The Structural Simulation Toolkit

Arun F. Rodrigues
Sandia National Labs
Albuquerque, NM
afrodri@sandia.gov

Jeanine Cook
New Mexico State University
Las Cruces, NM
jecook@nmsu.edu

Elliott Cooper-Balis
University of Maryland
College Park, MD
ecc17@umd.edu

K. Scott Hemmert
Sandia National Labs
Albuquerque, NM
kshemme@sandia.gov

Chad Kersey
Sandia National Labs
Albuquerque, NM
cdkerse@sandia.gov

Rolf Riesen
Sandia National Labs
Albuquerque, NM
rolf@sandia.gov

Paul Rosenfeld
University of Maryland
College Park, MD
prosenfeld@gmail.com

Ron Oldfield
Sandia National Labs
Albuquerque, NM
raoldfi@sandia.gov

Marlow Weston
Sandia National Labs
Albuquerque, NM
miwesto@sandia.gov

ABSTRACT

As supercomputers grow, understanding their behavior and performance has become increasingly challenging. New hurdles in scalability, programmability, power consumption, reliability, cost, and cooling are emerging, along with new technologies such as 3D integration, GP-GPUs, silicon-photonics, and other "game changers". Currently, they HPC community lacks a unified toolset to evaluate these technologies and design for these challenges.

To address this problem, a number of institutions have joined together to create the Structural Simulation Toolkit (SST), an open, modular, parallel, multi-criteria, multi-scale simulation framework. The SST includes a number of processor, memory, and network models. The SST has been used in a variety of network, memory, and application studies and aims to become the standard simulation framework for designing and procuring HPC systems.

Categories and Subject Descriptors

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

Keywords

Simulation, Architecture

1. INTRODUCTION

The HPC community has recognized that the development, procurement, and operation of large, capability-class supercomputers is necessary for the advancement of a range of scientific and technical challenges ranging from basic science to climate prediction to weapons design. As the HPC community reaches into the trans-petascale regime towards exascale, the task of building these computers is becoming increasingly difficult. In addition to the traditional challenges of raw performance and scaling, fundamental challenges in performance, power consumption[21], cost, reliability[16], programmability[32], and cooling arise. Overcoming these challenges will require new approaches to how we fabricate, architect, program, and operate future computers. Major changes will have to be made to the memory, network, processor, and IO subsystems, along with concurrent changes in the programming model and applications.

It is seldom practical to construct hardware prototype systems of sufficient size and number to explore this vast design space. Therefore, we will have to rely on simulation to guide many design and procurement decisions. Currently, the HPC architecture community lacks the tools needed for such evaluations. While a variety of simulators exist for individual system components, there is no unified framework to allow them to act in concert.

To address this problem, a number of institutions[1] have developed a simulation framework for simulating large-scale HPC systems. This simulator allows parallel simulation of large (tens to hundreds of thousands of nodes or more) machines at multiple levels of detail (from cycle-accurate execution-driven instruction-based to abstract message-driven simulation). It couples multiple models for processors, memory, IO, and network subsystems. This simulator, the Structural Simulation Toolkit (SST) aims to become the standard simulator framework for designing and procuring HPC systems by helping Industry, Academia, and the National Labs in designing and evaluating future architectures.

1.1 SST Requirements

Effective supercomputer design and evaluation requires a simulation environment for quickly simulating large HPC systems in a variety of ways. Some key requirements:

- **Scalable Parallel Simulation**: The simulation frame-

[1]Currently including Sandia National Labs, Oak Ridge National Laboratory, Micron Technologies, The University of Maryland, New Mexico State University, Georgia Tech, and Auburn

work must allow very large parallel simulations of even larger parallel machines. This will allow us to use the supercomputers of today to design and optimize the supercomputers and applications of tomorrow. Efficient parallel simulation will require built-in support for automatic partitioning, checkpointing, and event serialization.

- **Multiscale**: Different simulation models must allow either abstract or detailed evaluation of system components. This will allow different system characteristics to be evaluated at the necessary level of detail and accuracy, while still allowing other parts of the simulation to be performed in a faster but more abstract manner.
- **Holistic**: Raw performance is only one of several challenges for a codesigned system. The simulator should provide a unified interface to a variety of technology models, allowing components to easily estimate power, energy, area, cost, and reliability.
- **Open**: To be effective, the simulator must be accessible to as large an audience as possible, both from a legal and technical standpoint.

To meet these requirements, the SST is comprised of a simple simulation core that contains a parallel discrete event simulator and support services for simulation. `components`, representing hardware systems such as processors, network switches, or memory devices, interface with the simulation core to communicate and operate with a common notion of timeframe. The simulation core also provides support services such as power and area estimation, checkpointing, configuration/initialization of the simulation, and statistics gathering. The SST's modular interface eases the integration of existing simulators into a common framework and is licensed under a BSD-like license.

1.2 Parallel Simulation

The SST uses a parallel component-based discrete event simulation (DES) model layered on top of MPI. To achieve better performance, the SST uses a conservative (i.e. no rollback) distance-based [17] optimization. At the start of the simulation, the system topology is represented by a graph with components as nodes and connections between them as edges, with each edge labeled with the minimum latency between the connected components. The Zoltan [15] library is then used to partition components across the MPI ranks with the goal of balancing the load and partitioning across the highest latency links. Tests [29] indicate that the algorithm is scalable and shows less than 25% overhead at 128 ranks (11,904 simulated components) compared to a single rank for detailed simulations.

1.3 Multi-Scale

The SST includes a variety of processor, network, and memory models at different scales and levels of accuracy. This diversity allows the simulation user to make tradeoffs between accuracy, complexity, and time to solution, enabling an efficient design space exploration. The SST includes high level stochastic processor models (see Section 4.8) which take statistical representations of applications and run at faster than real-time speed and cycle-based detailed processor models based on SimpleScalar[12] (See section 4.1) which are driven by instruction execution. Network models range from detailed flit-level router models based on the RedStorm SeaStar router (See Section 4.6) to abstract models message-based models (See Section 4.4). Memory models include the highly detailed DRAMSim2 (See Section 4.2) and simple fixed latency models.

1.4 Holistic

Modern supercomputer design is complex multi-objective optimization in which execution speed must be balanced against power, energy, reliability, cost, and other factors. For example, current estimates of power consumption for an exascale machine using today's architectures range from hundreds of megawatts [20] to over a gigawatt[14]. In either case, with the cost of a megawatt-year of electricity being roughly $1 million[4], simply powering such a machine could cost hundreds of millions to billions of dollars a year. Clearly, this is infeasible.

To assist the designer with power and energy estimation, the SST provides a common interface to a variety of power estimation libraries including McPAT [23], Orion [7], and Sim-Panalyzer[3]. The interface also includes hooks to allow thermal modeling tools such as HotSpot[39] to be included. This basic interface can be extended to provide area, cost, and reliability estimates as well.

1.5 Open

The SST source code and most of its of components are licensed under a BSD-like license allowing free commercial and noncommercial use. However, this license is non-viral and the internal interfaces are design so that component writers do not have to expose any of their component internals. This allows commercial vendors to provide components without revealing the internal details of their implementation.

2. RELATED WORK

The SST simulation framework builds on a long tradition of architectural and network simulators such as M5[9], NS-3[18], and A-SIM[27]. In addition, it builds upon community experiences in modeling power dissipation[10][37]. The SST often seeks to directly include existing simulators to build a "best of breed" framework. The novel approach of the SST is to include these individual component models in a parallel, scalable, and open-source framework.

3. SST INTERNALS

The SST (Figure 1) consists of a simulator core, which provides simulation services, and pluggable `components` (see Section 4) which constitute the individual simulation models.

The simulator core provides simulation configuration and startup (Section 3.1), the parallel model of computation (Section 3.2.1), checkpointing (Section 3.3), and a common interface to the technology models.

3.1 Configuration and Job flow

The SST in configured with a XML file which lists the components instantiated in the simulation, any component parameters which must be passed in, the `links` between the components, and the latency on the component links. This configuration is processed into a graph, with the component

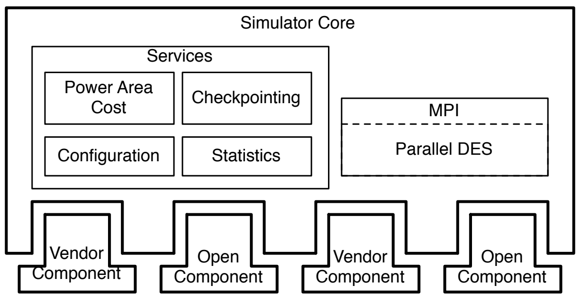

Figure 1: SST Structure

instances as nodes and the links between them as edges, which is then fed to the Zoltan libratry to find a partition which balances the number of components per host rank and which will maximimze the simulated lateny between components. Partitioning along high latency links means that rank will have to exchange messages less frequently in our conservative optimization.

3.2 Model of Computation

The simulation is carried out in a component-based discrete `event` model of computation. Each component can assign a clock to itself, to be triggered at regualar intervals. Components can also send events to other components along `links`, which have a minimum latency. When an `event` arrives at a component, it triggers an event handler function, in which the component can process and respond to the event. Alternately, the component can poll the `link` to recieve and process any outstanding messages.

3.2.1 Parallel Implementation

Parallelism is transparent to the component writer. Components interact through sending `events` to each other through `link` objects. All `events` inherit from a common base class, which also includes a time (see Section 3.2.2) tag to indicate when it should be delivered. All `events` must be serializable (using the Boost [1]Serialization Library), which can transform the `event` structure into a compact binary representation.

Whenever an `event` is sent, the SST core determines if the destination of the event is local (i.e. on the same MPI rank) or remote. Remote `events` are queued up for future delivery the next time the given ranks are due to synchronize. This occurs only as often as needed, based upon the latency patitioning given by the Zoltan library. I.e. if the components on two ranks are connected by a link with a minimum latency of 1000 ns, those ranks only need synchronize every 1000ns of simualted time.

The Boost MPI library is then used to perform the actual communication. When two ranks synchronize, they each serialize and send the pending `events` to each other. When the events are recieved, they are integrated with the local event queues, where they wait for delivery to their target components.

3.2.2 Time

Time in the simulator is represented using a single 64-bit unsigned integer to count the number of atomic timesteps that have passed since the beginning of simulation. The actual atomic timebase (time increment represented by each atomic timestep) is user programmable and has a default of 1 fs (10^{-12} seconds), which provides for over 200 days of simulated time. All times used by components and links are specified using strings (for example, "1.5 ns" or "1.73 GHz"), and are resolved at build time into a `TimeConverter` object. The `TimeConverter` object essentially represents a component's view of time and provides functions for converting from the component's timebase to the atomic timebase. The `TimeConverter` simply stores the number of atomic timesteps (refered to as its *factor*) in the desired time interval. In the case of a specified clock frequency, the factor represents the number of atomic timesteps in the clock period. For example, a component with a 1 GHz clock would get a `TimeConverter` object with a factor of 1000 (assuming the default atomic timebase of 1 fs), which would also be equal to the factor for 1 ns.

The component has two options when creating a `TimeConverter`. The first is to register a clock handler, in which case the handler is called once per clock period. The second is to simply register a timebase with the simulator, which can be used with the event driven interface. In either case the returned `TimeConverter` object is registered with the component's links, where it is used to convert latencies from the component's view of time to the atomic timebase. The use of `TimeConverters` insulates the components from both the need to know the value of the atomic timestep, as well as from knowing their own operating frequency. This allows a component be written with a generic timebase, which can be set at runtime.

3.3 Checkpointing

Because simulations may run for an extended period of time over a number of nodes, the simulator needs the ability to checkpoint and recover its state. To accomplish this, the simulator core uses the Boost Serialization Library to convert the core's state and the state of each component into a binary format. At a user defined interval, this binary state is dumped to a file which can be used to restart the simulation as needed.

4. COMPONENTS

4.1 genericProc

`genericProc` is a highly configurable multi-core processor simulator descended from the SimpleScalar[12] toolset. Specifically, it couples multiple copies of the `sim-out-order` pipeline model with a front-end emulation engine executing the PowerPC[26] ISA.

SimpleScalar is widely used in the architecture community and we have extended it with a cache coherency model, a prefetcher (using n-block lookahead), and by refactoring the memory model to allow connection with more accurate memory models, such as DRAMSim2. We have also added in event counting to help provide data for power/energy modeling. `genericProc` can be easily extended to access or control special hardware such as advanced memories or NICs. From the programmer's perspective this access can be done through overloading unused system calls or by a memory mapped interface. This makes the component useful for prototyping advanced processor features.

4.2 DRAMSim2

DRAMSim2[30] is a cycle accurate DDR2/3 memory system simulator developed at the University of Maryland. The simulator models a memory controller that receives memory transactions (read, write), converts them into DRAM device commands (RAS, CAS, PRE), and issues them to simulated ranks of DRAM devices. DRAMSim2 keeps track of the state of every bank and bus in the memory system and issues requests so that they do not violate DRAM timing constraints. The simulated memory controller can safely execute memory requests out of order while respecting potential dependences in the transaction stream. DRAM device timing and power consumption parameters along with system level parameters such as memory controller queue depths, queuing structures, address mapping scheme, and row buffer policy can be easily configured using a simple ini file. Device timing parameters can be obtained from manufacturer data sheets or can be tailored to reflect new or custom DDR devices. The output of the simulation includes bandwidth, latency, and power statistics both globally and per rank for each simulation epoch. Power computation is performed using an event counting methodology developed by Micron Technologies[19]. Additionally, a visualization tool that enables graphing and comparing DRAMSim2 simulation results is currently being developed.

One of the most important goals of DRAMSim2 is that it strives to be accurate. In addition to extensive testing by hand and manual analysis of simulation output, DRAMSim2 contains an HDL validation mode for automated testing. Any simulation can be configured to output a verification file which is turned into Verilog code that can be run using Micron's DDR2/3 behavioral Verilog models. These models do extensive checking for timing violations so one can be reasonably certain that if it passes this test, the simulation results are accurate. Many non-trivial DRAMSim2 simulations have been verified this way which is a good indication that the memory system model does not violate and of the many DDR timing constraints.

DRAMSim2 also has the goal of being simple to integrate into simulation frameworks, such as the SST. While making extensive use of the C++ STL data structures, DRAMSim2 requires no external library dependencies and has been successfully built on Linux, OSX, and Cygwin on Windows. The DRAMSim2 library has a straight forward interface which requires minimal wrapper code (less than 180 lines of code, including headers) to work with SST. The DRAMSimC SST component provides an interface to the DRAMSim2 library. Upon instantiation, the DRAMSimC component registers a callback function for completed requests and begins to issue a clock signal to DRAMSim2. DRAMSimC converts any incoming SST memory request messages into DRAMSim2 transactions. After the memory request completes, a certain number of clock ticks later, DRAMSim2 executes the response callback which is turned back into an SST message and sent back. This encapsulation allows any component to drive the DRAMSim simulation, such as a processor model or the included DRAMSimTraceC component which executes memory traces.

4.3 DiskSim

The long-term goal of our I/O simulation work is to develop a complete simulation framework to evaluate the scalability of experimental I/O systems and protocols. The first step toward that goal is demonstrate accurate simulation of existing disk-based storage technologies. Our SST components for disk simulation extend the functionality of the DiskSim software[11], a complex and well-proven simulation model capable of simulating a large variety of disks and storage topologies from all the major manufactures. If a particular disk is not explicitly supported by DiskSim through an existing parameter file, an additional tool called DIXtrac[33] is able to extract these parameters from a disk compliant with the SCSI protocol.

Our SST components for I/O are effectively lightweight wrappers around the DiskSim software, using it as a black box to provide accurate latency and bandwidth timings for block-based requests; however, integrating DiskSim with SST required a number of engineering fixes. First, we modified DiskSim to be 64-bit compliant to support current hardware architectures. After significant testing and validation, we submitted these changes back to the original developers for general distribution. Second, we developed "bridge" software to convert SST requests to DiskSim requests. Finally, we are making modifications to DiskSim to enable compatibility between the simulator clock used by SST.

At all levels of our DiskSim integration, we validated components by comparing simulated results to real measured values from the "skippy" and "seeker" benchmark codes [36]. The benchmark codes measure disk bandwidth, latency, rotational latency, head switch time, and cylinder switch time. All tests matched values to within reasonable error limits of the hardware.

4.4 Generic Router Models

The generic router model component can be used it situation where the simulation of a large network is required, but emphasis is on simulation of the endpoints and the detailed inner workings of the network routers are not as relevant. The router is a model of the type found in machines such as the Intel Paragon, ASCI Red, and to some extent Cray's XT line of machines.

Messages are wormhole routed[2] and use source-based routing.

When a message passes through a router, a configuraable hop delay is added to simulate processing of the route information. The router components act like full bandwidth preserving crossbar switches. If a path from an input port to an output port is available, the message is forwarded without further delays. If the output port is busy, the router component computes at what time the blocked message will be able to proceed and delays forwarding the event for the blocked message by that amount of time.

Using output port delays and input port event rescheduling, the router model can model congestion in the network, even though there is no flow control protocol between routers in place, and the links have, in essence, infinite capacity. Link bandwidth is a parameter passed to the router model which it uses to compute output port capacity and control the flow of outgoing data accordingly.

The router model does not support virtual channels. However, message deadlock cannot occur because messages are sent across the links in the form of single events, which do not prevent other messages from using the same links.

[2]Work is under way to allow flit-based routing. Dividing the available bandwidth among several simultaneous data flows is a better model, for NoC for example, than blocking entire messages.

The router model maintains a small number of counters to enable statistics on the number of messages coming in and going out of each port, how often congestion occurred and how much delay that caused. To provide power/energy consumption information, the McPAT or ORION power models can be enabled in a router component.

This generic router model component allows for a variety of topologies. Currently, there is support for two and three-dimensional meshes with or without wraparounds in the x, y, and z dimension, binary tree, binary fat tree, hypercube, a flattened two-dimensional butterfly, and a fully connected graph. An XML configuration file generation tool, *genTopo*, is included to make configuration of large networks easier.

4.5 The Communication Pattern SST Component

For many simulation studies, it is important to have realistic network traffic, but computation at the endpoints is of limited relevance. The communication pattern component of SST allows the generation of network traffic without incurring the processing and memory overhead of running a full endpoint simulator.

The only communication pattern implemented at the moment is ghost which simulates ghost cell exchanges on a five-point stencil operator where each rank communicates only with its East, West, South, and North neighbor. Implementations of communication patterns for FFT, the NAS parallel benchmark integer sort (IS), and master/slave are under way.

Each pattern generator is implemented as a state machine. They simulate compute time by suspending operations until a future event indicates the passing time and the need to transition to another state in the state machine. The state machines contain states for waiting for messages, if the algorithm has a dependency on incoming data. The state machines have additional states to enable various checkpoint/restart methods, the handling of faults, and the recovery after a fault.

4.6 Red Storm Router Model

The Red Storm router model is a near cycle accurate model of the SeaStar router using in the Cray XT3 through XT6 line of supercomputers. The component primarily models the internal crossbar and input/output queues of the SeaStar. Added flexibility is enabled by parameterizing queue depths, FLIT size and number of FLITs in a packet. The router model has been compared to actual runs on Red Storm and was found to match within 5% for long messages and 12% for short messages.[38]

4.7 QSim

QSim is a front-end for execute-at-fetch microarchitectural simulations that extends and instruments the QEMU[8] processor emulator. QSim adds the ability to arbitrarily control the advancement of execution of the emulated CPU cores and to register callbacks to examine instructions and memory accesses. For this reason, it can be regarded as a re-implementation of Shade [13] for the manycore era. However, unlike Shade, which provides only user mode emulation of the Sparc and MIPS architectures, QSim provides a paravirtualized full-system simulation of 32 and 64 bit x86 CPUs. Because the advancement of execution within QSim's emulated CPUs is controlled by an external timing model, achievement of accurate instruction timing is possible, although some features, like wrong-path execution for misspeculations, remain difficult to implement.

QSim is a library external to the simulation environment, only requiring calls to a timer interrupt function to convey the passage of time to the operating system running within it. While this disables certain CPU features like the Timestamp Counter (TSC) and High-Precision Event Timer (HPET), it simplifies the design of Qsim and increases the freedom of the QSim user.

SST and QSim are combined by a set of simulator-independent components called Slide. Though Slide is still work in progress, the QSim library has been successfully demonstrated with both a simple multi-cycle timing model based on the Intel 386 instruction timings and a cycle-level model of a uniprocessor nonblocking cache hierarchy using SST as the simulation back end.

4.8 SST Stochastic Processor Models

SST currently implements two stochastic processor models that can be used in system simulation. These include an AMD Opteron and a Sun Niagara 2 processor model. The Opteron is presently a single-core model; we are in the process of developing a multi-core Opteron model. We have both a single- and multi-core Niagara 2 model. All of these models are statistical and based on a Monte Carlo technique [6, 34, 35].

The Monte Carlo processor modeling technique is based on the equation, $CPI = CPI_i + CPI_s$, where CPI_i is the ideal or intrinsic CPI based on the instruction issue width and CPI_s is the CPI due to stalls (CPI is cycles per instruction). CPI_i is obtained from processor manuals; the stall causes are determined from both processor manuals and from micro-benchmarks designed to stress a particular processor component. For many processors, general reasons for stalls include cache misses, branch mis-predictions, and issue stalls due to data dependencies.

Processor models comprise most of the major micro-architecture components, including caches, branch predictors, issue queues, and execution units. Parameters to a processor model include characteristics of the micro-architecture and application characteristics. Micro-architecture characteristics primarily consist of component latencies, that are obtained either from processor manuals or from micro-benchmarks. Application characteristics include dynamic instruction mix and statistics on stall causes. These are collected using hardware performance counters and dynamic binary instrumentation tools.

The current versions of these models within SST take the dynamic execution trace, push each instruction through the model, and essentially return the cycle at which each instruction is completed. This information is passed out of the model to any connecting component models. Models can be used as high-level processor components of a larger system simulation or they can be used as stand-alone models for performance prediction and design-space analysis.

5. EXAMPLE MEMORY STUDY

The SST has been used for a number of studies, including network and memory studies, power/thermal modeling studies, application analysis, and network protocol optimization. A few examples are presented below.

For a variety of applications, the main memory subsystem is the dominant factor in on-node performance. Understanding how different applications stress the memory system is an important part of optimizing applications and of designing future memory systems. Using the SST, Sandia has been studying a variety of applications to understand their memory characteristics.

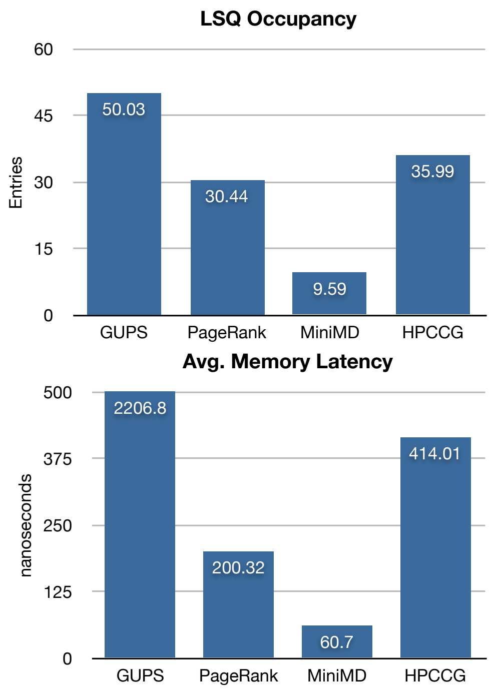

Figure 2: Memory Access Characteristics

Using the `genericProc` and DRAMSim2 components, an 8-core processor connected to a DDR3 memory system was simulated. Several applications were run to examine different memory usage patterns: GUPS (random access)[5]; PageRank from the MTGL[31](graph traversal); Mantevo's[2] MiniMD (molecular dynamics); and Matenvo's HPCCG (sparse matrix-vector multiplication). The effect of the average load-store queue length and memory latency are shown in Figure 2. This simple experiment quickly isolates which applications are most memory intensive (GUPS and HPCCG) and highlights a major performance bottleneck in the memory system when running GUPS. The extraordinarily high latency of memory operations shows an overloaded memory controller and bandwidth limitation, indicating a need for redesign.

Using McPAT and DRAMSim2's internal memory models, it is also possible to determine what the major power consumers are for each application (Figure 3).

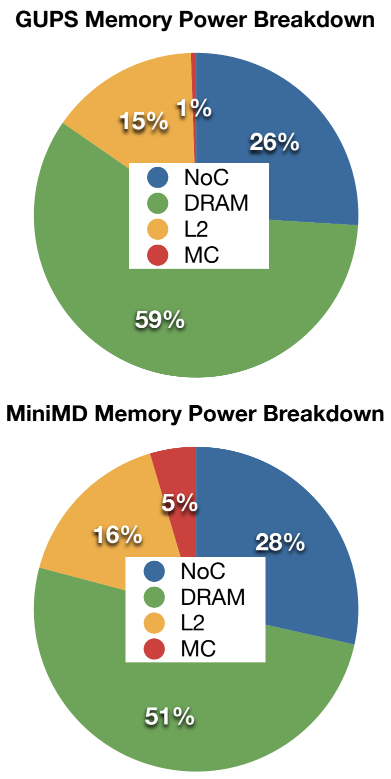

Figure 3: Memory System Power Usage

6. SUMMARY

The SST is an open, modular, parallel, multi-objective, multi-scale simulation framework for HPC architectural exploration. It contains a number of components including processors, memory models, network components and storage models, ranging from very detailed to very abstract. Interfaces to a number of power and thermal models allows multi-objective design space exploration. The SST has been used in a number of architectural studies.

6.1 Future Work

The SST project is continuing to grow in a variety of ways. A few of our current projects include:

- Development of area, cost, and reliability technology models.
- Improvement of the partitioning algorithm to include estimates of component computational and memory requirements.
- More complex storage topologies and RAID [28] configurations, simulation of file system software overheads, and simulation of evolving non-volatile storage architectures such as SSDs and Phase-change memory.
- Integration of stochastic processor models with execution-based front-end(s) and detailed memory modeled. Addition of Monte Carlo processor for the IBM Cell BE, the HP Itanium 2, and the Sun Niagara 1.
- Integration of the MacSim[22] GPU model, Zesto [24, 25] processor model, and M5 node model.

Acknowledgment

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. ADDITIONAL AUTHORS

Additional authors: Brian Barrett (Sandia National Labs, email: bwbarre@sandia.gov) and Bruce Jacob (University of Maryland, email: blj@umd.edu)

8. REFERENCES

[1] *BOOST C++ Libraries.* http://www.boost.org.

[2] *Mantevo Project Home Page.* https://software.sandia.gov/mantevo/.

[3] *The SimpleScalar-Arm Power Modeling Project.* http://www.eecs.umich.edu/ panalyzer/, url = http://www.eecs.umich.edu/ panalyzer/.

[4] Annual energy review 2007. Technical Report DOE/EIA-0384(2007), Department of Energy Energy Information Administration, 2007.

[5] Hpc challenge awards competition. http://www.hpcchallenge.org/, April 2010.

[6] W. Alkohlani, J. Cook, and R. Srinivasan. Extending the Monte Carlo Processor Modeling Technique: Statistical Performance Models of the Niagara 2 Processor. *Proceedings of the IACC International Conference on Parallel Processing (ICPP)*, 2010.

[7] 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.

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

[9] 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. *IEEE Micro*, 26:52–60, 2006.

[10] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *In Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, 2000.

[11] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger. The disksim simulation environment version 4.0 reference manual. Tech Report CMU-PDL-08-101, Carnegie Mellon University, May 2008.

[12] D. Burger and T. Austin. *The SimpleScalar Tool Set, Version 2.0.* SimpleScalar LLC.

[13] B. Cmelik and D. Keppel. Shade: a fast instruction-set simulator for execution profiling. In *SIGMETRICS '94: Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 128–137, New York, NY, USA, 1994. ACM.

[14] O. o. S. Department of Energy. Advanced architectures and critical technologies for exascale computing. Funding Opportunity Number: DE-FOA-0000255, January 2010.

[15] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughn. Zoltan data management services for parallel dynamic applications. *omputing in Science and Engineering*, 4(2):90–97, 2002.

[16] E. M. E. (ed). Systems Resiliance at Extreme Scale. Technical report, DARPA, 2009.

[17] R. Fujimoto. Parallel discrete event simulation. In *Proceedings of the 21st conference on Winter simulation*, pages 19–28, New York, 1989. ACM.

[18] T. Henderson, , T. R. Henderson, and S. Roy. ns-3 project goals.

[19] M. T. Inc. Calculating memory system power for ddr2. Technical Report TN-47-04, 2005.

[20] D. Jensen and A. Rodrigues. Embedded systems and exascale computing. *Computing in Science And Engineering*, 2010. Accepted for Publication.

[21] P. M. e. Kogge. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. Technical report, University of Notre Dame CSE Department Technical Report, TR-2008-13, September 28, 2008.

[22] N. B. Lakshminarayana and H. Kim. Effect of instruction fetch and memory scheduling on gpu performance. In *Workshop on Language, Compiler, and Architecture Support for GPGPU, in conjunction with HPCA/PPoPP 2010*, 2010.

[23] S. Li, J. H. Ahn, R. D. Strong, J. B. B. an d Dean M. Tullsen, and N. P. Jouppi. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Sy mposium on Microarchitecture*, pages 469–480, 2009.

[24] G. Loh, S. Subramaniam, and Y. Xie.

[25] G. H. Loh, S. Subramaniam, and Y. Xie. Zesto: A cycle-level simulator for highly detailed microarchitecture exploration. In *In Proc. of the Int. Symp. on Performance Analysis of Systems and Software*, 2009.

[26] Motorola. Motorola mpc7400 powerpc microprocessors. Technical report, Motorola, 2005.

[27] D. Nellans, V. K. Kadaru, and E. Brunv. Asim- an asynchronous architectural level simulator abstract.

[28] D. A. Patterson, G. A. Gibson, and R. H. Katz.

[29] A. Rodrigues. Gossamer simulator design document. Tech Report T2005-10, University of Notre Dame, Computer Science and Engineering, South Bend, IN, 2005.

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

[31] Sandia National Laboratories. MultiThreaded Graph Library. https://software.sandia.gov/trac/mtgl.

[32] V. e. Sarkar. Exascale Software Study. Technical report, DARPA, September 14, 2009.

[33] J. Schindler and G. Ganger. Automated disk drive characterization. Technical Report CMU-CS-99-176, School of Computer Science, Carnegie Mellon University, 1999.

[34] R. Srinivasan, J. Cook, and O. Lubeck. Performance Modeling Using Monte Carlo Simulation. *IEEE Computer Architecture Letters*, 5(1), 2006.

[35] R. Srinivasan, J. Cook, and O. Lubeck. Ultra-Fast CPU Performance Prediction: Extending the Monte Carlo Approach. *Proceedings of the IEEE International Symposium on Computer Architecture*

and High Performance Computing (SBAC-PAD), 2006.

[36] N. Talagala, R. H. Dusseau, and D. Patterson. Microbenchmark-based extraction of local and global disk characteristics. Technical report CSD-99-1063, University of California at Berkeley, June 2000.

[37] D. T. S. Thoziyoor, D. Tarjan, and S. Thoziyoor. Cacti 4.0. Technical report, 2006.

[38] K. Underwood, M. Levenhagen, and A. Rodrigues. Simulating red storm: Challenges and successes in building a system simulation. In *IEEE International Parallel and Distributed Processing Symposium*, Long Beach, CA, 2007. IEEE.

[39] S. G. R. J. R. W. Huang, K. Skadron and M. R. Stan. Differentiating the roles of ir measurement and simulation for power and temperature-aware design. In *Proceedings of the 2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2009.