A Simulator for Large-Scale Parallel Computer Architectures

Curtis L. Janssen, Sandia National Laboratories, USA
Helgi Adalsteinsson, Sandia National Laboratories, USA
Scott Cranford, Sandia National Laboratories, USA
Joseph P. Kenny, Sandia National Laboratories, USA
Ali Pinar, Sandia National Laboratories, USA
David A. Evensky, Sandia National Laboratories, USA
Jackson Mayo, Sandia National Laboratories, USA

ABSTRACT
Efficient design of hardware and software for large-scale parallel execution requires detailed understanding of the interactions between the application, computer, and network. The authors have developed a macro-scale simulator (SST/macro) that permits the coarse-grained study of distributed-memory applications. In the presented work, applications using the Message Passing Interface (MPI) are simulated; however, the simulator is designed to allow inclusion of other programming models. The simulator is driven from either a trace file or a skeleton application. Trace files can be either a standard format (Open Trace Format) or a more detailed custom format (DUMPI). The simulator architecture is modular, allowing it to easily be extended with additional network models, trace file formats, and more detailed processor models. This paper describes the design of the simulator, provides performance results, and presents studies showing how application performance is affected by machine characteristics.

Keywords: Computer Architecture Simulation, Macro-scale Simulator, Message Passing Interface, Network Congestion, Network Models

INTRODUCTION
The degree of parallelism that must be exposed to efficiently utilize modern large-scale parallel computing systems is intimidating. Because individual processor performance gains are currently achieved primarily through multiple cores on a chip and multiple threads of execution in a core, the rate at which parallelism must be exposed by an application will increase as a function of overall machine performance relative to historical trends. This results in greater design complexity for both machine architects and application software developers. The use
of simulation, however, can aid both in their efforts to obtain high utilization from future computing platforms.

Simulation is already used extensively in the design of computing systems for both functional verification and timing estimation. As an example of the range of capabilities available, including just a few examples of open-source timing simulators, there are processor simulators (Binkert et al., 2006; M5Sim), memory simulators (Jacob; Wang et al., 2005), and network simulators ns-3 (ns-3).

Several simulators have been developed to generate performance estimates for high-performance computing architectures. These range from high-fidelity and computationally expensive simulators for measuring performance between two nodes (Rodrigues et al., 2003; Underwood, Levenhagen, & Rodrigues, 2007) to lower-fidelity and lower-cost simulators that can estimate performance on large-scale machines. These lower-fidelity simulators use a variety of approaches to generate the application’s processor and network workload including tracing, direct execution, and the use of skeleton applications. Additionally, the flow of data through the network is modeled with varying fidelity. In the present paper we are concerned with lower-fidelity and lower-cost simulation techniques to enable simulation at very large scales, and we will briefly discuss these simulator variants in more detail, giving examples of simulators supporting each capability before turning to a detailed description of our simulator in Section 2.

When an application is traced, the full program is run in order to collect information about how it executes. The resulting data is output into a trace file, which contains data such as the time spent in computation and the amount of data sent and received by each node. This trace file is read by the simulator, allowing it to replay the run, adjusting the simulated times to account for differences between the simulated machine and that which was used to collect the traces (Zheng, Wilmarth, Jagadishprasad, & Kale, 2005). In the case of Message Passing Interface (MPI) (Message Passing Interface Forum, 2008) traces, events that are higher level than simple sends and receives are recorded, such as all-to-all broadcast or all-to-one reduce. These network events along with associated parameters are logged without the details of the underlying messages that are used to implement the operation. It is the responsibility of the simulator to either convert these higher-level operations into the low-level messages that implement the operation or to provide an appropriate timing model that does not require simulation of the low-level messages.

In the direct execution approach the full application is run on each node (Prakash et al., 2000; Riesen, 2006; Zheng et al., 2005). This is different from normal benchmarking because, instead of real time, a virtual time is used to determine the execution time. The virtual time is computed by using a network model to estimate communication times. The contribution to the virtual time due to processor execution can be determined simply by using the measured real time for non-communication work or by using a processor model. This model can be informed by measurements of actual application processor utilization or more detailed processor simulations.

The third approach to generating the machine’s workload does not use a full application. Instead, a so-called skeleton application is used that provides enough information to the simulator for it to model both computation and communication. This takes advantage of the fact that the computations needed to determine program flow are a small subset of the total number of computations needed by typical high-performance computing applications. The skeleton application can be constructed in a variety of ways. An application programmer could directly program a skeleton application, giving the programmer the opportunity to experiment with different algorithms before having to write the full application. Existing applications can be skeletonized by replacing portions of the code doing computation with calls that instruct the simulator to account for the time implicitly (Susukita et al., 2008). Skeleton applications can also be constructed using
History and Evolution of GPU Architecture
www.igi-global.com/chapter/history-and-evolution-of-gpu-architecture/139841?camid=4v1a

Measuring the Characteristics of Hypervisor I/O Scheduling in the Cloud for Virtual Machine Performance Interference
Ziye Yang, Haifeng Fang, Yingjun Wu and Chunqi Li (2013). *International Journal of Grid and High Performance Computing* (pp. 5-29).