SAND2011-3179C

SST/macro: The Structural Simulation Toolkit macroscale components for coarse-grained architecture simulation

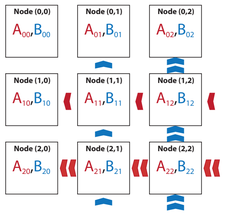
Curtis Janssen, Helgi Adalsteinsson, Scott Cranford, Damian Dechev, David Evensky, Joe Kenny, Nicole Lemaster, Ali Pinar
Sandia National Laboratories, Livermore, CA

Motivation: The changing landscape of high-performance computing architectures requires simultaneous exploration of both the hardware and software design space, a process referred to as co-design. The Structural Simulation Toolkit (SST) enables co-design of extreme-scale architectures by allowing simulation of diverse aspects of hardware and software. The macroscale components of SST aim to simulate full scale machines using a coarse-grained simulation approach. The parallel machine is represented by models which are used to estimate the performance of processing and network components. Applications can be represented by skeletons that replicate the control flow and communication behavior of an actual application without the cost of real message passing or heavyweight computation. The behavior of existing Message Passing Interface (MPI) applications can be captured using the DUMPI library distributed as a part of SST/macro. DUMPI traces can be replayed in SST/macro to simulate a machine different from that used to collect the original trace. SST/macro can be easily be extended with additional network models, trace file formats, and more detailed processor models.

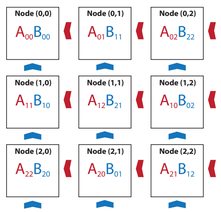
Case Study: Systolic Matrix Multiplication

We base our systolic matrix multiply skeleton application on Cannon's 2D algorithm. Given two $n \times n$ matrices, **A** and **B**, we wish to compute elements $C_{ij} = \sum_k A_{ik} B_{kj}$.

Assigning the nodes to a logical 2D mesh, the **A** and **B** matrices are partitioned into one block per processor as illustrated in the figure to the left. Each **A** block is shifted left by its row number and each **B** block is shifted up by its column number (wrapping around in both dimensions) so that the initial block locations are as in the figure to the right. The algorithm iterates $n_{block} - 1$ times, at each step using non-blocking send/receive calls to simulate shifting its current **A** block one node to the left and its current **B** block one node up. Simulation of the overlapped multiplication of the current blocks with accumulation of the result into the local **C** block is performed. Upon completion of the $n_{block} - 1$ iterations, a compute call simulates the multiplication of the blocks received during the final loop iteration, completing simulation of the algorithm. The SST/macro skeleton code performing this operation is shown below.



GIven layout and necessary shifts for matrix multiplication.

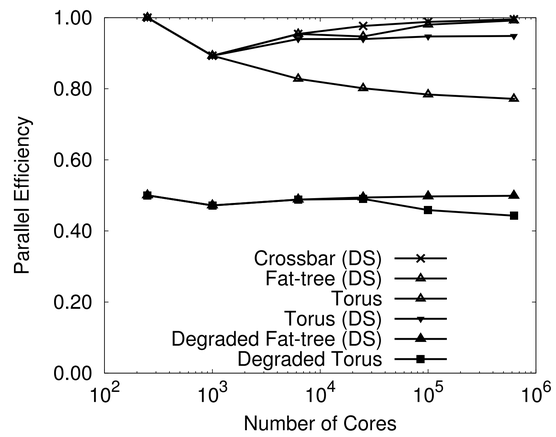


Initial block distribution and block shift pattern.

```
// Set up the instructions object to tell the processor model
// how many fused multiply-add instructions each compute call executes
boost::shared_ptr<sstmac::eventdata> instructions =
  sstmac::eventdata::construct();
instructions->set_event("FMA",blockrowsize*blockcolsize*blocklnksize);
// Iterate over number of remote row and column blocks
for (int i=0; i<nblock-1; i++) {
  std::vector<sstmac::mpiapi::mpirequest_t> reqs;
  // Begin non-blocking left shift of A blocks
  sstmac::mpiapi::mpirequest_t req;
  mpi()->isend(blocksize, sstmac::mpitype::mpi_double,
               sstmac::mpiid(myleft), sstmac::mpitag(0), world, req);
  reqs.push_back(req);
  mpi()->irecv(blocksize, sstmac::mpitype::mpi_double,
               sstmac::mpiid(myright), sstmac::mpitag(0), world, req);
  reqs.push_back(req);
  // Begin non-blocking down shift of B blocks
  sstmac::mpiapi::mpirequest_t req;
  mpi()->isend(blocksize, sstmac::mpitype::mpi_double,
               sstmac::mpiid(myup), sstmac::mpitag(0), world, req);
  reqs.push_back(req);
  mpi()->irecv(blocksize, sstmac::mpitype::mpi_double,
               sstmac::mpiid(mydown), sstmac::mpitag(0), world, req);
  reqs.push_back(req);
  // Simulate computation with current blocks
  compute_api()->compute(instructions);
  std::vector<sstmac::mpiapi::const_mpistatus_t> statuses;
  // Wait for data needed for next iteration
  mpi()->waitall(reqs, statuses);
}
// Simulate computation with blocks received during last loop iteration
compute_api()->compute(instructions);
```

A code fragment implementing a skeleton program for a systolic matrix multiplication (restricted to square blocks).

The MPI-based matrix multiply skeleton application was used to simulate the parallel efficiency of the skeleton algorithm for a hypothetical extreme-scale architecture. The machine parameters used are 10 GByte/sec internode bandwidth, 1 µs internode latency, 10 GByte/s intranode bandwidth, 10 ns intranode latency, a 4 Gop/sec fused multiply and add rate for each core, and 250 cores per MPI rank. The skeleton application is organized so that a single MPI rank runs on each node, and multi-threading is used for parallelism on the nodes. Since we focus on the communications-related performance and programmability, we do not explicitly model each thread on each node. Each node starts with a single matrix block of size 10,000 on each side. The degraded runs include a single node running at half of the computation rate as the other nodes. For the fat-tree and crossbar networks, the systolic shifts in the initial redistribution phase are replaced by direct sends to the target nodes. We will refer to this algorithm as DS.



Weak scaling parallel efficiency of the systolic matrix matrix algorithm: 1) full crossbar network (DS), 2) fat-tree with radix 36 switches (DS), 3) 2D torus network, 4) 2D torus network (DS),5) fat-tree with radix 36 switches with a single degraded node (DS), 6) 2D torus with a single degraded node. DS designates use of the direct send algorithm.

Results are shown in the figure to the right. Moving from a single node to multiple nodes, we see a dip in performance due to the necessity of moving data through the network. As the number of nodes increases, the crossbar and fat-tree outperform the torus because they do not systolically form the initial data distribution. The fat-tree is at times slightly slower than the crossbar, because it uses static routes and some congestion is possible. For the degraded cases, performance is roughly halved because the systolic nature of the algorithm forces all nodes to wait for the degraded node at each synchronization point.

For comparison, the torus results were also obtained using the direct send (DS) algorithm, and these are also shown in the figure. For the machine and problem parameters in our study, DS was faster than using a systolic startup algorithm. The torus is still slower than the fat-tree and crossbar networks due to network congestion in forming the initial block distribution.

How can the CSC Community Benefit from SST/Macro?

- Algorithms and models for load balancing, task and communication scheduling, mapping of tasks to processors, design of interconnects and associated communication algorithms has been at the heart of combinatorial computing.
- Evaluating the effectiveness of these algorithms however, has been troublesome, since we design algorithms for future machines, and timings of existing machines vary widely since external factors cannot be controlled.
- We measure our success in our terms (such as volume of communication, number of communicating pairs, maximum load among all processors, etc.), which may limit adoption of our results by the broader scientific computing community.
- SST/macro offers a platform to evaluate our proposed techniques:
  - on expected future platforms
  - using our own computational resources
  - in a controlled environment, where we can observe the effects of our proposed improvements without the burden of external factors.
- SST/macro can help us answer questions such as:
  - What is the best way to model communication load of a system? What are the trade-offs among communication volume, number of messages, and distribution of communication load among processors, and others?
  - How should we tailor our load balancing algorithms based on architectural trends?
  - What will be the new performance bottlenecks in the future architectures and how can CSC help overcome these bottlenecks?

Code Availability

SST is publicly available. For further information, visit
http://sst.sandia.gov/using_sstmacro.html

Contact Information

Further information is available at http://sst.sandia.gov/
You can also contact
Project leader: Curtis Janssen (cljanss@sandia.gov)
Poster presenter: Ali Pinar (apinar@sandia.gov)

Funding Statement

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.