pages 1033–1038. IEEE, 2010.

[135] Markus Becker, Henning Zabel, and Wolfgang Müller. Qemu/systemc cosimulation at different abstraction levels. In *1st International QEMU Users Forum*, volume 32, 2011.

[136] Tse-Chen Yeh and Ming-Chao Chiang. On the interface between qemu and systemc for hardware modeling. In *2010 International Symposium on Next Generation Electronics*, pages 73–76. IEEE, 2010.

[137] Daniele Bortolotti, Andrea Marongiu, and Luca Benini. Virtualsoc: A research tool for modern mpsocs. *ACM Trans. Embed. Comput. Syst.*, 16(1):3:1–3:27, October 2016.

[138] Jason E Miller, Harshad Kasture, George Kurian, Charles Gruenwald, Nathan Beckmann, Christopher Celio, Jonathan Eastep, and Anant Agarwal. Graphite: A distributed parallel simulator for multicores. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12. IEEE, 2010.

[139] Shuai Jiao, Da Wang, Xiaochun Ye, Weizhi Xu, Hao Zhang, and Ninghui Sun. Partitionsim: A parallel simulator for many-cores. In *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 119–126. IEEE, 2012.

[140] George Kurian, Sabrina M Neuman, George Bezerra, Anthony Giovinazzo, Srinivas Devadas, and Jason E Miller. Power modeling and other new features in the graphite simulator. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 132–134. IEEE, 2014.

[141] Shafagh Jafer, Qi Liu, and Gabriel Wainer. Synchronization methods in parallel and distributed discrete-event simulation. *Simulation Modelling Practice and Theory*, 30(0):54 – 73, 2013.

[142] Huiwei Lv, Yuan Cheng, Lu Bai, Mingyu Chen, Dongrui Fan, and Ninghui Sun. P-gas: Parallelizing a cycle-accurate event-driven many-core processor simulator using parallel discrete event simulation. In *Principles of Advanced and Distributed Simulation (PADS), 2010 IEEE Workshop on*, pages 1–8, may 2010.

[143] Zheng Cao, Jianwei Xu, Mingyu Chen, Gui Zheng, Huiwei Lv, and Ninghui Sun. Hppnetsim: a parallel simulation of large-scale interconnection networks. In *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, pages 32:1–32:8, San Diego, CA, USA, 2009. Society for Computer Simulation International.

[144] Edgar A. León, Rolf Riesen, Arthur B. Maccabe, and Patrick G. Bridges. Instruction-level simulation of a cluster at scale. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 3:1–3:12, New York, NY, USA, 2009. ACM.

[145] R Ubal, J Sahuquillo, S Petit, and P López. Multi2sim: A simulation framework to evaluate multicore-multithread processors. In *IEEE 19th International Symposium on Computer Architecture and High Performance computing, page (s)*, pages 62–68, 2007.

[146] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI - The Complete Reference: The MPI-2 Extensions*, volume 2. MIT Press, Cambridge, MA, 1998.

[147] Zhangxi Tan, Andrew Waterman, Rimas Avizienis, Yunsup Lee, Henry Cook, David Patterson, and Krste Asanović. Ramp gold: an fpga-based architecture simulator for multiprocessors. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 463–468, New York, NY, USA, 2010. ACM.

[148] Zhangxi Tan, Andrew Waterman, Henry Cook, Sarah Bird, Krste Asanović, and David Patterson. A case for fame: Fpga architecture model execution. *SIGARCH Comput. Archit. News*, 38(3):290–301, June 2010.

[149] D. Chiou, Dam Sunwoo, Joonsoo Kim, N.A. Patil, W. Reinhart, D.E. Johnson, J. Keefe, and H. Angepat. Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 249–261, dec. 2007.

[150] Eric S. Chung, Eriko Nurvitadhi, James C. Hoe, Babak Falsafi, and Ken Mai. Protoflex: Fpga-accelerated hybrid functional simulation. Technical report, 2007.

[151] N. Agarwal, T. Krishna, Li-Shiuan Peh, and N.K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 33–42, April 2009.

[152] Jean-Michel Berge, Alain Fonkoua, Serge Maginot, and Jacques Rouillard. Verilog and vhdl, 1992.

[153] P. Mishra and N. Dutt. Architecture description languages for programmable embedded systems. *Computers and Digital Techniques, IEE Proceedings* -, 152(3):285–297, may 2005.

[154] Eric M Dashofy, André Van der Hoek, and Richard N Taylor. A highly-extensible, xml-based architecture description language. In *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*, pages 103–112. IEEE, 2001.

[155] Barnes Christopher and Jaehwan John Lee. A dynamically configurable discrete event simulation framework for many-core chip multiprocessors. 2015.

[156] James Michael Schmidt. Towards many-core processor simulation on cloud computing platforms. Master's thesis, PURDUE UNIVERSITY, USA, july-aug. 2011.

[157] Jan Kuper, Christiaan Baaij, Matthijs Kooijman, and Marco Gerards. Exercises in architecture specification using c$\lambda$ash. In *Proceedings of Forum on Specification and Design Languages, FDL 2010*, pages 178–183, Gières, France, September 2010. ECSI Electronic Chips & Systems design Initiative.

[158] R. Wester, C.P.R. Baaij, and J. Kuper. A two step hardware design method using c$\lambda$ash. In *22nd International Conference on Field Programmable Logic and Applications, FPL 2012*, pages 181–188. IEEE Computer Society, August 2012.

[159] Christiaan Baaij, Matthijs Kooijman, Jan Kuper, Arjan Boeijink, and Marco Gerards. C$\lambda$ash: Structural descriptions of synchronous hardware using haskell. In *Proceedings of the 13th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools*, pages 714–721.

IEEE Computer Society, September 2010.

[160] Lieven Eeckhout and Koen De Bosschere. Speeding up architectural simulations for high-performance processors. *SIMULATION-TRANSACTIONS OF THE SOCIETY FOR MODELING AND SIMULATION INTERNATIONAL*, 80(9):451–468, 2004.

[161] Benard Xypolitidis, Rudin Shabani, Satej V Khandeparkar, Zain Ul-Abdin, Tomas Nordstr, et al. Towards architectural design space exploration for heterogeneous manycores. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 805–810. IEEE, 2016.

[162] Tao Huang, Xue Li, Ting Zhang, and De-Tang Lu. Gpu-accelerated direct sampling method for multiple-point statistical simulation. *Computers & Geosciences*, 57:13–23, 2013.

[163] S. Hassani, G. Southern, and J. Renau. Livesim: Going live with microarchitecture simulation. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 606–617, March 2016.

[164] David Meisner, Junjie Wu, and Thomas F Wenisch. Bighouse: A simulation infrastructure for data center systems. In *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*, pages 35–45. IEEE, 2012.

[165] Lieven Eeckhout, Hans Vandierendonck, and Koen Bosschere. Workload design: Selecting representative program-input pairs. In *Parallel Architectures and Compilation Techniques, 2002. Proceedings. 2002 International Conference on*, pages 83–94. IEEE, 2002.

[166] Michael Van Biesbrouck, Lieven Eeckhout, and Brad Calder. Representative multiprogram workloads for multithreaded processor simulation. In *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, pages 193–203. IEEE, 2007.

[167] Maximilien B Breughe and Lieven Eeckhout. Selecting representative benchmark inputs for exploring microprocessor design spaces. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4):37, 2013.

[168] AJ KleinOsowski, John Flynn, Nancy Meares, and David J. Lilja. Workload characterization of emerging computer applications. chapter Adapting the SPEC 2000 Benchmark Suite for Simulation-based Computer Architecture Research, pages 83–100. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[169] Irfan Uddin. Design space exploration in the microthreaded many-core architecture. *arXiv preprint arXiv:1309.5551*, 2013.

[170] Shin-haeng Kang, Donghoon Yoo, and Soonhoi Ha. Tqsim: A fast cycle-approximate processor simulator based on qemu. *Journal of Systems Architecture*, 2016.

[171] P. D. Bryan, J. A. Poovey, J. G. Beu, and T. M. Conte. Accelerating multithreaded application simulation through barrier-interval time-parallelism. In *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 117–126, Aug 2012.

[172] Lau Mai Chan and Rajagopalan Srinivasan. A hybrid cpu-graphics processing unit (gpu) approach for computationally efficient simulation-optimization. *Computers & Chemical Engineering*, 2016.

[173] Michael C Fu, Fred W Glover, and Jay April. Simulation optimization: a review, new developments, and applications. In *Proceedings of the 37th conference on Winter simulation*, pages 83–95. Winter Simulation Conference, 2005.

[174] J. Wu, X. Zhu, T. Li, and X. Sui. Wbsp: A novel synchronization mechanism for architecture parallel simulation. *IEEE Transactions on Computers*, 65(3):992–1005, March 2016.

[175] Marcus Eggenberger, Manuel Strobel, and Martin Radetzki. Globally asynchronous locally synchronous simulation of nocs on many-core architectures. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 763–770. IEEE, 2016.

[176] Z. Yu, Y. Chen, Y. Song, and S. Huang. Comparison of parallel implementations of controls on gpu for transient simulation of power system. In *2016 35th Chinese Control Conference (CCC)*, pages 9996–10001, July 2016.

[177] N. Belhadj, N. Bahri, Z. Marrakchi, M. A. Ben Ayed, N. Masmoudi, and H. Mehrez. H.264/avc high definition intra coding implementation on multiprocessor system on chip technology architecture. *IET Computers Digital Techniques*, 9(5):259–267, 2015.

[178] D. Genbrugge, S. Eyerman, and L. Eeckhout. Interval simulation: Raising the level of abstraction in architectural simulation. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12, jan. 2010.

[179] Joe H Novak and Erik Brunvand. Using fpgas to prototype a self-timed floating point co-processor. In *Custom Integrated Circuits Conference, 1994., Proceedings of the IEEE 1994*, pages 85–88. IEEE, 1994.

[180] Jose Nunez-Yanez and Geza Lore. Enabling accurate modeling of power and energy consumption in an arm-based system-on-chip. *Microprocessors and Microsystems*, 37(3):319 – 332, 2013.

[181] Abderazek BenA., Yoshinaga Tsutomu, and Sowa Masahiro. High-level modeling and fpga prototyping of produced order parallel queue processor core. *The Journal of Supercomputing*, 38(1):3–15, 2006.

[182] J. Wawrzynek, D. Patterson, M. Oskin, Shin-Lien Lu, C. Kozyrakis, J.C. Hoe, D. Chiou, and K. Asanovic. Ramp: Research accelerator for multiple processors. *Micro, IEEE*, 27(2):46–57, march-april 2007.

[183] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz. Ramp blue: A message-passing manycore system in fpgas. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 54–61, aug. 2007.

[184] Benjamin C. Lee and David M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *SIGOPS Oper. Syst. Rev.*, 40(5):185–194, October 2006.

[185] Reiner Jung, Robert Heinrich, and Wilhelm Hasselbring. Geco: A generator composition approach for aspect-oriented dsls. In *International Conference on Theory and Practice of Model Transformations*, pages 141–156. Springer, 2016.

[186] Meriem Chibani, Brahim Belattar, and Abdelhabib Bourouis. Toward an aspect-oriented simulation. *International Journal of New Computer Architectures and their Applications (IJNCAA)*, 3(1):1–10, 2013.

[187] Tudor B Ionescu, Andreas Piater, Walter Scheuermann, and Eckart Laurien. An aspect-oriented approach for the development of complex simulation software. *Journal of Object Technology*, 9(1):161–181, 2010.

[188] Meriem Chibani, Brahim Belattar, and Abdelhabib Bourouis. The use of the aspect oriented programming (aop) paradigm in discrete event simulation domain: Overview and perspectives. In *The Third International Conference on Digital Information Processing and Communications*, pages 653–660. The Society of Digital Information and Wireless Communication, 2013.

[189] Meriem Chibani, Brahim Belattar, and Abdelhabib Bourouis. Practical benefits of aspect-oriented programming paradigm in discrete event simulation. *Modelling and Simulation in Engineering*, 2014:47, 2014.

[190] Jean Bézivin. *Model Driven Engineering: An Emerging Technical Space*, pages 36–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[191] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society, 2007.

[192] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of computer programming*, 72(1):31–39, 2008.

[193] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. Atl: a qvt-like transformation language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 719–720. ACM, 2006.

[194] Lei Yonglin, Wang Weiping, Li Qun, and Zhu Yifan. A transformation model from {DEVS} to {SMP2} based on {MDA}. *Simulation Modelling Practice and Theory*, 17(10):1690 – 1709, 2009.

[195] Abdurrahman Alshareef, Hessam S. Sarjoughian, and Bahram Zarrin. An approach for activity-based devs model specification. In *Proceedings of the Symposium on Theory of Modeling & Simulation*, TMS-DEVS '16, pages 25:1–25:8, San Diego, CA, USA, 2016. Society for Computer Simulation International.

[196] Byunghun Lee, Dae-Kyoo Kim, Hyosik Yang, and Sungsoo Oh. Model transformation between {OPC} {UA} and {UML}. *Computer Standards and Interfaces*, 50:236 – 250, 2017.

[197] Hessam S. Sarjoughian, Abdurrahman Alshareef, and Yonglin Lei. Behavioral devs metamodeling. In *Proceedings of the 2015 Winter Simulation Conference*, WSC '15, pages 2788–2799, Piscataway, NJ, USA, 2015. IEEE Press.

[198] Heng He1, Ruixuan Li, Xinhua Dong1, Zhi Zhang, and Hongmu Han. An efficient and secure cloud-based distributed simulation system. *Applied*

*Mathematics and Information Sciences*, 6(3):729–736, May 2012.

[199] Georgia Sakellari and George Loukas. A survey of mathematical models, simulation approaches and testbeds used for research in cloud computing. *Simulation Modelling Practice and Theory*, 39(0):92 – 103, 2013. S.I.Energy efficiency in grids and clouds.

[200] Faruk Caglar, Shashank Shekhar, Aniruddha Gokhale, Satabdi Basu, Tazrian Rafi, John Kinnebrew, and Gautam Biswas. Cloud-hosted simulation-as-a-service for high school {STEM} education. *Simulation Modelling Practice and Theory*, 58, Part 2:255 – 273, 2015. Special issue on Cloud Simulation.

[201] Wenhong Tian, Minxian Xu, Aiguo Chen, Guozhong Li, Xinyang Wang, and Yu Chen. Open-source simulators for cloud computing: Comparative study and challenging issues. *Simulation Modelling Practice and Theory*, 58, Part 2:239 – 254, 2015. Special issue on Cloud Simulation.

[202] Bogumił Kamiński and Przemysław Szufel. On optimization of simulation execution on amazon {EC2} spot market. *Simulation Modelling Practice and Theory*, 58, Part 2:172 – 187, 2015. Special issue on Cloud Simulation.

[203] Lutz Schubert and Alexander Kipp. Principles of service oriented operating systems. In Pascale Vicat-Blanc Primet, Tomohiro Kudoh, and Joe Mambretti, editors, *Networks for Grid Applications*, volume 2 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 56–69. Springer Berlin Heidelberg, 2009.

[204] The S(o)OS Consortium . S(o)os (service-oriented operating system): Resource-independent execution support on exa-scale systems. http://www.soos-project.eu/, 2010-2013. [Online: accessed 5-September-2014].

[205] Javad Zarrin, Rui L. Aguiar, and João Paulo Barraca. Elcore: Dynamic elastic resource management and discovery for future large-scale many-core enabled distributed systems. *Microprocessors and Microsystems*, pages –, 2016.

[206] Tommaso Cucinotta. Challenges in operating system design for future many-core systems. All Hands Meeting (AHM) 2010, Cardiff, UK, Available at http://retis.sssup.it/~tommaso/presentations/AHM-2010.pdf, 2010. [Online: accessed 15-April-2016].

[207] Javad Zarrin, Rui L. Aguiar, and João Paulo Barraca. Dynamic, scalable and flexible resource discovery for large-dimension many-core systems. *Future Generation Computer Systems*, 53:119 – 129, 2015.

[208] Lutz Schubert. Dynamicity requirements in future cloud-like infrastructures. Invited Speaker, EuroCloud CLASS Conference, Available at http://videolectures.net/classconference2012_schubert_infrastructures/, 2012. [Online: accessed 15-April-2016].

[209] Javad Zarrin, Rui L. Aguiar, and Joao Paulo Barraca. A self-organizing and self-configuration algorithm for resource management in service-oriented systems. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*, pages 1–7, June 2014.

[210] Lutz Schubert, Alexander Kipp, and Stefan Wesner. Above the clouds: From grids to service-oriented operating systems. In *Future Internet Assembly*, pages 238–249, 2009.

[211] Sunil Kumar, Tommaso Cucinotta, and Giuseppe Lipari. A latency simulator for many-core systems. In *Proceedings of the 44th Annual Simulation Symposium*, ANSS '11, pages 151–158, San Diego, CA, USA, 2011. Society for Computer Simulation International.

[212] Rui L. Aguiar, Diogo Gomes, JoaoPaulo Barraca, and Nuno Lau. Cloud-thinking as an intelligent infrastructure for mobile robotics. *Wireless Personal Communications*, 76(2):231–244, 2014.