

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

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

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

Jie Meng  
Boston University  
College of Engineering  
Boston, MA  
jiemeng@bu.edu

Ayse Coskun  
Boston University  
College of Engineering  
Boston, MA  
acoskun@bu.edu

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

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 at some scale. 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 simulator into the parallel Structural Simulation Toolkit (SST). We add the fast-forward capability in the SST/gem5 and ported the lightweight Kitten operating system on gem5. In addition, we improve the reliability model in SST with more comprehensive analysis of system reliability. Utilizing the simulation framework, we evaluate the impact of two energy-efficient resource-conscious scheduling policies on system reliability. 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*SIMUTools 2012* March 19–23, Desenzano, Italy.  
Copyright 2012 ICST, ISBN .

## 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 at some scale. However, most simulators are either not sufficiently detailed (often glossing over important details in processor, memory, or network implementations), not scalable (often trying to simulate 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 highly detailed gem5<sup>1</sup> simulator into the parallel Structural Simulation Toolkit (SST) [11]. We extend the gem5 simulator by adding fast-forwarding capabilities, porting the HPC-oriented Kitten operating system [8], and adding reliability analysis capabilities. Experimental results show the benefits gained from the combination of gem5, SST and Kitten. Specifically, this paper makes the following contributions:

- We create a useful new tool for exploring key issues in HPC, by leveraging the combined capabilities of gem5 (strong processor models, detailed and flexible cache models, and a full-system simulation capacity) and SST (has a rich set of HPC-oriented architecture components, is scalable parallel simulation, and has the ability to analyze power and reliability).
- We add the fast-forward facility in the simulation framework, enabling multiple levels of granularity for the same component in the same simulation.
- We integrate the lightweight Kitten operating system with gem5, enabling more rapid evaluation of system software and run-time ideas.
- We extend the reliability model of SST, enabling SST to analyze system reliability more comprehensively. Utilizing the integrated simulation framework, we evaluate the impact of two resource-conscious scheduling policies on system reliability.

<sup>1</sup>[http://gem5.org/Main\\_Page/](http://gem5.org/Main_Page/)

The rest of the paper starts with the background and related work. Section 3 describes the integration of SST and gem5. Several improvements 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) [11] is an open-source, multi-scale, and parallel architectural simulator aimed 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. The SST has been validated with two real systems [7]. The SST tends to include existing simulation models as is, and to validate the SST, the simulation models themselves need to be accurate. For example, the gem5 has been validated with real systems with about 10% error.

The SST’s main multi-core processor model, `genericproc`, is descended 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. In addition to the functionalities SimpleScalar provides, `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 [12]. However, it has limitations, such as it adopts a simple cache coherency protocol and its memory hierarchy is not configurable. This drives the need for a more detailed and extensive processor model in SST.

### 2.2 gem5

The gem5 simulator 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 the gem5 models inclusive/exclusive cache hierarchies with various replacement policies, coherence protocol implementations, DMA, and memory controllers.

### 2.3 Related Work

There has been a number of tools in the area of computer architectural performance and power modeling. For example, SimpleScalar [4] is extended with several power models, such as Wattch [3], for examining trade-offs of performance and power. However, it 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 [9]. However, it does not have a modular design to allow integration of other architectural models not shipped with the package. Moreover, it uses ORION [2] for power estimation which

only provides power modeling for network components. The modular design of SST eases integration of existing simulators to interact with each other in a parallel, scalable, and open-source framework.

## 3. INTEGRATION OF SST AND GEM5

In this section, we provide the details of the integration of gem5 and SST. The integration requires synchronizing gem5’s clock with SST’s, working around synchronous messages, and allowing communication with other SST components. We test the integration with two examples: (1) with a Portals 4 [10] offload NIC and router; (2) with an external memory model DRAMSimII [12].

### 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 `SimObject`s live inside an `sst::component`. The 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 is scheduled to arrive at that cycle. It then stop and return control back to SST until the next clock interval.

There are some changes to the initialization as well. The gem5 traditionally uses Python to dynamically build the simulation. However, Python is not available on some large HPC systems. In addition, SST’s uses a two-stage initialization process (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. Configuration is controlled by an XML schema.

#### 3.1.1 Working Around Synchronous Messages

One key difference between gem5’s ports and SST’s `links` is that ports have an 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. It is enabled by the fact that gem5 is a serial simulator where all `SimObjects` exist in the same address space. SST does not allow the same convenience, as the greater scalability of MPI comes at the cost of a distributed memory model.

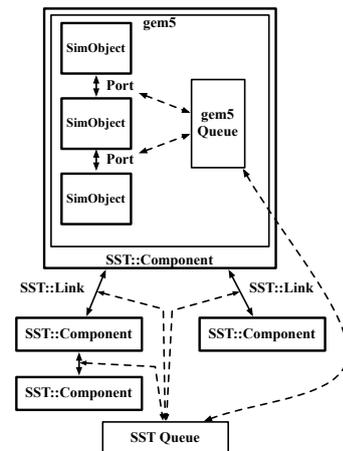
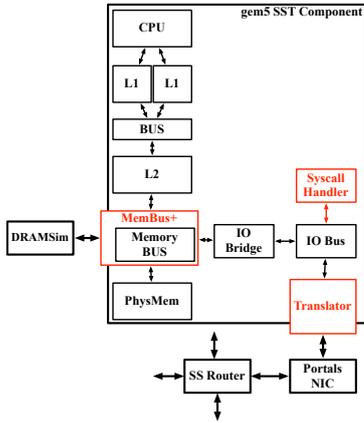


Figure 1: Gem5 encapsulated as an `sst::component`



**Figure 2: Translation and assistance objects to interface gem5 with SST and allow parallel operation**

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 invokes the host OS to emulate the call. For example, in a `write()` call the processor directly accesses the buffer in the main memory `SimObject` and pass 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 that new `simObjects` be added to gem5 which are aware of SST and able to communicate over sst links (see Figure 2). The first example of this connects gem5 to DRAMSim II—a very robust and highly validated DRAM memory model. To support this, a new `simObject` is created which inherits from the gem5 memory bus. This new object, `memBusPlus`, replaces the default memory bus, and would pass incoming memory requests through an SST link to the DRAMSim component. DRAMSim is modified to include a backing store of data (by default DRAMSim only provides timing, not actual storage), allowing it to replace the gem5 physical memory model. DRAMSim 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 [15]. 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 in to 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 was written to. When this occurs, the buffered data would be assembled into a Portals message and this message would be transferred over an 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 were 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 are added 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, integrating full system mode does not require some additional handler objects.

## 3.3 Effectiveness of Parallelism of SST/gem5

To test the effectiveness of parallelism of SST/gem5, we conduct experiments combined the gem5 O3 processor model (Alpha ISA) with the Portals NIC and the SeaStar router models described in Section 3.1.2. The gem5 was run in system emulation mode and with only a single core per simulated rank. Our test machine was a small 4-socket 32-core x86 machine running Redhat Enterprise Linux 5 and OpenMPI 1.4.3. Though this is a shared memory machine, the SST can run over distributed memory machines as well. In these experiments, the processors were 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

**Table 1: Speedup from parallel simulation**

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

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). Though 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 was 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 put it 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.<sup>2</sup> We integrate Kitten with gem5 for two main technical reasons. First, the simpler code-base compared to a full Linux system enables more rapid prototyping and evaluation of new system software and runtime ideas. Second, lightweight kernels in general can enable faster and more reproducible simulation compared to a Linux system. For example, the authors of [5] 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 require days to boot in cycle-accurate simulator, while CNK require only a couple of hours.

Kitten is designed to run on commodity x86 systems, so the port to the gem5-FS x86 architecture model was straightforward. Kitten boots identically to Linux, however the gem5’s boot mechanism does not support separate Linux kernel and initramfs files. 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 were required: 1) The gem5 was modified to pass “console=serial,vga” to the kernel being booted, and 2) the vendor string returned by the CPUID instruction was modified to return “GenuineIntel”. With these changes, Kitten was able to boot in a gem5 simulation and run user-level processes and threads. Applications being simulated can use the standard gem5 utility library identically to how it is used in a Linux simulation. For example, the application source code can be modified to call the gem5 library’s exit function when the simulation is complete, instruct the simulator to dump out performance statistics, or cause a checkpoint to be taken.

One of the first areas we are exploring are novel power management schemes. These require low-overhead task migration mechanisms, as well 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 [14] 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. Kitten and Linux 2.36.25.7 re-

<sup>2</sup><http://code.google.com/p/kitten/>

**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 [14]	83 ns	4094 ns
Best Linux [14]	83 ns	2870 ns

sults are from evaluations on an Intel X5570 2.93GHz CPU, while prior work [14] experiments on an Intel X5160 3.0GHz CPU. The results of this comparison are shown in Table 2, where the gettid benchmark performs 1,000,000 iterations per run. 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 [14]. It is likely Kitten would perform even better if the more complex V2 or V3 algorithms from the paper were implemented. In this instance, Kitten is able to achieve best-in-class performance without having to resort to lengthy optimization endeavors.

## 4. EXTENSIONS TO THE SST/GEM5

In this section, we introduce the extensions to the SST/gem5 by adding the fast-forward facility, which enables multiple levels of granularity for the same component within the same simulation (Section 4.1). We also extend and improve the reliability model of SST, which provides a more comprehensive analysis of system reliability (Section 4.2).

### 4.1 Fast-Forward

To accelerate the simulation speed running real benchmark suites, we add the fast-forwarding feature in SST/gem5 to enable the simulations to reach the region of interest (ROI) 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 cases where a detailed model is not necessary. The gem5 O3 CPU has detailed timing memory access for running precise simulation. 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 number 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 and memory access latencies), the number of instructions to reach the ROI, and the benchmark. 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 overshadowed by the synchronization between processes for 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-bounded “mg” benchmark, which is 75.90% faster with fast-forwarding than with the standard O3 CPU model. For the most computationally intensive

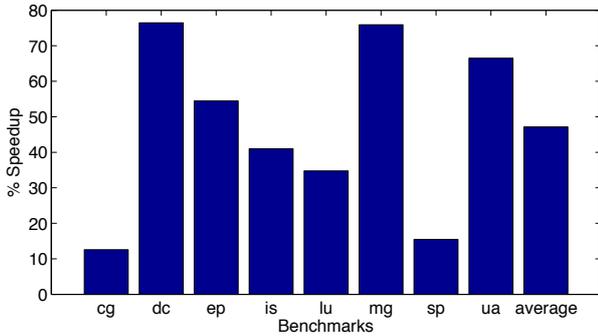


Figure 3: Fast-forward performance improvements

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

To help guide design, program and operate future computers, in [6, 7], we implement the technology interface, the core of power, temperature and reliability simulation in SST. In this work, 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 we use is defined in [13] and its failure rate is given by:

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

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

Our previous reliability model [6] was based on two assumptions which limit the applicability of the model. First, the model assumed the simulated system is a serial failure system. Second, the model assumed a constant failure rate in each failure mechanism. In this work, we address the two limitations 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 [13], which are able to calculate the system-level reliability of either a serial-parallel failure system or a serial failure system. Second, we use lognormal distribution for each failure mechanism, which has been found to better model wear-out mechanism of semiconductors. In any single iteration of the Monte-Carlo simulation, Equation (2) is used to generate a random lifetime from lognormal distribution for each failure mechanism and structure of the system:

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

where  $rand_{lognormal}$  is a random lognormal distribution representing a random lifetime for each structure, MTTF is provided by the reliability model described earlier, and  $rand1$  and  $rand2$  are two random uniform variables. A MIN-MAX analysis is performed on these lifetimes based

Table 3: SPEC2006 scenarios used for evaluation of scheduling policies

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

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 obtained.

## 5. THE IMPACT OF SCHEDULING POLICIES ON SYSTEM RELIABILITY

### 5.1 Experimental Setup

We model a dual-socket Intel Xeon processor comprising eight total cores. 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 conduct a study on all possible 4-thread workloads that can be constructed from the 12 representative SPEC2006 benchmarks (memory-bound benchmarks: bzip2, gcc, gobmk, h264ref, hmmmer, omnetpp, mcf; computation-bound benchmarks: libquantum, namd, specrand\_fp, specrand\_int, astar). We choose all four SPEC2006 scenarios listed in Table 3, where c and m respectively stand for computation-bound and memory-bound benchmarks. We compare the Power DI policy and the vector balancing policy plus the frequency scaling. The results are shown in Figure 5 and Figure 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 Scenario 1, 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 the computation-bound case. As shown in Figure 4, memory-bound threads can contend for shared resource and destructively interfere increasing energy consumption and temperature and degrade reliability.

We compare the Energy-Delay Product (EDP) and the

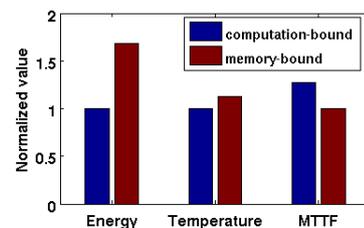


Figure 4: The effect of application characteristics on system energy, temperature, and reliability

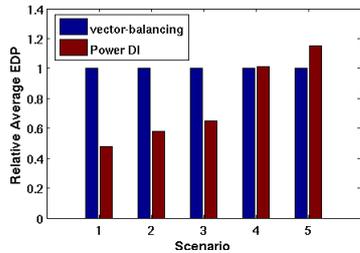


Figure 5: The effect of scheduling policies on EDP

system reliability yield by the two scheduling policies. Figure 5 shows that in general the Power DI policy yields better EDP when the fraction of the memory-bound tasks in the workload is lower. The effectiveness of scheduling policies differ according to the the number of memory-bound benchmarks. For example, in Scenario 1, the computation-bound tasks are allocated to cores differently by the two policies (Power DI clusters all tasks to the cores in the same socket 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 lower power state. Conversely, in Scenario 5 all the tasks are memory-bound, and the best policy was to spread them across memory domains so no two tasks would end up running on the same memory domain. In this scenario, 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 the system reliability. We assume a serial-parallel failure system, where cores in the same socket are in series (failure of any one 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 lower. In Scenario 1, the Power DI policy clusters all tasks in the same socket, leaving the cores on the other socket idle, and therefore achieves much better reliability. It is worth noting that when the system topology changes, the impact of scheduling policies on system reliability will change 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 HPC OS to provide a flexible HPC-oriented OS and fast-forwarding to accelerate the simulation speed. We have also improved the reliability model in SST. Experiments have demonstrated that the SST/gem5 integration is scalable, allowing simulations of many effects that are only visible at scale, such as load imbalance or system overheads. We have used the simulation framework to eval-

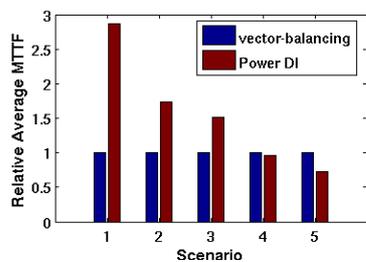


Figure 6: The effect of scheduling policies on system reliability

uate two resource-conscious job-scheduling policies. Results show that although both policies are designed to be energy-efficient, their effectiveness on energy and reliability depends on workload and the system topology. Our future work will use this infrastructure to create novel temperature management techniques with consideration of application characteristics to optimize performance, energy consumption, and reliability of multithreaded multicore systems.

## 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.

## 7. REFERENCES

- [1] *Mantevo Project Home Page*. <https://software.sandia.gov/mantevo/>.
- [2] L.-S. P. Andrew Kahng, Bin Li and K. Samadi. 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.
- [3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, 2000.
- [4] D. Burger and T. Austin. The simplescalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, 1997.
- [5] M. Giampapa, T. Gooding, T. Inglett, and R. W. Wisniewski. Experiences with a lightweight supercomputer kernel: Lessons learned from blue gene's cnk. In *SC '10: Proceedings of the International Conference on High-Performance Computing, Networking, Storage, and Analysis*, November 2010.
- [6] M. Hsieh. A scalable simulation framework for evaluating thermal management techniques and the lifetime reliability of multithreaded multicore systems. In *1st International IEEE Workshop on Thermal Modeling and Management (TEMM 11)*, July.
- [7] M. Hsieh, R. Riesen, K. Thompson, W. Song, and A. Rodrigues. Sst: A scalable parallel framework for architecture-level performance, power, area and thermal simulation. *The Computer Journal*, 55:181–191, 2012.
- [8] 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 *24th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2010)*, April 2010.
- [9] M. Lis, P. Ren, M. H. Cho, K. S. Shim, C. W. Fletcher, O. Khan, and S. Devadas. Scalable, accurate multicore simulation in the 1000-core era. *Performance Analysis of Systems and Software, IEEE International Symposium on*, 0:175–185, 2011.
- [10] 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.
- [11] A. F. 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, March 2011.
- [12] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2. <http://www.ece.umd.edu/dramsim/>, July 2010.
- [13] Srinivasan et al. Exploiting structural duplication for lifetime reliability enhancement. In *ISCA*, pages 520–531, June 2005.
- [14] 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.
- [15] K. Underwood, M. Levenhagen, and A. Rodrigues. Simulating red storm: Challenges and successes in building a system simulation. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10. IEEE, 2007.