Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes

A. Calotoiu¹, T. Hoefler², M. Poke¹, F. Wolf¹

¹) German Research School for Simulation Sciences
²) ETH Zurich

September 13, 2013
Analytical performance modeling

Disadvantages
• Time consuming
• Danger of overlooking unscalable code

Identify kernels
• Parts of the program that dominate its performance at larger scales
• Identified via small-scale tests and intuition

Create models
• Laborious process
• Still confined to a small community of skilled experts
Our approach

Generate an empirical model for each part of the program automatically
- Run a manageable number of small-scale performance experiments
- Launch our tool
- Compare extrapolated performance to expectations

Key ideas
- Exploit that space of function classes underlying such model is small enough to be searched by a computer program
- Abandon model accuracy as the primary success metric and rather focus on the binary notion of scalability bugs
- Create requirements models alongside execution models
SPPEXA project Catwalk

- German Research School for Simulation Sciences, Laboratory for Parallel Programming (Prof. Dr. Felix Wolf)
- Technische Universität Darmstadt, Institute for Scientific Computing (Prof. Dr. Christian Bischof)
- Swiss Federal Institute of Technology Zurich, Institute of Computer Systems, (Prof. Dr. Torsten Hoefler)
- Forschungszentrum Jülich, Jülich Supercomputing Centre (Dr.-Ing. Bernd Mohr)
- Goethe University Frankfurt, Goethe Center for Scientific Computing (Prof. Dr. Gabriel Wittum)
Scalability bug detector

**Input**
- Set of performance measurements (profiles) on different processor counts \( \{p_1, \ldots, p_{\text{max}}\} \) w/ weak scaling
- Individual measurement broken down by program region (call path)

**Output**
- List of program regions (kernels) ranked by their predicted execution time at target scale \( p_t > p_{\text{max}} \)
- Or ranked by growth function \( (p_t \rightarrow \infty) \)

- Not 100\% accurate but good enough to draw attention to right kernels
- False negatives when phenomenon at scale is not captured in data
- False positives possible but unlikely

- Can also model parameters other than \( p \)
Workflow

Statistical quality control → Performance measurements → Performance profiles → Model generation → Scaling models → Accuracy saturated?

Yes → Model refinement → No → Scaling models → Accuracy saturated?

Kernel refinement → Statistical quality control

Model generation → Scaling models → Accuracy saturated?

Accuracy saturated? → Yes → Model refinement → No

No → Scaling models → Accuracy saturated?

Accuracy saturated? → Yes → Model refinement → No
Model generation

Performance Model Normal Form (PMNF)

\[ f(p) = \sum_{k=1}^{n} c_k \times p^{i_k} \times \log_2^{j_k}(p) \]

- Not exhaustive but works in most practical scenarios
- An assignment of \( n, i_k \) and \( j_k \) is called model hypothesis
- \( i_k \) and \( j_k \) are chosen from sets \( I, J \subset Q \)
- \( n, |I|, |J| \) don’t have to be arbitrarily large to achieve good fit

Instead of deriving model through reasoning, make reasonable choices for \( n, I, J \) and try all assignment options one by one
- Select winner through cross-validation
Model refinement

- Start with coarse approximation
- Refine to the point of statistical shrinkage
  → Protection against over-fitting
Requirements modeling

Disagreement may be indicative of wait states
Evaluation

We demonstrate that our tool

• identifies a scalability issue in a code that is known to have one
• does not identify a scalability issue in a code that is known to have none
• identifies two scalability issues in a code that was thought to have only one

Test platform:
IBM Blue Gene/Q
Juqueen in Jülich

\[ I = \left\{ \frac{0}{2}, \frac{1}{2}, \frac{2}{2}, \frac{3}{2}, \frac{4}{2}, \frac{5}{2}, \frac{6}{2} \right\} \]
\[ J = \{0, 1, 2\} \]
\[ n = 5 \]
Sweep3D

Solves neutron transport problem
• 3D domain mapped onto 2D process grid
• Parallelism achieved through pipelined wave-front process

LogGP model for communication developed by Hoisie et al.

\[ t^{\text{comm}} = [2(p_x + p_y - 2) + 4(n^{\text{sweep}} - 1)] \times t^{\text{msg}} \]

\[ t^{\text{comm}} = c \times \sqrt{p} \]
## Sweep3D (2)

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Runtime[%] $p_t=262k$</th>
<th>Increase $t(p=262k)/t(p=64)$</th>
<th>Model [s] $t = f(p)$</th>
<th>Predictive error [%] $p_t=262k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>sweep-&gt;MPI_Recv</td>
<td>65.35</td>
<td>16.54</td>
<td>4.03$\sqrt{p}$</td>
<td>5.10</td>
</tr>
<tr>
<td>sweep</td>
<td>20.87</td>
<td>0.23</td>
<td>582.19</td>
<td>0.01</td>
</tr>
<tr>
<td>global_int_sum-&gt;MPI_Allreduce</td>
<td>12.89</td>
<td>18.68</td>
<td>$1.06\sqrt{p}+0.03\sqrt{p}\log(p)$</td>
<td>13.60</td>
</tr>
<tr>
<td>sweep-&gt;MPI_Send</td>
<td>0.40</td>
<td>0.23</td>
<td>$11.49+0.09\sqrt{p}\log(p)$</td>
<td>15.40</td>
</tr>
<tr>
<td>source</td>
<td>0.25</td>
<td>0.04</td>
<td>$6.86+9.13\times10^{-5}\log(p)$</td>
<td>0.01</td>
</tr>
</tbody>
</table>

$p_i \leq 8k$
Sweep3D (3)

![Graph showing the relationship between processes and time, with markers for data, prediction, and relative error. The x-axis represents processes, and the y-axis represents time (s) and relative error (%).]
MILC/su3_rmd – code from MILC suite of QCD codes with performance model manually created by Hoefler et al.

- Time per process should remain constant except for a rather small logarithmic term caused by global convergence checks

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Model [s] t=f(p)</th>
<th>Predictive Error [%] p_i=64k</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute_gen_staple_field</td>
<td>2.40*10^{-2}</td>
<td>0.43</td>
</tr>
<tr>
<td>g_vecdoublesum&gt;MPI_Allreduce</td>
<td>6.30*10^{-6}*log_2p</td>
<td>0.01</td>
</tr>
<tr>
<td>mult_adj_su3_fieldlink_lathwec</td>
<td>3.80*10^{-3}</td>
<td>0.04</td>
</tr>
</tbody>
</table>

p_i ≤ 16k
## MILC – Varying grid points per process

**Test platform:** Juropa in Jülich (Intel Nehalem cluster)  
**p = 32= constant**

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Flops</th>
<th>Visits</th>
<th>Flops/Visit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model[s]</td>
<td>flops=f(V)</td>
<td>Model visits=f(V)</td>
<td>Model</td>
</tr>
<tr>
<td></td>
<td>$R^2$ [$10^{-3}$]</td>
<td>$R^2$ [$10^{-3}$]</td>
<td>$R^2$ [$10^{-3}$]</td>
</tr>
<tr>
<td>load lnglinks</td>
<td>$5.64 \times 10^4 V$</td>
<td>$0.030$</td>
<td>$2.31 \times 10^3$</td>
</tr>
<tr>
<td>load_fatlinks_cpu</td>
<td>$1.95 \times 10^6 V$</td>
<td>$0.210$</td>
<td>$7.14 \times 10^4$</td>
</tr>
<tr>
<td>ks_congrad</td>
<td>$1.16 \times 10^8$ +</td>
<td>$0.292$</td>
<td>$5.11 \times 10^4$ +</td>
</tr>
<tr>
<td></td>
<td>$3.24 \times 10^5 V^{5/4}$</td>
<td></td>
<td>$1.38 \times 10^4 V^{1/4}$</td>
</tr>
<tr>
<td>imp_gauge_force_cpu</td>
<td>$1.65 \times 10^6 V$</td>
<td>$0.015$</td>
<td>$7.40 \times 10^4$</td>
</tr>
<tr>
<td>eo_fermion_force_two_terms_site</td>
<td>$4.02 \times 10^6 V$</td>
<td>$0.002$</td>
<td>$1.27 \times 10^5$</td>
</tr>
</tbody>
</table>
Core of the Community Atmospheric Model (CAM)
- Spectral element dynamical core on a cubed sphere grid

<table>
<thead>
<tr>
<th>Kernel</th>
<th>p₁≤15k</th>
<th>p₁≤43k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model [s] t = f(p)</td>
<td>Predictive error [%] p₁ = 130k</td>
</tr>
<tr>
<td>Box_rearrange-&gt;MPI_Reduce</td>
<td>0.03+2.53<em>10⁻⁵ p</em> √p+1.24*10⁻¹² p³</td>
<td>57.02</td>
</tr>
<tr>
<td>Vlaplace_sphere_vk</td>
<td>49.53</td>
<td>99.32</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compute_and_apply_rhs</td>
<td>48.68</td>
<td>1.65</td>
</tr>
</tbody>
</table>
Two issues

Number of iterations inside a subroutine grows with $p^2$
- Ceiling for up to and including 15k
- Developers were aware of this issue and had developed work-around

Growth of time spent in reduce function grows with $p^3$
- Previously unknown
- Function invoked during initialization to funnel data to dedicated I/O processes
- Execution time at 183k ~ 2h, predictive error ~40%

The G8 Research Councils Initiative on Multilateral Research Funding
Interdisciplinary Program on Application Software towards Exascale Computing for Global Scale Issues
Automated performance modeling is feasible

Generated models accurate enough to identify scalability bugs and in good agreement with hand-crafted models

Advantages of mass production also performance models
• Approximate models are acceptable as long as the effort to create them is low and they do not mislead the user
• Code coverage is as important as model accuracy

Future work
• Study influence of further hardware parameters
• More efficient traversal of search space (allows increase of modeling parameters)
• Integration into Scalasca