Mass-producing insightful performance models of parallel applications

Felix Wolf, TU Darmstadt
Acknowledgement

- Alexandru Calotoiu (TU Darmstadt)
- Torsten Höfler (ETH Zurich)
- Sergei Shudler (TU Darmstadt)
- Alexandre Strube (Jülich Supercomputing Centre)
- Andreas Vogel (GU Frankfurt)

- Marius Poke (RWTH Aachen)
- Paul Wiedeking (RWTH Aachen)
Latent scalability bugs

System size

Execution time
Scalability model

\[ 3 \cdot 10^4 p^2 + c \]
Analytical scalability modeling

**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

**Disadvantages**
- Time consuming
- Danger of overlooking unscalable code
Automated empirical modeling (2)

main() {
    foo()
    bar()
    compute()
}

Input

Output

Performance measurements (profiles)

Instrumentation

• All functions

Automated modeling

Ranking:
• Asymptotic
• Target scale \( p_t \)

<table>
<thead>
<tr>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_3 )</th>
<th>( p_4 )</th>
<th>( p_5 )</th>
<th>( p_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1,024</td>
<td>2,048</td>
<td>4,096</td>
</tr>
</tbody>
</table>

1. foo
2. compute
3. main
4. bar
[...]

1/17/15 | Department of Computer Science | Laboratory for Parallel Programming | Prof. Dr. Felix Wolf | 6
Primary focus on scaling trend

Our ranking

1. $F_1$
2. $F_3$
3. $F_2$
Primary focus on scaling trend

![Diagram showing scaling trend]  

**Actual measurement in laboratory conditions**

Our ranking:

1. $F_1$
2. $F_3$
3. $F_2$
Primary focus on scaling trend

Our ranking

1. $F_1$
2. $F_3$
3. $F_2$
Model building blocks

Computation

- LU
  \( t(p) \sim c \)

- FFT
  \( t(p) \sim \log_2(p) \)

- Naïve N-body
  \( t(p) \sim p \)

- Samplesort
  \( t(p) \sim p^2 \log_2^2(p) \)

Communication

- LU
  \( t(p) \sim c \)

- FFT
  \( t(p) \sim c \)

- Naïve N-body
  \( t(p) \sim p \)

- Samplesort
  \( t(p) \sim p^2 \)
Performance model normal form

\[ f(p) = \sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot \log_{j_k}^2(p) \]

\[ n \in \mathbb{N} \]
\[ i_k \in I \]
\[ j_k \in J \]
\[ I, J \subset \mathbb{Q} \]

\[ n = 1 \]
\[ I = \{0,1,2\} \]
\[ J = \{0,1\} \]
Performance model normal form

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

\[ n \in \mathbb{N} \]

\[ n = 2 \]

\[ I = \{0,1,2\} \]

\[ J = \{0,1\} \]

\[ c_1 \cdot \log(p) + c_2 \cdot p \]

\[ c_1 \cdot \log(p) + c_2 \cdot p \cdot \log(p) \]

\[ c_1 \cdot \log(p) + c_2 \cdot p^2 \]

\[ c_1 \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p) \]

\[ c_1 + c_2 \cdot p \]

\[ c_1 + c_2 \cdot p \cdot \log(p) \]

\[ c_1 + c_2 \cdot p^2 \]

\[ c_1 + c_2 \cdot p^2 \cdot \log(p) \]

\[ c_1 + c_2 \cdot \log(p) \]

\[ c_1 + c_2 \cdot p \cdot \log(p) \]

\[ c_1 + c_2 \cdot \log(p) + c_2 \cdot p \]

\[ c_1 + c_2 \cdot \log(p) + c_2 \cdot p^2 \]

\[ c_1 + c_2 \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p) \]

\[ c_1 \cdot p + c_2 \cdot p \cdot \log(p) \]

\[ c_1 \cdot p + c_2 \cdot p^2 \]

\[ c_1 \cdot p + c_2 \cdot p^2 \cdot \log(p) \]

\[ c_1 \cdot p \cdot \log(p) + c_2 \cdot p \]

\[ c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \]

\[ c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p) \]

\[ c_1 \cdot p^2 + c_2 \cdot p^2 \cdot \log(p) \]
Modeling operations vs. time

Program

Computation

FLOPS
Load
Store

Communication

#Msgs
#Bytes

Disagreement may be indicative of wait states

Time
Case studies

Sweep3d
Lulesh
Milc
HOMME
JUSPIC
XNS
NEST
UG4
MP2C
BLAST
MPI
OpenMP
**Sweep3D - Neutron transport simulation**

- LogGP model for communication developed by Hoisie et al.

\[
t_{\text{comm}} = [2(p_x + p_y - 2) + 4(n_{\text{sweep}} - 1)] \cdot t_{\text{msg}}
\]

\[
t_{\text{comm}} = c \cdot \sqrt{p}
\]

<table>
<thead>
<tr>
<th>Kernel [2 of 40]</th>
<th>Model [s] ( t = f(p) )</th>
<th>Predictive error [%] ( p_t=262k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>sweep \rightarrow MPI_Recv</td>
<td>4.03(\sqrt{p})</td>
<td>5.10</td>
</tr>
<tr>
<td>sweep</td>
<td>582.19</td>
<td>0.01</td>
</tr>
</tbody>
</table>

\( p_i \leq 8k \)

#bytes = const.
#msg = const.
HOMME – Climate

Core of the Community Atmospheric Model (CAM)

- Spectral element dynamical core on a cubed sphere grid

<table>
<thead>
<tr>
<th>Kernel [3 of 194]</th>
<th>Model [s] $t = f(p)$</th>
<th>Predictive error [%] $p_t = 130k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>box_rearrange $\rightarrow$ MPI_Reduce</td>
<td>$0.026 + 2.53 \cdot 10^{-6} p \cdot \sqrt{p} + 1.24 \cdot 10^{-12} p^3$</td>
<td>57.02</td>
</tr>
<tr>
<td>vlaplace_sphere_vk</td>
<td>49.53</td>
<td>99.32</td>
</tr>
<tr>
<td>compute_and_apply_rhs</td>
<td>48.68</td>
<td>1.65</td>
</tr>
</tbody>
</table>

$p_i \leq 15k$
HOMME – Climate

Core of the Community Atmospheric Model (CAM)

- Spectral element dynamical core on a cubed sphere grid

<table>
<thead>
<tr>
<th>Kernel [3 of 194]</th>
<th>Model [s] ( t = f(p) )</th>
<th>Predictive error [%] ( p_t = 130k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>box_rearrange → MPI_Reduce</td>
<td>( 3.63 \cdot 10^{-6} p \cdot \sqrt{p} + 7.21 \cdot 10^{-13} p^3 )</td>
<td>30.34</td>
</tr>
<tr>
<td>vlaplace_sphere_vk</td>
<td>( 24.44 + 2.26 \cdot 10^{-7} p^2 )</td>
<td>4.28</td>
</tr>
<tr>
<td>compute_and_apply_rhs</td>
<td>49.09</td>
<td>0.83</td>
</tr>
</tbody>
</table>

\( p_i \leq 43k \)
are currently working on a solution. The example demonstrated a problem that was reported back to the developers at NCAR. One hour in the large-scale experiments we conducted, the problem was revealed in terms of the overall runtime, which is in the order of minutes. While still not yet critical, the performance phase of the code that was not assumed to be performance-relevant in larger production runs started to show signs of issues. While not yet critical, the performance at larger scales started to become significant even if executed at smaller scales, it will have an explosive effect on the overall runtime.

The code reuse is needed to funnel data to dedicated I/O processes. In contrast to the previous problem, the cubic growth of the number of iterations being executed to grow rapidly, causing a significant drop in performance. It turned out the developers were aware of this issue and had already developed a temporary solution, involving manual adjustments of their source code. However, the developers were not informed of this issue until today is that it belongs to the initialization phase of the code that was not assumed to be performance-relevant in larger production runs.

The reason why this phenomenon remained unnoticed until today is that it belongs to the initialization phase of the code that was not assumed to be performance-relevant in larger production runs. While still not yet critical, the performance at larger scales started to become significant even if executed at smaller scales, it will have an explosive effect on the overall runtime.

The code reuse is needed to funnel data to dedicated I/O processes. In contrast to the previous problem, the cubic growth of the number of iterations being executed to grow rapidly, causing a significant drop in performance. It turned out the developers were aware of this issue and had already developed a temporary solution, involving manual adjustments of their source code. However, the developers were not informed of this issue until today.

Interestingly, the formula by which the number of iterations is computed contained a term that limits the number of iterations to one for up to and including the number of processes being executed. The reason why this phenomenon remained unnoticed until today is that it belongs to the initialization phase of the code that was not assumed to be performance-relevant in larger production runs.

The coexistence of the phenomenon was revealed in terms of the overall runtime, which is in the order of minutes. While still not yet critical, the performance phase of the code that was not assumed to be performance-relevant in larger production runs started to show signs of issues. While not yet critical, the performance at larger scales started to become significant even if executed at smaller scales, it will have an explosive effect on the overall runtime.

The code reuse is needed to funnel data to dedicated I/O processes. In contrast to the previous problem, the cubic growth of the number of iterations being executed to grow rapidly, causing a significant drop in performance. It turned out the developers were aware of this issue and had already developed a temporary solution, involving manual adjustments of their source code. However, the developers were not informed of this issue until today.

Interestingly, the formula by which the number of iterations is computed contained a term that limits the number of iterations to one for up to and including the number of processes being executed. The reason why this phenomenon remained unnoticed until today is that it belongs to the initialization phase of the code that was not assumed to be performance-relevant in larger production runs.

The code reuse is needed to funnel data to dedicated I/O processes. In contrast to the previous problem, the cubic growth of the number of iterations being executed to grow rapidly, causing a significant drop in performance. It turned out the developers were aware of this issue and had already developed a temporary solution, involving manual adjustments of their source code. However, the developers were not informed of this issue until today.

Interestingly, the formula by which the number of iterations is computed contained a term that limits the number of iterations to one for up to and including the number of processes being executed. The reason why this phenomenon remained unnoticed until today is that it belongs to the initialization phase of the code that was not assumed to be performance-relevant in larger production runs.

The code reuse is needed to funnel data to dedicated I/O processes. In contrast to the previous problem, the cubic growth of the number of iterations being executed to grow rapidly, causing a significant drop in performance. It turned out the developers were aware of this issue and had already developed a temporary solution, involving manual adjustments of their source code. However, the developers were not informed of this issue until today.

Interestingly, the formula by which the number of iterations is computed contained a term that limits the number of iterations to one for up to and including the number of processes being executed. The reason why this phenomenon remained unnoticed until today is that it belongs to the initialization phase of the code that was not assumed to be performance-relevant in larger production runs.
• Numerical framework for grid-based solution of partial differential equations (~500,000 lines of C++ code, 2,000 kernels)
  • Application: drug diffusion through the human skin
• In general, all kernels scale well
  • Multigrid solver kernel (MGM) scales logarithmically
  • Number of iterations needed by the unpreconditioned conjugate gradient (CG) method depends on the mesh size
    • Increases by factor of two with each refinement
    • Will therefore suffer from iteration count increase in weak scaling

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Model (time [s])</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG</td>
<td>$0.227 + 0.31 \times p^{0.5}$</td>
</tr>
<tr>
<td>MGM</td>
<td>$0.219 + 0.0006 \times \log^2(p)$</td>
</tr>
</tbody>
</table>
Issue with MPI communicator group creation

- Create MPI communicator groups for each level of multigrid hierarchy
- Exclude processes that do not own a grid part on that level
- **Before**: Membership info communicated using MPI_Allreduce with array of length $p$ - non-scalable $p \times O(MPI\_Allreduce)$ complexity
- **Now**: MPI_Allreduce replaced by MPI_Comm_split - enhanced algorithms of which are known to have $O(\log^2 p)$ complexity

(C. Siebert, F. Wolf: Parallel sorting with minimal data. Recent Advances in the Message Passing Interface, 2011)
Which problem?
Where in the program?
Which process?

Tutorials at EuroMPI’15 & SC15
Algorithm engineering

algorithm engineering

realistic models
real Inputs

design

falsifiable hypotheses
induction

experiments
appl. engin.

implementation

algorithm−libraries

analysis
deduction
perf.−guarantees

Courtesy of Peter Sanders, KIT
How to validate scalability in practice?

Program

Small text book example

Real application

Expectation

Verifiable analytical expression

Asymptotic complexity

\#FLOPS = n^2(2n - 1)

\#FLOPS = O(n^{2.8074})
HPC libraries

- Focus on algorithms rather than applications
- Theoretical expectations more common
- Reuse factor makes scalability even more important

Example:
MPI communication library
Scalability evaluation framework

- Search space generation
- Model generation
- Benchmark

Expectation: $\log p$
Scaling model: $p$
Divergence model: $p/\log p$

- Initial validation
- Comparing alternatives
- Regression testing
Customized search space

- Constructed around expectation
- Supports wider range of model functions than original PMNF
### MPI

<table>
<thead>
<tr>
<th>Platform</th>
<th>Juqueen</th>
<th>Juropa</th>
<th>Piz Daint</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Barrier [s]</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>$O (\log p)$</td>
<td>$O (p^{0.5})$</td>
<td>$O (p^{0.67} \log p)$</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0.87</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>Match</td>
<td>✔</td>
<td>✔</td>
<td>❌</td>
</tr>
<tr>
<td><strong>Bcast [s]</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>$O (\log p)$</td>
<td>$O (p^{0.5}/\log p)$</td>
<td>$O (p^{0.67})$</td>
</tr>
<tr>
<td>$R^2$</td>
<td>0</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>Match</td>
<td>✔</td>
<td>✔</td>
<td>❌</td>
</tr>
<tr>
<td><strong>Comm_dup [B]</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>$2.2e5$</td>
<td>256</td>
<td>$3770 + 18p$</td>
</tr>
<tr>
<td>$R^2$</td>
<td>1</td>
<td>1</td>
<td>0.99</td>
</tr>
<tr>
<td>Match</td>
<td>✔</td>
<td>✔</td>
<td>❌</td>
</tr>
</tbody>
</table>
MAFIA

Sub-space clustering code used in data-mining

- Cluster dimensionality $k$ is the model parameter
- Result: observed behavior matched the expectations

<table>
<thead>
<tr>
<th></th>
<th>gen</th>
<th>dedup</th>
<th>pcount</th>
<th>unjoin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expectation</td>
<td>$O(k^3 2^k)$</td>
<td>$O(k^4 2^k)$</td>
<td>$O(k^2 k)$</td>
<td>$O(k^3 2^k)$</td>
</tr>
<tr>
<td>Model</td>
<td>$O(k^4 2^k)$</td>
<td>$O(k^4 2^k)$</td>
<td>$O(k^2 k)$</td>
<td>$O(k^2 2^k)$</td>
</tr>
<tr>
<td>Divergence</td>
<td>$O(k)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1/k)$</td>
</tr>
<tr>
<td>Match</td>
<td>~</td>
<td>✔</td>
<td>✔</td>
<td>~</td>
</tr>
</tbody>
</table>
Mass-producing performance models

- Is feasible
- Offers insight
- Requires low effort
- Improves code coverage


S. Shudler, A. Calotoiu, T. Hoefler, A. Strube, F. Wolf: Exascaling Your Library: Will Your Implementation Meet Your Expectations?. In Proc. of the International Conference on Supercomputing (ICS), Newport

Thank you!