Performance Modeling for Systematic Performance Tuning

Torsten Hoefler, William Gropp, Marc Snir, Bill Kramer

Paper Presentation at Supercomputing 2011 (SotP)
November 15th 2011

All used images belong to the owner/creator!
Special Announcement!

• Blue Waters is now officially back!

• … but back to the talk (examples are still POWER7)

Details: http://www.ncsa.illinois.edu/BlueWaters/system.html
The Perspective of a Computing Center

• Performance = “completed science per cost and time”
• Optimizing this metric can be manifold:
  • Application optimization (support application teams)
  • Architecture optimization (select best hardware)
  • Optimize Middleware (scheduler, libraries etc.)
  • Optimize Policies (scheduling, charging etc.)
  • … and many more
Performance Modeling – State of the Practice

- Delivers the “science per cost/time” metric
  - Can be used to drive optimizations!
- Who does performance modeling?
  - Mostly computer scientists, in-house teams
- BUT: most development is done by application developers and/or domain scientists
  - They should develop performance models during software development
    - See performance modeling panel @3:30 in TCC 101
(Ideal) State of the Practice @NCSA

• Propose to use simple performance modeling to characterize the behavior of applications
  • Enables rough optimization (cf. “80/20 rule”)
• We provide a set of simple modeling guidelines
  • Semi-analytic performance modeling
  • Small number of parameters, use other techniques where necessary

Benchmark  ---  Full Simulation  ---  Model Simulation  ---  Model
Overview of Performance Modeling

• Analytic modeling:
  • Determine application requirements and system speeds to compute time (e.g., bandwidth)

• Empirical modeling (e.g. [1,2]):
  • “Black-box” approach: machine learning, neural networks, statistical learning …

• Semi-empirical modeling:
  • “White box” approach: find asymptotically tight analytic models, parameterize empirically (curve fitting)

[1]: Barnes, Rountree, Lowenthal, Reeves, Supinski, Schulz: A regression-based approach to scalability prediction
[2]: McKee, Singh, Supinski, Schulz: Constructing Application Performance Models Using Neural Networks
A Quick Example - MM

- Matrix multiplication ($N^3$ algorithm)

```c
for(int i=0; i<N; ++i)
    for(int j=0; j<N; ++j)
        for(int k=0; k<N; ++k)
            C[i+j*N] += A[i+k*N] * B[k+j*N];
```

- Trivial (non-blocked) algorithm

- Analytic Model:
  - $N^3$ FP add/mult, $4N^3$ FP load/store, +int ops
  - How can we get to an execution time? → very hard!
Semi-Empiric Model for MM

- \( T(N) = tN^3 \)
- POWER7
  - \( t=2.2\text{ns} \)
  - 0.8% err
- Is that all?
  - Requirement Model delivers more insight!
Requirements Model for MM

- Required floating point operations: $2N^3$ (verified)

- Cache misses?
  - Semi-analytic!
  - $C(N) = aN^3 - bN^2$

- POWER7
  - $a=3.8e^{-4}$
  - $a=2.7e^{-1}$
Our Ubiquitous Modeling Philosophy

- Modeling during each phase of SW development:
  - Analysis – pick right method (asymptotic models)
  - Design – pick right algorithms (asymptotic models)
  - Implementation – show good usage of machine, e.g., blocking in MM (semi-empirical models)
  - Testing – fulfilling model expectations as correctness criterion (compare tests with models)
  - Maintenance – monitor performance on different architectures (compare times with models)
More uses of Models

• Performance Optimization
  • Identify bottlenecks and problems during porting
• System Design
  • Co-design based on application requirements
• System Deployment and Testing
  • Know what to expect, find performance issues quickly
• During System Operation
  • Detect silent (and slow) performance degradation
Six-Steps to a Model

• Our very high-level strategy consists of the following six steps:

1) Identify input parameters that influence runtime
2) Identify application kernels
3) Determine communication pattern
4) Determine communication/computation overlap
5) Determine sequential baseline
6) Determine communication parameters
All Steps By Example – MILC

- MIMD Lattice Computation
  - Gains deeper insights in fundamental laws of physics
  - Determine the predictions of lattice field theories (QCD & Beyond Standard Model)
- Major NSF application
- Challenge:
  - High accuracy (computationally intensive) required for comparison with results from experimental programs in high energy & nuclear physics

Bernard, Gottlieb et al.: Studying Quarks and Gluons On Mimd Parallel Computers
Step 1: Critical Parameters

- **Best way:** ask a domain expert!
  - Or: look through the code/input file format
- **For MILC (thanks to S. Gottlieb):**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>number of PEs (intrinsic parameter)</td>
</tr>
<tr>
<td>nx, ny, nz, nt</td>
<td>size in x, y, z, t dimension</td>
</tr>
<tr>
<td>warms, trajecs</td>
<td>warmup rounds and trajectories (outer loop)</td>
</tr>
<tr>
<td>traj_between_meas</td>
<td>measurement “frequency”</td>
</tr>
<tr>
<td>steps_per_trajectory</td>
<td>number of “steps” in each trajectory</td>
</tr>
<tr>
<td>beta, mass1, ...</td>
<td>physics parameters that influence CG iterations</td>
</tr>
<tr>
<td>max_cg_iterations</td>
<td>limits the conjugate gradient iterations</td>
</tr>
</tbody>
</table>
Step 2: Find Kernels

• E.g., investigate call-tree or source-code
• Control logic
  • update
• MILC’s kernels:
  • LL (load_longlinks)
  • FL (load_fatlinks)
  • CG (ks_congrad)
  • GF (imp_gauge_force)
  • FF (eo_fermion_force_twoterms)
Step 4: Sequential Performance

- MILC “only” loops over the lattice $\Theta(V)$
  - $T(V) = tV$
    - Wait, it’s not that simple with caches 😞
    - Small $V$ fit in cache!
  - $T(V) = t_1 \times \min(s, V) + t_2 \times \max(0, V-s)$
    - Cache holds $s$ data elements
    - Three parameters for each kernel
An Example Kernel: GF (Gauge Force)

- On POWER7:
  - $t_1 = 62.4 \mu s$
  - $t_2 = 92 \mu s$
  - $s = 4.000$

- Errors
  - Max $< 10\%$
  - Cum $< 3\%$
Complete Serial Performance Model

\[ T_{\text{serial}}(V) = \left( \text{trajecs} + \text{warms} \right) \cdot \text{steps} \cdot \left[ T(FF, V) + T(GF, V) + 3(T(LL, V) + T(FL, V)) \right] + \left\lfloor \frac{\text{trajecs}}{\text{meas}} \right\rfloor \left[ T(LL, V) + T(FL, V) \right] + \text{niters} \cdot T(CG, V) \]
Step 3: Communication Pattern

- 4d domain is cut in all dimensions (cubic)
  - 4d nearest-neighbor communication (8 neighbors)
- Allreduce to check CG convergence
  - One per iteration on full process set
- We counted messages and sizes
  - Separate for each kernel
  - See paper for sizes and full model equation!

<table>
<thead>
<tr>
<th>kernel</th>
<th>#Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF</td>
<td>((\text{trajecs} + \text{warms}) \cdot \text{steps} \cdot 1616)</td>
</tr>
<tr>
<td>GF</td>
<td>((\text{trajecs} + \text{warms}) \cdot \text{steps} \cdot 828)</td>
</tr>
<tr>
<td>LL</td>
<td>((3 \cdot \text{steps} \cdot (\text{trajecs} + \text{warms}) + \frac{\text{trajecs}}{\text{meas}}) \cdot 8)</td>
</tr>
<tr>
<td>FL</td>
<td>((3 \cdot \text{steps} \cdot (\text{trajecs} + \text{warms}) + \frac{\text{trajecs}}{\text{meas}}) \cdot 288)</td>
</tr>
</tbody>
</table>
Step 6: Communication Parameters

- Two options:
  - Semi-empiric – fit measurements to get effective latency and bandwidth
    - Enables to check if they match expectations
  - Analytic – derive parameters separately (e.g., documentation or separate benchmark)
    - Often problematic if they do not match expectations
- Our model was analytic
  - Uses LogGP parameters (measured by Netgauge [1])

The Fully-Parameterized Parallel Model
Conclusions and Future Work

- Models in use for predictions and optimizations
  - First successes: ~10-20% improved performance [1]
- Simple strategy enables application team models
  - Better chance to be maintained than external models
  - Critical for performance-centric software development
- We need (and work on):
  - More examples for irregular/dynamic codes
  - Better tool support for modeling