Using BigSim to Estimate Application Performance

Ryan Mokos
Parallel Programming Laboratory
University of Illinois at Urbana-Champaign
Outline

• Overview
• BigSim Emulator
• BigSim Simulator
BigSim

- Built on Charm++
- Object-based processor virtualization
  - Virtualized execution environment that allows running large-scale simulations on small-scale systems
- Runs Charm++ and AMPI applications at large scale
Performance Prediction

- Two components:
  - Time to execute blocks of sequential, computation code
    - SEBs = Sequential Execution Blocks
  - Communication time based on a particular network topology
BigSim Components

- Emulator
  - Generates traces that capture SEB execution times, dependencies, and messages
- Simulator (BigNetSim)
  - Trace-driven
  - PDES (Parallel Discrete Event Simulation)
  - Calculates message timing based on the specified network model
Using BigSim to Estimate Application Performance

**BigSim Architecture**

- **Performance visualization (Projections), Link utilization stats, Terminal output**
- **BigSim Simulator**
- **Simulation trace logs**
- **AMPI Runtime**
- **Charm++ Runtime**
- **BigSim Emulator**
- **Charm++/AMPI applications**
Limitations

- BigSim does not:
  - Include cycle-accurate/instruction-level simulation
    - But can be integrated with external simulators
  - Predict cache and virtual memory effects
  - Model interference
    - Operating system
    - External job
  - Model non-deterministic applications
Outline

• Overview
• BigSim Emulator
• BigSim Simulator
**Emulator**

- Implemented on Charm++
  - Libraries link to user application
- Virtualized execution environment
  - Each physical processor emulates multiple target processors
  - Be careful of increased memory footprint
- Efficiencies realized
  - NAMD: virt. ratio 128 ⇒ 7x memory, 19x run time
Using the Emulator (64-Bit Linux Example)

- Convert MPI application to AMPI (or use a Charm++ application)
- Install emulator
  - Download Charm++
    - http://charm.cs.uiuc.edu/download/
  - Compile Charm++/AMPI with “bigemulator” option
    - `./build AMPI net-linux-x86_64 bigemulator -j8 -O`
    - This builds Charm++ and emulator libraries under net-linux-x86_64-bigemulator (work in this directory)
Using the Emulator (continued)

• Set parameters in a config file
  • Note: the same topology (x, y, z dimensions and number of worker threads) will be used by the simulator
    • \( \text{wth} = \# \text{ worker threads} = \# \text{ cores} / \text{ node} \)
• Compile the application to be emulated in the \(<\text{net-layer}>\text{-bigemulator}\) directory
• Run the application with the config file via \(+\text{bgconfig} \ <\text{config file}>\)
Example - AMPI Cjacobi3D

- `cd charm/net-linux-x86_64-bigemulator/examples/ami/Cjacobi3D`
- `make`
- Modify config file `bg_config` as desired:

  ```
  x  4
  y  2
  z  2
  Cth 1
  wth 8
  stacksize  10000
  timing walltime
  #timing bgelapse
  #timing counter
  cpufactor 1.0
  fpfactor 5e-7
  traceroot .
  log  yes
  correct no
  network  bluegene
  #projections 2,4-8
  ```
Example - AMPI Cjacobi3D (continued)

- Run emulation of 8 target processors (virtual processors) on 2 physical processors
  - ./charmrun +p2 ./jacobi 2 2 2 +vp8 +bgconfig bg_config

- As long as “log yes” is specified in the config file, 3 trace files will be written:
  - bgTrace – summary file
  - bgTrace0 – trace file for the 4 vps on processor 0
  - bgTrace1 – trace file for the 4 vps on processor 1
LogAnalyzer

• Tool for analyzing emulator traces
• Run as `./LogAnalyzer –i` (interactive) or `./LogAnalyzer –c <choice #>` (good for scripting)
• Options:
  • Display time line lengths
  • Convert traces to ASCII files
  • Display the number of messages sent and received by each target processor
  • Display the total execution time of all events on each target processor
  • Display the number of packets sent by each target processor
  • Execution time estimation (experimental)
Skip Points

• Add to actual application code at places where control is completely given back to processor 0 (e.g., after allreduce, barrier, load balancing, etc.)
  • $BgSetStartEvent()$
• Skip points marked in trace files
• Simulator can execute between skip points
• Uses:
  • Bypass start-up sequence
  • Simulate only one application step
Projections

• Visual tool for analyzing program runs
• Link the emulated application with \textit{tracemode projections} to get projections traces
• Can be modified by the simulator
Projections Example - MPI AlltoAll Timeline

Using BigSim to Estimate Application Performance
Emulator - Other Features

- Different levels of fidelity available for predicting performance
  - Wallclock time with cpu scaling factor (already discussed)
  - Manually elapse time with BgElapse() calls
  - Performance counters
  - Instruction-level/cycle-accurate simulation
  - Model-based (time most-used functions and interpolate to create model)
- Out-of-core execution when emulation won’t fit in main memory
- Record/replay subset of traces
Outline

• Overview
• BigSim Emulator
• BigSim Simulator
**Simulator (BigNetSim)**

- PDES simulator built on top of Charm++
- Run BigNetSim on emulator traces to get final run results for a particular network model
- Pre-compiled binaries supplied for this workshop
- To download and build source code from public repository (does not contain Blue Waters model), see the 2009 Charm++ Workshop BigSim tutorial
Running BigNetSim on Blue Print

• Ensure bgTrace files, charmrun, and charmrun.ll are in the same directory as the executable.

• Update charmrun.ll with desired *output* and *error* file names.

• Submit job to loadleveler
  
  • ./charmrun +p <# procs> +n <# nodes> ./<executable> <BigNetSim arguments>
  
  • E.g.: ./charmrun +p 1 +n 1 ./bigsimulator -linkstats -check
  
  • Note: there must be a space between +p and +n and their numbers.
  
  • Note: the +p parameter specifies the total number of processors.
  
  • E.g., running on all 16 procs of each of 4 nodes (64 procs total) would be +p 64 +n 4
Simple Latency Model vs. Blue Waters Model

• Simple Latency includes processors and nodes and implements the network with an equation:
  • \( \text{lat} + \frac{N}{\text{bw}} + [\text{cpp} \times \frac{N}{\text{psize}}] \)
    • \( \text{lat} \) = latency in \( \mu \text{s} \)
    • \( \text{bw} \) = bandwidth in GB/sec
    • \( \text{cpp} \) = cost per packet in \( \mu \text{s} \)
    • \( \text{psize} \) = packet size in bytes
    • \( N \) = number of bytes sent

• Blue Waters includes processors, nodes, Torrents, and links between the Torrents
Command-Line Arguments - Simple Latency

- Bandwidth and latency must be specified:
  - `-bw <double>` Link bandwidth in GB/s
  - `-lat <double>` Link latency in µs

- Other optional arguments:
  - `-help` Displays all available arguments
  - `-cpp <double>` Cost per packet in µs
  - `-psize <int>` Packet size in bytes
  - `-bw_in <double>` Intra-node bandwidth in GB/s
    Defaults to `-bw` value if not specified
  - `-lat_in <double>` Intra-node latency in µs
    Defaults to 0.5µs if not specified
Command-Line Arguments - Simple Latency (continued)

- **-check** Checks for unexecuted events at the end of the simulation
- **-cpufactor <double>** A constant by which SEB execution times are multiplied; defaults to 1.0
- **-debuglevel <0|1>** 0: no debug statements
  1: high-level debug statements and summary info
- **-projname <string>** Sets the name of the projections logs that will be corrected based on network simulation
- **-skip_start <int>** Sets the skip point at which simulation execution begins
- **-skip_end <int>** Sets the skip point at which simulation execution ends
- **-tproj** Generate projections logs based only on network simulation
Command-Line Arguments - Blue Waters

• No arguments are required; all are optional:
  • -help Displays all available arguments
  • -check Checks for unexecuted events at the end of the simulation
  • -cpufactor <double> A constant by which SEB execution times are multiplied; defaults to 1.0
  • -debuglevel <0|1> 0: no debug statements
                        1: high-level debug statements and summary info
  • -linkstats Enable link stats for display at the end of the simulation
  • -projname <string> Sets the name of the projections logs that will be corrected based on network simulation
  • -skip_start <int> Sets the skip point at which simulation execution begins
  • -skip_end <int> Sets the skip point at which simulation execution ends
  • -tproj Generate projections logs based only on network simulation
Command-Line Arguments - Blue Waters (continued)

- -tracelinkstats  Enable tracing of link stats
- -tracecontention  Enable tracing of contention
**BigNetSim Output - Terminal (Text)**

- BgPrintf(char *) statements
  - Added to actual application code
  - “%f” in function call argument converted to committed time during simulation
- GVT = Global Virtual Time
  - Final simulation virtual time expressed in GVT ticks
  - 1 GVT tick = 1 ns for the provided binaries
- Link utilization statistics
BigNetSim Output - Terminal (Text) - Example

Charm++: standalone mode (not using charmrun)
Charm++> Running on 1 unique compute nodes (8-way SMP).
================= Simulation Configuration =================
Production version: 1.0 (10/13/2010)
Simulation start time: Fri Oct 15 13:11:09 2010
Number of physical PEs: 1
POSE mode: Sequential
Network model: Blue Waters
...
==================================================================
Construction phase complete
Initialization phase complete
Info> invoking startup task from proc 0 ...
Info> Starting at the beginning of the simulation
Info> Running to the end of the simulation
Entire first pass sequence took about 18.532318 seconds
[0:user_code] #MILC# - WHILE Loop Iterarion Starting at 0.509469
[0:user_code] #MILC# - LL-Fat Starting at 0.510801
...
Sequential Endtime Approximation: 906988512
Final link stats [Node 0, Channel 0, LL Link]: ovt: 906953211, utilization time: 257562, utilization %: 0.028397, packets sent: 2290 gvt=906988512
Final link stats [Node 0, Channel 11, LR Link]: ovt: 906953211, utilization time: 631426, utilization %: 0.069618, packets sent: 1827 gvt=906988512
1 PE Simulation finished at 74.104628.
Program finished.
BigNetSim Output - Projections

• Copy emulation Projections logs and sts file into directory with executable
  • Two ways to use:
    • Command-line parameter: -projname <name>
      • Creates a new set of logs by updating the emulation logs
      • Assumes emulation Projections logs are: <name>.*.log
      • Output: <name>-bg.*.log
      • Disadvantage: emulation Projections overhead included
    • Command-line parameter: -tproj
      • Creates a new set of logs from the trace files, ignoring the emulation logs
      • Must first copy <name>.sts file to tproj.sts
      • Output: tproj.*.log
      • Advantage: no emulation Projections overhead included
Using BigSim to Estimate Application Performance

Projections - Ring Example

Emulation

Simulation: -lat 1 (latency = 1µs) generated with -tproj
BigNetSim Output - Link Stat and Contention Traces (experimental)

- Enabled on the command line at run time
- Placed in a unique folder named `link_traces_<simulation start time>`
- May significantly increase run time and memory footprint
- LinkStatTraceAnalyzer tool examines link stat traces for links with high utilization and contention
  - Writes reports listing links in order from most to least utilized
BigNetSim Validation

- Network traffic generator tests of the BigNetSim Blue Waters network model give simulation results within a couple percent of those of IBM’s hardware simulator.
## BigNetSim Performance

- Examples of sequential simulator performance on Blue Print

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Memory Footprint Estimate (GB)</th>
<th>Startup Time (hours)</th>
<th>Execution Time (hours)</th>
<th>Total Run Time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sim Lat</td>
<td>BW</td>
<td>Sim Lat</td>
<td>BW</td>
</tr>
<tr>
<td>4k-VP MILC</td>
<td>2.3</td>
<td>2.6</td>
<td>0.72</td>
<td>0.73</td>
</tr>
<tr>
<td>256k-VP 3D Jabobi (10x10x10 grid, 3 iters)</td>
<td>17.5</td>
<td>18.3</td>
<td>0.51</td>
<td>0.51</td>
</tr>
<tr>
<td>256k-VP NAMD (1M atoms, 8 iters, skip startup)</td>
<td>14.9</td>
<td>15.9</td>
<td>0.49</td>
<td>0.47</td>
</tr>
</tbody>
</table>

- Parallel performance is comparable to sequential but does not scale well yet outside a single node on Blue Print
BigNetSim - Other Features

- Other network models (e.g., BlueGene)
- Transceiver (traffic pattern generator) for testing network models without traces
- Checkpoint-to-disk to allow restart if hardware on which BigNetSim is running goes down
- Load balancing
Additional Resources

• BigSim manuals:
  • http://charm.cs.uiuc.edu/manuals/

• Recent Charm++ Workshop tutorials and talks
  • 2008 BigSim tutorial (bottom of page)
    • http://charm.cs.illinois.edu/workshops/charmWorkshop2008/slides.html
  • 2009 BigSim tutorial (bottom of page)
    • http://charm.cs.uiuc.edu/workshops/charmWorkshop2009/program.html
  • 2010 BigSim talk (near top of page)
    • http://charm.cs.uiuc.edu/charmWorkshop/program.php

• E-mail PPL for help: ppl@cs.uiuc.edu