Tools for Performance Modeling and Optimization of Parallel Applications on Future Supercomputers

Celso L. Mendes & Sanjay Kale
http://charm.cs.uiuc.edu
Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana-Champaign
Presentation Outline

• Introduction
  – HPC Landscape
  – PPL: mission and approach, programming methodology

• BigSim: Simulation of Large Systems
  – BigSim organization
  – BigSim application emulation
  – BigSim simulation of target system

• Scalable Performance Analysis
  – Performance data volume
  – Selective data reduction
Introduction

• **Current HPC Landscape**
  – Petascale era started; exascale not too far (DOE meetings 2007)
  – **Roadrunner@LANL (#1 in Top500):**
    • Linpack: 1.026 Pflops, Peak: 1.375 Pflops
  – Heterogeneous systems starting to spread (Cell, GPUs, …)
  – Processor counts in Top-500:
    • #1 **Roadrunner@LANL**: 122K
    • #2 **BG/L@LLNL**: 212K
    • #3 **BG/P@ANL**: 163K
  – Clear need for scalable tools
Parallel Programming Lab
Parallel Programming Lab - PPL

• http://charm.cs.uiuc.edu

• One of the largest research groups at Illinois

• Currently:
  – 1 faculty, 2 research scientists, 4 research programmers
  – 12 grad students, 1 undergrad student
  – Open positions 😊
PPL Mission and Approach

• To enhance *Performance and Productivity* in programming *complex* parallel applications
  – *Performance*: scalable to thousands of processors
  – *Productivity*: of human programmers
  – *Complex*: irregular structure, dynamic variations

• Application-oriented yet CS-centered research
  – Develop enabling technology, for a wide collection of apps.
  – Embody it into easy to use abstractions
  – Implementation: Charm++
    • Object-oriented runtime infrastructure
    • Freely available for non-commercial use (see BOF@SC08)
Synergy between Computer Science research and applications has been beneficial to both

- LeanCP
- NAMD
- ChaNGa
- Charm++
- Space-time meshing
- Rocket Simulation

Other Applications

Techniques & libraries

Issues
Methodology: Migratable Objects

Programmer: [Over] decomposition into objects (“virtual processes” - VPs)

Runtime: Assigns VPs to real processors dynamically, during execution

Enables *adaptive runtime strategies*

Implementations: Charm++, AMPI

---

**Benefits of Virtualization**

- **Software engineering**
  - Number of virtual processes can be independently controlled
  - Separate VP sets for different modules in an application

- **Message driven execution**
  - Adaptive overlap of computation/communication

- **Dynamic mapping**
  - Heterogeneous clusters
    - Vacate, adjust to speed, share
  - Automatic checkpointing
  - Change set of processors used
  - Automatic dynamic load balancing
  - Communication optimization
Adaptive MPI (AMPI): MPI + Virtualization

• Each virtual process implemented as a user-level thread embedded in a Charm++ object
  – Must properly handle globals and statics (analogous to what’s needed in OpenMP)
  – But… thread context-switch is much faster than other techniques
BigSim Simulation System
Performance Tuning for Future Machines

• For example, Blue Waters will arrive in 2011
  – But we need to prepare applications for it, starting now

• Even for existing machines:
  – Full size machine may not be available as often as needed for tuning runs

• BigSim: a simulation-based approach
  – Based on Charm++ virtualization technique
  – Full scale \{application+system\} simulation
  – History: developed for BlueGene/C
    • NSF/NGS grant 2001-2006
BigSim Simulation System

• Major BigSim features:
  – Emphasis on bottleneck identification, not prediction accuracy
  – Multi-resolution modeling for computation and communication
  – Detailed network simulation, driven by application code

• General BigSim organization
BigSim Simulation System

• Typical BigSim usage:
  – Phase 1: Program emulation – obtain \textit{execution logs}
  – Phase 2: Trace-driven simulation(s) – obtain predictions
  • Can be repeated for different target machine configurations
  – Phase 3: Performance analysis with Projections

• Envisioned use scenarios:
**BigSim – Phase 1: Emulation**

- **BigSim Emulation:**
  - Run an existing, full-scale Charm++ or MPI application
  - Uses an emulation layer in Charm++ that pretends to be the target machine
    - Target cores are emulated as Charm+ virtual processes
  - Resulting execution logs:
    - Recording of computation and communication activity
    - Information stored:
      - Characteristics of SEBs (Sequential Execution Blocks)
      - Dependences between execution blocks and messages
BigSim – Emulation Challenges

• **Memory issues:**
  – Applications with large memory footprints on target processor may require a lot of memory space during emulation
  – Virtual Memory (VM) might handle it, but not efficiently
    • VM has no knowledge about underlying pieces

• **BigSim approaches:**
  – Implicit memory emulation component
    • Reuse (read-only) data across target processors
    • May require application modification
  – Out-of-Core emulation component
    • Idea: keep in physical memory only a subset of target processors; bring new target processors on demand
BigSim – Emulation Challenges (cont.)

- Out-of-Core Emulation Scheme:

- Because the emulation is run with Charm++, the Charm++ scheduler knows which piece(s) will be required next and which pieces should be evicted from physical memory.
BigSim – Emulation Challenges (cont.)

• Out-of-Core emulation status:
  – Prototype version implemented, undergoing evaluation and integration to BigSim distribution
  – Number of resident target processors as large as allowed by existing physical memory
  – Slowdown in observed emulation performance is acceptable

• Planned Out-of-Core optimizations:
  – Prefetch of target processors from disk
  – Smarter eviction policies (currently only LRU)
BigSim – Phase 2: Simulation

• Trace-driven parallel simulation
  – Parallel discrete event simulator
    • Implemented as a Charm++ code
  – Multiple resolution simulation of sequential execution:
    • simple scaling factor between existing/target processors
    • scaling based on performance counter data
      – Perfex and PAPI supported
    • modeling based on cycle-accurate simulators
      – e.g. IBM’s Mambo
BigSim – Phase 2: Simulation

- **Workflow with cycle-accurate simulator:**

```c
void func(p1,… )
{
    StartBigSim(p1,… )
    ...
    EndBigSim()
}
```

- **Mambo**
  - Cycle-accurate prediction of sequential blocks on POWER7 processor

- **BigSim Parallel Emulation**
  - Log files
  - Parameter files for sequential blocks

- **Interpolation**
  - Replace sequential timing

- **Prediction for Target System**
  - Adjusted log files
BigSim – Phase 2: Simulation

- **Interconnection Network Modeling**
  - Multiple resolution simulation of the Network:
    - simple latency/bandwidth model
    - detailed packet and switching port level modeling
  - Flexible selection from a variety of:
    - Topologies (Mesh, Torus, FatTree, …)
    - Routing algorithms (static or adaptive)
    - Input/Output virtual channel selection algorithms
  - Implementation approach:
    - Network layer constructs (NIC, switch, node, etc): objects
    - Network data constructs (message, packet, etc): event methods
  - Network characterization:
    - Source code, plus runtime configuration files
BigSim – Phase 2: Simulation

- Network Simulation Output (NAMD-apoa1, 15 steps)
  - Link Utilization

![Graph showing link utilization over time]

10/21/2008
BigSim Validation: BG/L Predictions

Simulations run on 8 processors, simple network model:

![Diagram showing actual and predicted execution times for NAMD Apoa1 simulations with varying numbers of processors.](chart.png)

- **Actual execution time**
- **Predicted time**

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Actual Execution Time</th>
<th>Predicted Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>80 seconds</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>60 seconds</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>40 seconds</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>20 seconds</td>
<td></td>
</tr>
<tr>
<td>2250</td>
<td>10 seconds</td>
<td></td>
</tr>
</tbody>
</table>
BigSim – Simulation Challenges

• Improved Scalability
  – More efficient handling of log files
  – Load balancing of simulation
  – Smarter memory management and optimized communication

• New enhancements planned
  – New strategies and algorithms for networks
  – Ports to BG/P and Cray-XT/4 for running simulation
  – Network fault simulation
BigSim – Phase 3: Analysis

- Phase 3: Analyze performance
  - BigSim can produce Projections-compatible data
  - Simulation data can be analyzed by the same tools used for data obtained on real machines (e.g. Projections)
Scalable Performance Analysis
Scalable Performance Analysis

• Scaling and performance data volumes
  – Both weak scaling and strong scaling lead to performance data volume growth
  – Example: Performance data from NAMD on 200 iterations

<table>
<thead>
<tr>
<th></th>
<th>92k Atoms</th>
<th>327k Atoms</th>
<th>1000k Atoms</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 cores</td>
<td>827 MB</td>
<td>1,800 MB</td>
<td>2,800 MB</td>
</tr>
<tr>
<td>1024 cores</td>
<td>938 MB</td>
<td>2,200 MB</td>
<td>3,900 MB</td>
</tr>
<tr>
<td>2048 cores</td>
<td>1,200 MB</td>
<td>2,800 MB</td>
<td>4,800 MB</td>
</tr>
<tr>
<td>4096 cores</td>
<td></td>
<td></td>
<td>5,700 MB</td>
</tr>
</tbody>
</table>

Strong Scaling
  – Obvious conclusion: tools must also scale to deal with this growth

Weak Scaling
Scalable Performance Analysis

- Our Approach:
  - Retain full traces of a reduced number of processors
  - Keep summaries for remaining processors
  - Select “interesting” processors based on observed data
  - Criteria for processor selection:
    - Cluster the observed data into equivalence classes, according to some metric(s)
    - Pick “representatives” and “outliers” from each class based on a certain threshold
    - Initial implementation: k-means clustering
  - Details in HIPS paper @ IPDPS’08
Scalable Performance Analysis

- Schematic representation:
Scalable Performance Analysis

Several factors to consider:

- Which metrics to use
- Metrics may require normalization
- Whether there is correlation between metrics
- Number of clusters
- Placement of initial seeds
- Number of representatives chosen
- Number of outliers chosen
Scalable Performance Analysis

• Preliminary Evaluation:
  – NAMD executions at PSC’s Cray XT-3, 1 million atoms
  – Roll-back to 2002 code version that had a grain-size problem
  – Histograms of NAMD method durations in Projections:

    Tuned NAMD

    Problem Injected

  – Experiment goal: compare histograms obtained with full data and with reduced-processor data, for the 2002 code execution
Scalable Performance Analysis

Data reduction after keeping data from 10% of processors:

![Graph showing data volume vs. processor cores]

- **Original Dataset**
- **Reduced Dataset**

Data Volume (megabytes) vs. Processor Cores
Scalable Performance Analysis

- **Quality of data reduction:**

Original Data: 1000 procs

Reduced Data: 100 procs

How close is $H_r^i/H_o^i$ to 0.100 on average?
Scalable Performance Analysis

- Resulting quality of data (for 5%, 10%, 20%)

<table>
<thead>
<tr>
<th>$P_o$</th>
<th>$P_r$</th>
<th>$P_r/P_o$</th>
<th>Average H</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>25</td>
<td>0.0488</td>
<td>0.0641</td>
<td>0.00732</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td>0.0996</td>
<td>0.1180</td>
<td>0.00768</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>0.1992</td>
<td>0.2237</td>
<td>0.00732</td>
</tr>
<tr>
<td>1024</td>
<td>51</td>
<td>0.0498</td>
<td>0.0511</td>
<td>0.00168</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>0.0996</td>
<td>0.1008</td>
<td>0.00157</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>0.1992</td>
<td>0.1921</td>
<td>0.00264</td>
</tr>
<tr>
<td>2048</td>
<td>102</td>
<td>0.0498</td>
<td>0.0487</td>
<td>0.00122</td>
</tr>
<tr>
<td></td>
<td>204</td>
<td>0.0996</td>
<td>0.0977</td>
<td>0.00216</td>
</tr>
<tr>
<td></td>
<td>408</td>
<td>0.1992</td>
<td>0.1883</td>
<td>0.00575</td>
</tr>
<tr>
<td>4096</td>
<td>204</td>
<td>0.0498</td>
<td>0.0501</td>
<td>0.00170</td>
</tr>
<tr>
<td></td>
<td>409</td>
<td>0.0998</td>
<td>0.0981</td>
<td>0.00203</td>
</tr>
<tr>
<td></td>
<td>818</td>
<td>0.1997</td>
<td>0.1975</td>
<td>0.00163</td>
</tr>
</tbody>
</table>
Scalable Performance Analysis

• Ongoing work:
  – Analyzing effects from each factor
  – Looking at other clustering alternatives
  – Extending tests to other codes
  – Applying same scheme to MPI codes

• Current integration to Projections
  – Basic features:
    • Processors with max/min metric values
    • Averages across processors for various metrics
    • K% processors most distant from average
Summary

• Our scalable tools:
  – BigSim: simulation of large systems (sequential & network)
  – Projections: performance data handling
  – Both leverage Charm++ and its virtualization approach
  – Software distribution: http://charm.cs.uiuc.edu

• Our sponsors:
  – NSF, Dep. Energy, NIH, NCSA/NSF, Nasa
Thank You!