SST + gem5 = A Scalable Simulation Infrastructure for High Performance Computing

Mingyu Hsieh
Sandia National Labs P.O.Box 5800
Albuquerque, NM
myhsieh@sandia.gov

Jie Meng
Boston University
ECE Department
Boston, MA
jiemeng@bu.edu

Michael Levenhagen
Sandia National Labs
P.O.Box 5800
Albuquerque, NM
mjleven@sandia.gov

Kevin Pedretti
Sandia National Labs
P.O.Box 5800
Albuquerque, NM
ktpedre@sandia.gov

Ayse Coskun
Boston University
ECE Department
Boston, MA
acoskun@bu.edu

Arun Rodrigues
Sandia National Labs
P.O.Box 5800
Albuquerque, NM
afrodri@sandia.gov

ABSTRACT

High Performance Computing (HPC) faces new challenges in scalability, performance, reliability, and power consumption. Solving these challenges will require radically new hardware and software approaches. It is impractical to explore this vast design space without detailed system-level simulations. However, most of the existing simulators are either not sufficiently detailed, not scalable, or cannot evaluate key system characteristics such as energy consumption or reliability.

To address this problem, we integrate the highly detailed gem5 performance simulator into the parallel Structural Simulation Toolkit (SST). We add the fast-forwarding capability in the SST/gem5 and port the lightweight Kitten operating system on gem5. In addition, we improve the reliability model in SST with a comprehensive analysis of system reliability. Utilizing the simulation framework, we evaluate the impact of two energy-efficient resource-conscious scheduling policies on system reliability. Our results show that the effectiveness of scheduling policies differ according to the composition of workload and system topology.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*Simulation*

General Terms

Performance, Reliability

Keywords

Simulation, Architecture

Simutools 2012, March 19-23, Desenzano del Garda, Italy
Copyright © 2012 ICST 978-1-936968-47-3
DOI 10.4108/icst.simutools.2012.247745

1. INTRODUCTION

As HPC continues to push the bounds of computation, it faces new challenges in scalability, performance, reliability, and power consumption. To address these challenges, we need to design new architectures, operating systems, programming models, and software. It is impractical to explore this vast design space without detailed system-level simulations. However, most simulators are either not sufficiently detailed (i.e., glossing over important details in processor, memory, or network implementations), not scalable (i.e., simulating highly parallel architectures with serial simulators), or not able to evaluate key system characteristics such as energy consumption or reliability.

To address this problem, we have integrated the instruction-level gem5 simulator into the parallel Structural Simulation Toolkit (SST) [17]. We extend the SST/gem5 simulator by adding fast-forwarding capabilities, porting the HPC-oriented Kitten operating system [12], and adding reliability analysis capabilities. We evaluate the benefits of the SST/gem5 framework by analyzing scheduling policies. Specifically, this paper makes the following contributions:

- We create a new tool for exploring key issues in HPC by combining the capabilities of gem5 (detailed processor and cache models, and full-system simulation ability) and SST (a rich set of HPC-oriented architecture components, scalable parallel simulation, and the ability to analyze power and reliability).
- We add fast-forwarding features in the simulation framework, enabling multiple levels of granularity for the same component in the same simulation.
- We integrate the Kitten operating system with gem5. In comparison to Linux system, Kitten's lightweight kernel enables faster simulation and prototyping of system software and run-time management ideas.
- We extend the reliability model of SST by implementing the Monte Carlo method and adding a model for NBTI failure mechanism, enabling more comprehensive system reliability analysis. Utilizing the integrated simulation framework, we evaluate the impact of two resource-conscious scheduling policies on system reliability.

The rest of the paper starts with a discussion of the background and related work. Section 3 describes the integration of SST and gem5. Extensions to the simulation framework are described in Section 4. Section 5 demonstrates the evaluation results and is followed by the conclusion.

2. BACKGROUND

2.1 SST

The Structural Simulation Toolkit (SST) [17] is an open-source, multi-scale, and parallel architectural simulator that aims at the design and evaluation of HPC systems. The core of SST utilizes a component-based event simulation model built on top of MPI for efficient parallel simulations. The hardware devices, such as processors, are modeled as components in SST. The component-based modular interface of SST enables the integration of existing simulators (such as gem5) into a scalable simulation framework. SST has been validated with two real systems [9]. The SST's main multicore processor model, `genericproc`, is derived from the SimpleScalar [4]. It couples multiple copies of the sim-out-order pipeline model with a front-end emulation engine executing the PowerPC ISA. `Genericproc` has a cache coherency model, a prefetcher, and a refactored memory model that can be connected with more accurate memory models, such as DRAMSim2 [18]. However, there are limitations such as genericproc has a simple cache coherency protocol and non-configurable memory hierarchy. These limitations drive the need for more detailed processor models in SST.

2.2 gem5

The gem5 simulator [2] is an event-driven performance simulation framework for computer system architecture research. It models major structural simulation components (such as CPUs, buses, and caches) as objects, and supports performance simulations in both full-system and syscall-emulation modes. The full-system mode simulates a complete computer system including operating system kernel and I/O devices, while syscall-emulation mode simulates statically compiled binaries by functionally emulating necessary system calls. The memory subsystem in gem5 models inclusive/exclusive cache hierarchies with various replacement policies, the implementation of coherence protocols, DMA, and memory controllers. The gem5 has been validated with real systems with about 10% error.

2.3 Related Work

There are a number of tools in the area of microarchitecture-level performance and power modeling. For example, SimpleScalar [4] is integrated with several power models, such as Wattch [3], for examining the trade-off of performance and power. However, SimpleScalar runs only user-mode single-threaded workloads and cannot simulate multiple processor cores. Recently, Lis et al. present HORNET, a parallel manycore simulator with power and thermal modeling [14]. HORNET does not have a modular design to allow integration of other architectural models not shipped with the package. In addition, it uses ORION [11] for power estimation which focuses on power modeling of network components.

McPAT is an integrated power, area, and timing modeling framework for multicore architectures [13]. A widely used tool for modeling temperature is HotSpot, which provides steady-state and transient temperature responses given chip properties and power traces [10]. The SST/gem5 framework includes both McPAT and HotSpot for power and thermal modeling. The modular design of SST eases integration of existing simulators and the interactions among simulators in a parallel, scalable, and open-source framework.

A distinguishable feature of SST/gem5 is the ability to evaluate the impact of scheduling policies on system reliability. Prior research has introduced performance and energy-aware job scheduling policies that take into account of application memory characteristics [6, 15]. Merkel et al. [15] employ task activity vectors to characterize applications based on their resource utilization and schedules applications to improve system performance and energy efficiency. Fedorova et al. [6] develop the Power Distributed Intensity (Power DI) scheduling policy that clusters the computation bound applications on as few machines and as few memory domains as possible. However, none of these resource-conscious scheduling policies take into consideration of system reliability. In this work, we use the integrated SST/gem5 framework to evaluate two scheduling policies by examining which policy achieves better system reliability.

3. INTEGRATION OF SST AND GEM5

In this section, we provide the details of the integration of gem5 and SST. The integration requires synchronizing the clock of gem5 and SST, working around synchronous messages, and allowing the communication between gem5 and other SST components. We test the integration with two examples: (1) a Portals 4 [16] offload NIC and router; (2) an external memory model DRAMSim2 [18].

3.1 System Emulation Mode Integration

Figure 1 shows a high level view of how gem5 is encapsulated as an SST component. On each rank, all gem5 `SimObjects` live inside an `sst::component`. Gem5 event queue is modified so that it is driven by the SST event queue. The gem5 is triggered by an SST clock event and the gem5 event queue delivers events which are scheduled to arrive at that cycle. Gem5 then stops and returns control back to SST until the next clock interval.

There are some changes to the initialization. Gem5 traditionally uses Python to dynamically build the simulation. However, Python is not available on some large HPC systems. In addition, SST uses a two-stage initialization pro-

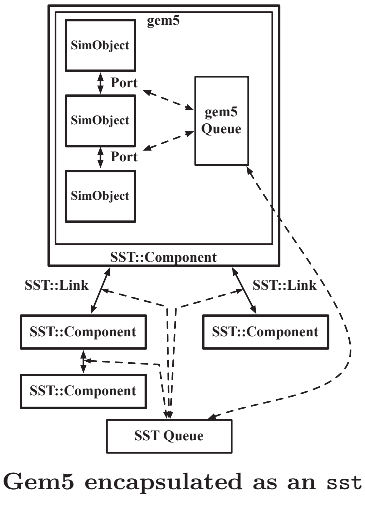

Figure 1: Gem5 encapsulated as an `sst::component`

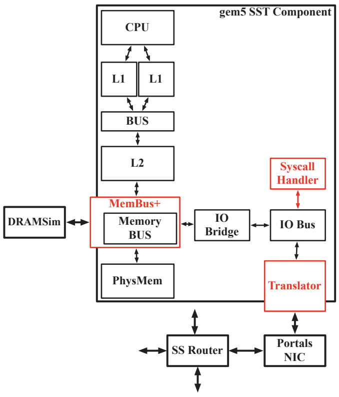

Figure 2: Translation and assistance objects to interface gem5 with SST and allow parallel operations.

cess (components are partitioned before they are constructed), which is not compatible with gem5's initialization method. Thus, we repackage gem5 as a static library and then directly call the `SimObject` constructors. The configuration is controlled by an XML schema.

3.1.1 Working Around Synchronous Messages

A key difference between gem5's ports and SST's `links` is that ports have a synchronous interface, which allows instant untimed communication between two `SimObjects`. This convenience is used for some "backdoor" operations such as system calls, loading binaries into memory, and for debugging. The instant communication is enabled by the fact that gem5 is a serial simulator where all `SimObjects` exist in the same address space. However, SST does not allow the synchronous interface, as the greater scalability of MPI comes at the cost of a distributed memory model.

For system calls, the system emulation mode of gem5 uses synchronous access to the physical memory to copy data directly to/from the processor, which evokes the host OS to emulate the calls. For example, in a `write()` call the processor directly accesses the buffer in the main memory `SimObject` and passes that to the host OS. To avoid the synchronous call, we add a system call handler `SimObject`. All system calls are transformed into memory mapped requests to an uncached region of memory. These requests are then directed to the system call handler `SimObject`, which can issue DMA transactions to copy buffers to/from memory and then perform the required call to the host OS. This comes at the cost of having to extend the system call handler for each system call, but the number which requires emulation is relatively small for our applications.

Another use of synchronous messages is to perform the initial loading of the binary into main memory. Normally, the gem5 processor directly accesses the main memory object. For the integrated SST/gem5, we instead tell the memory object (which may reside outside of the gem5 component) to load the data directly. The gem5 cpu object then only loads the initial thread state from the binary.

3.1.2 SST/gem5 Translators

The integrated gem5 must be able to interact with other components in SST. This requires new `simObjects` to be added to gem5. These objects should be aware of SST and able to communicate over sst `links` (see Figure 2). The first example of this connects gem5 to DRAMSim2−−a robust and validated DRAM memory model. To support the connection to DRAMSim2, a new `simObject` is created which inherites from the gem5 memory bus. This new object, `memBusPlus`, replaces the default memory bus, and passes incoming memory requests through an SST `link` to the DRAMSim2 `component`. DRAMSim2 is modified to include a backing store of data (by default DRAMSim2 only provides timing, not actual storage), allowing it to replace the gem5 physical memory model. DRAMSim2 is also modified to load the application binary into its backing store, as mentioned in Section 3.1.1.

To perform larger HPC network experiments, gem5 is also interfaced with models of an HPC network. The SST contains models of a high performance protocol offload NIC which uses the Portals network API. This Portals NIC can connect to a cycle-accurate model of an HPC 3D torus router, based on the Red Storm SeaStar network [21]. This setup allows a detailed processor and cache model (gem5) to be connected to a detailed HPC NIC model (Portals NIC) and a detailed HPC router model (the SeaStar router).

Integrating the Portals/SeaStar models with gem5 starts with creating a translator `simObject` inside of gem5. This object is designed as a memory-mapped device within gem5 and can be accessed by writing into a reserved uncached part of the address space from the gem5 CPU model. The memory bus and IO bridge diverts accesses to this address space to the translator object which would buffer them until a mailbox address is written to. When this occurs, the buffered data would be assembled into a Portals message and this message would be transferred over a SST `link` to the Portals NIC `component`. The Portals NIC could also send events to the translator object to notify the processor of incoming messages or to start DMA transfers to or from memory. Because communication between SST `components` (such as the Portals NIC and the SeaStar router) can be serialized and passed between ranks of SST, the combination of gem5, the Portals NIC, and the SeaStar router allow detailed exploration of HPC network protocols and parameters in parallel. Experiments (See Section 3.3) show that running this configuration in the SST's parallel simulation environment achieves significant simulation speedup.

3.2 Full System Mode (gem5_FS) Integration

Running the gem5 full system model is similar to running the system emulation model except that the gem5_FS additionally needs to load a Linux kernel, and mounts one or more disk images for its filesystems. The disk image should contain the application binaries that one wants to run. The path to the kernel image and the disk image need to be set up when configuring gem5_FS. Integrating the gem5_FS with SST is similar to integrating the system emulation mode described in previous sections. For example, the `simObjects` that are created and inherited from the gem5 in-order CPUs, out-of-order (O3) CPUs, caches, and physical memory are kept the same. On the other hand, the interface for gem5_FS and SST integration no longer needs the workload `simObject` (the application that is assigned to a core in system emulation mode) and adds several `simObjects` for full system components, such as interrupts. The path to the system files (kernel and disk images) is configured by the SST's System Description Language (SDL). Because the full system mode does not require synchronous messages for system calls, additional handler objects are not needed.

Table 1: Speedup from parallel simulation

Number of host ranks	Execution time (s)
1	24869
32	578

3.3 Effectiveness of Parallelism of SST/gem5

In order to test the effectiveness of parallelism of SST/gem5, we conduct experiments which combines the gem5 O3 processor model (Alpha ISA) with the Portals NIC and the SeaStar router models as described in Section 3.1.2. The gem5 is run in system emulation mode and with only a single core per simulated rank. Our test machine is a small 4-socket 32-core x86 machine running Redhat Enterprise Linux 5 and OpenMPI 1.4.3. The SST can run over distributed memory machines even the system is using a shared memory. In these experiments, the processors are running the miniGhost compact application from the Mantevo Suite [1]. MiniGhost performs a simple 3D multi-point stencil computation on a regular grid. We compare the execution time of running the simulation with 128 nodes in serial with running the simulation on 32 host ranks. Our results demonstrate the gem5 benefits from the parallel simulation which results in a speedup of 43x (Table 1). Although these results are only for small systems, the ability to run even a hundred gem5 cores in a single scalable simulation with a realistic HPC network represents a significant increase in what is previously possible with gem5. Also, because each gem5 component can be configured with multiple cores, even a 128 node simulation could be a simulation of several thousand cores, which fits well into the range of encountering interesting HPC effects.

3.4 Kitten on gem5 Integration

Kitten is an open-source, lightweight kernel designed to operate as the compute node operating system on distributed memory supercomputers [1]. We integrate Kitten with gem5 for two main technical reasons. First, the simpler code-base system enables more rapid prototyping and evaluation of new system software and runtime ideas in comparison to a full Linux system. Second, lightweight kernels in general can enable faster and more reproducible simulation compared to a Linux system. For example, the authors of [7] point out that CNK, a lightweight kernel for PowerPC with similar design to Kitten, is an essential tool for chip verification. A full Linux kernel requires days to boot in cycle-accurate simulator, while CNK requires only a couple of hours. We modify Kitten to support embedding its equivalent of an initramfs file in the Kitten kernel image, resulting in a single file for the gem5 to boot. Two additional minor changes are required: 1) The gem5 is modified to pass “console=serial,vga” to the kernel being booted, and 2) the vendor string returned by the CPUID instruction is modified to return “GenuineIntel”. With these changes, Kitten is able to boot in a gem5 simulation and run user-level processes and threads.

We explore novel power management schemes which require low-overhead task migration mechanisms, as well as frequent experimentation with different task scheduling policies. Such changes are relatively straightforward to implement in the Kitten kernel. As an example, we implement the V1 optimized core-switching strategy described in [20] in Kitten in approximately a day of effort. Since neither the Linux sys_gettid() modifications or gettid() benchmark used in the paper are publicly available, we re-implement them in Linux 2.6.35.7 so that Kitten could be compared to Linux. The results of this comparison are shown in Table 2, where the gettid benchmark performs 1,000,000 iterations per run. Our results are from evaluations on an Intel X5570 2.93GHz CPU, while prior work [20] experiments on an Intel X5160 3.0GHz CPU. It is interesting to note that Kitten achieves a speedup of 1.69 compared to Linux running on the same hardware, which is better than the best speedup of 1.43 achieved in [20]. Kitten is expected to perform better for more complex V2 or V3 algorithms in [20].

Table 2: Core-switching performance between two cores in the same processor

	unmodified gettid()	modified gettid() with context-switch
Kitten	44 ns	2630 ns
Linux 2.6.35.7	47 ns	4435 ns
V1 Linux [20]	83 ns	4094 ns
Best Linux [20]	83 ns	2870 ns

4. EXTENSIONS TO THE SST/GEM5

In this section, we introduce the extensions to SST/gem5 by adding the fast-forwarding facility and improving the reliability model. The fast-forwarding feature enables multiple levels of granularity for the same component within the same simulation (Section 4.1). The enhancement of reliability model in SST/gem5 provides a more comprehensive analysis of system reliability (Section 4.2).

4.1 Fast-Forward

In order to accelerate the simulation speed running real benchmark suites, we add the fast-forwarding feature in SST/gem5. It enables the simulations to reach the region of interest (ROI) of applications at a faster speed. We create a new `O3switchCPU` object, which consists of a simple gem5 in-order CPU object and a detailed gem5 out-of-order (O3) CPU object. The gem5 in-order CPU object uses atomic memory access and is suitable for the cases where a detailed model is not necessary. The gem5 O3 CPU has detailed timing memory access for running precise simulations. When the `O3switchCPU` object is initialized, it first starts the simulation process with the in-order CPU. After reaching the start point of the ROI, it switches out the in-order CPU and makes the O3 CPU take over the process. The numbers of instructions to fast-forward and to execute within ROI are configurable in SST's SDL.

The effectiveness of fast-forwarding is highly dependent on the system architectural configurations (such as the system bus-width, memory access rate, and cache / memory access latencies), the number of instructions before reaching the ROI, and the benchmarks. To evaluate the effectiveness of fast-forwarding in SST/gem5, we run each of the eight NAS Parallel Benchmarks (Figure 3) on a single node with 16 threads. This is to avoid the simulation speedup owing to fast-forwarding being overshadowed by the synchronization between processes of workloads running on multiple nodes. For each benchmark, we fast-forward 50% of the operations and the results show an average increase of 47% in simulation speed (Figure 3). The largest improvement is for the most memory-bound “mg” benchmark, which is

[1] http://code.google.com/p/kitten/

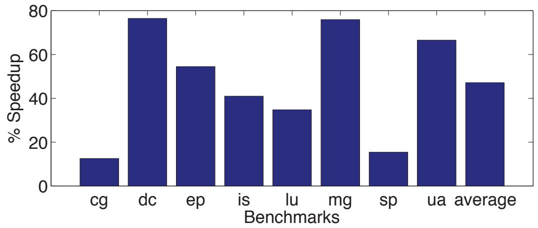

Figure 3: Fast-forward performance improvements

75.90% faster with fast-forwarding than with the standard O3 CPU model. For the most computationally intensive benchmark "cg", using fast-forwarding could also make simulation 12.57% faster. In general, benchmarks that have more memory accesses tend to benefit more by using fast-forwarding in their simulations.

4.2 Reliability Modelling

We implement the technology interface for the power, temperature, and reliability simulation in SST to help guide the design and operation of future computers [8, 9]. We add a model for another emerging critical failure mechanism, Negative Bias Temperature Instability (NBTI). NBTI typically occurs when the input to a gate is low while the output is high, resulting in an accumulation of positive charges in the gate oxide. This accumulation causes the threshold voltage of the transistor to increase and eventually leads to processor failure due to timing constraints. The NBTI model is defined in [19] and its failure rate is given by:

$$\lambda_{NBTI} = C_{NBTI}[(\ln(\frac{A}{1+2e^{\frac{B}{kT}}}) - \ln(\frac{A}{1+2e^{\frac{B}{kT}}} - C))\frac{T}{e^{\frac{-D}{kt}}}]^{\frac{1}{\beta}} \quad (1)$$

where C_{NBTI}, A, B, C, D, and β are fitting parameters. The values that we use are C_{NBTI} = 0.006, A = 1.6328, B = 0.07377, C = 0.01, D = -0.06852, and β= 0.3.

In our prior work [8], the system reliability is modeled based on two assumptions which limit the applicability of the model. First, the model assumes that the simulated system is a serial failure system. Second, the model assumes a constant failure rate in each failure mechanism. In this work, we address the limitations of the two assumptions and enable the reliability model to evaluate structural duplication with consideration of wear-out failure mechanism.

We implement the Monte Carlo method and the MIN-MAX analysis in the reliability model, which are able to calculate the system-level reliability of serial-parallel or all-serial failure system [19, 5]. Also, we use lognormal distribution for each failure mechanism, which has been found to better model wear-out mechanism of semiconductors [19]. In each iteration of the Monte-Carlo simulation, we generate a random lifetime from lognormal distribution for each failure mechanism and component of the system as in Equation (2):

$$rand_{lognormal} = e^{ln(MTTF)-0.125+0.5sin(2\pi rand1)\sqrt{-2\ln(rand2)}} \quad (2)$$

where $rand_{lognormal}$ is a random lognormal distribution representing a random lifetime for each structure. MTTF is provided by the reliability model described earlier. $rand1$ and $rand2$ are two random uniform variables. A MIN-MAX analysis is performed on these lifetimes based on the configuration of the system and gives the system-level lifetime for that iteration. The MTTF of the system is calculated by repeating this process over many iterations (1000 iterations in our study) and averaging the system-level lifetimes.

Table 3: SPEC2006 benchmark scenarios

Scenario	Benchmarks
1	four c
2	three c and one m
3	two c and two m
4	one c and three m
5	four m

5. THE IMPACT OF SCHEDULING POLICIES ON SYSTEM RELIABILITY

5.1 Experimental Setup

We model a dual-socket Intel Xeon processor comprising eight cores in total. Each socket has two chips, and each chip has one L2 cache shared by a pair of cores. When multiple cores share a resource, the threads running on those cores can either constructively or destructively use this resource depending on if the threads share data or not. We study all combinations of the 4-thread workloads that are constructed from 12 representative SPEC2006 benchmarks (computation-bound: libquantum, namd, specrand_fp, specrand_int, astar; memory-bound: bzip2, gcc, gobmk, h264ref, hmmer, omnetpp, mcf). Table 3 lists the five SPEC2006 scenarios, where c and m stand for computation-bound and memory-bound benchmarks respectively. We evaluate the Power DI policy and the vector balancing policy with frequency scaling. The results are shown in Figure 5 and 6.

5.2 Evaluation

We show the impact of resource contention on system performance, energy, and reliability in Figure 4. In the simulation, we consider two thread-to-core mapping scenarios. In one scenario, each of the eight cores is assigned with a memory-bound application (gcc). In the other scenario, each core is assigned with a computation-bound application (libquantum). Simulation results show that in the memory-bound case, the number of L2 cache access is about 3,400 times more than in the computation-bound case. As shown in Figure 4, memory-bound threads can contend for shared resources, destructively increase energy consumption and temperature, and degrade reliability.

We compare the Energy-Delay Product (EDP) and the system reliability yield by the two scheduling policies. Figure 5 shows that the Power DI policy results in better EDP when the fraction of the memory-bound tasks in the workload is lower. The effectiveness of scheduling policies differs according to the number of memory-bound benchmarks in each Scenario. For example, in Scenario 1, the two policies allocates the computation-bound tasks to cores differently. Power DI clusters all tasks to the cores in the same socket,

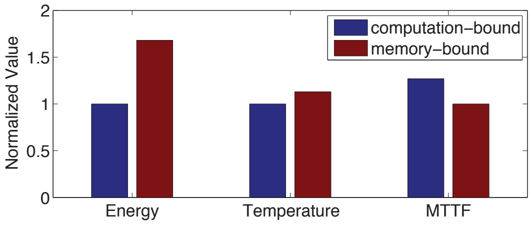

Figure 4: The effect of application characteristics on system energy, temperature, and reliability

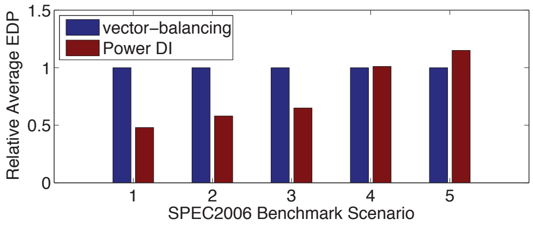

Figure 5: The effect of scheduling policies on EDP

while the vector balancing policy spreads the tasks among all cores. The frequency scaling of the vector balancing policy is not invoked in this scenario. The Power DI policy, on the other hand, can save power by switching the idle cores to a low power state. On the other hand, all the tasks in Scenario 5 are memory-bound, and the best policy is to spread them across memory domains so that any two tasks would not end up running on the same memory domain. In this case, the vector balancing policy improves EDP by saving energy from frequency scaling, while the power saving strategy in the Power DI policy is not invoked. Figure 6 shows the impact of the two policies on system reliability. We assume a serial-parallel failure system, where cores in the same socket are in series (failure of any core results in the socket failure) and the two sockets are in parallel (system fails when both sockets fail). Results show that the Power DI policy yields better system reliability when the fraction of the memory-bound tasks in the workload is low. In Scenario 1, the Power DI policy clusters all tasks in the same socket while leaving the cores on the other socket idle, thus achieves much better reliability. It is worth noting that when the system topology changes, the impact of scheduling policies on system reliability changes as well.

6. SUMMARY

In this paper, we have introduced the integration of SST and gem5. We have enhanced the integrated framework by adding the Kitten OS to provide a flexible HPC-oriented OS for architectural experimentation, and the fast-forwarding feature to accelerate the simulation speed. We have also improved the reliability model in SST. Experiments have demonstrated that the SST/gem5 integration is scalable. This allows simulating the impact of many effects, such as load imbalance or system overheads, that are only visible at scale. We have used the simulation framework to evaluate two resource-conscious job-scheduling policies. Results show that the effectiveness of both polices on system energy and reliability depends on workload and the system topology. Our future work will use the integrated SST/gem5 infrastructure to develop novel temperature management techniques with consideration of application characteristics to optimize the performance, energy consumption, and reliability of multithreaded multicore systems.

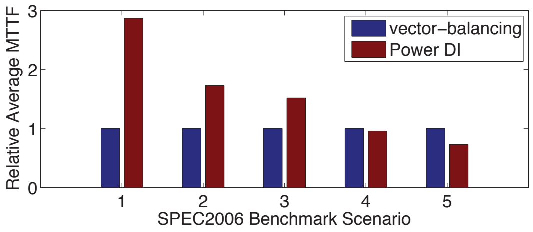

Figure 6: The effect of scheduling policies on system reliability

Acknowledgments

Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] *Mantevo Project.* https://software.sandia.gov/mantevo/.

[2] N. Binkert et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39:1–7, 2011.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture (ISCA)*, pages 83–94, 2000.

[4] D. Burger and T. Austin. The simplescalar tool set 2.0. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[5] A. K. Coskun, T. S. Rosing, K. Mihic, Y. Leblebici, and G. D. Micheli. Analysis and optimization of mpsoc reliability. *Journal of Low Power Electronics (JOLPE)*, 2(1):56–69, April 2006.

[6] A. Fedorova, S. Blagodurov, and S. Zhuravlev. Managing contention for shared resources on multicore processors. *Communications of the ACM*, 53(2):49–57, 2010.

[7] M. Giampapa et al. Experiences with a lightweight supercomputer kernel: Lessons learned from blue gene's cnk. In *International Conference on High-Performance Computing, Networking, Storage, and Analysis (SC)*, November 2010.

[8] M. Hsieh. A scalable simulation framework for evaluating thermal management techniques and the lifetime reliability of multithreaded multicore systems. In *IEEE Workshop on Thermal Modeling and Management (IGCC-TEMM)*, 2011.

[9] M. Hsieh, A. Rodrigues, R. Risen, K. Thompson, and W. Song. A framework for architecture-level power, area, and thermal simulation and its application to network-on-chip design exploration. *ACM SIGMETRICS Performance Evaluation Review*, 38:63–68, 2011.

[10] W. Huang et al. Differentiating the roles of ir measurement and simulation for power and temperature-aware design. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2009.

[11] A. Kahng et al. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *Design Automation and Test in Europe (DATE)*, April 2009.

[12] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen, and R. Brightwell. Palacios and kitten: New high performance operating systems for scalable virtualized and native supercomputing. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2010.

[13] S. Li et al. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *International Symposium on Microarchitecture*, 2009.

[14] M. Lis et al. Scalable, accurate multicore simulation in the 1000-core era. In *ISPASS*, 2011.

[15] A. Merkel, J. Stoess, and F. Bellosa. Resource-conscious scheduling for energy efficiency on multicore processors. In *Proceedings of the 5th European conference on Computer systems*, pages 153–166. ACM, 2010.

[16] R. E. Riesen, K. T. Pedretti, R. Brightwell, B. W. Barrett, K. D. Underwood, T. B. Hudson, and A. B. Maccabe. The Portals 4.0 message passing interface. Technical Report SAND2008-2639, Sandia National Laboratories, April 2008.

[17] A. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. The structural simulation toolkit. *SIGMETRICS Perform. Eval. Rev.*, 38:37–42, 2011.

[18] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2. http://www.ece.umd.edu/dramsim/, July 2010.

[19] Srinivasan et al. Exploiting structural duplication for lifetime reliability enhancement. In *ISCA*, pages 520–531, June 2005.

[20] R. Strong, J. Mudigonda, J. C. Mogul, N. Binkert, and D. Tullsen. Fast switching of threads between cores. *ACM SIGOPS Operating Systems Review*, 43:35–45, April 2009.

[21] K. Underwood, M. Levenhagen, and A. Rodrigues. Simulating red storm: Challenges and successes in building a system simulation. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10, 2007.