The price performance of performance models

Felix Wolf, Technical University of Darmstadt
IEEE Cluster Conference 2020, Kobe, Japan
## Acknowledgement

**TU Darmstadt**
- Yannick Berens
- Alexandru Calotoiu
- Alexander Geiß
- Alexander Graf
- Daniel Lorenz
- Benedikt Naumann
- Thorsten Reimann
- Sebastian Rinke
- Marcus Ritter
- Sergei Shudler

**ETH Zurich**
- Alexandru Calotoiu
- Marcin Copik
- Tobias Grosser
- Torsten Hoefler
- Nicolas Wicki

**LLNL**
- David Beckingsale
- Christopher Earl
- Ian Karlin
- Martin Schulz

**FZ Jülich**
- Alexandre Strube
Scaling your code can harbor performance surprises*…

*Goldsmith et al., 2007
Performance model

Formula that expresses relevant performance metric as a function of one or more execution parameters

\[ t = 3 \cdot 10^{-4} p^2 + c \]

Analytical (i.e., manual) creation challenging for entire programs

- Identify kernels
  - Incomplete coverage
- Create models
  - Laborious, difficult
Empirical performance modeling

Performance measurements with different execution parameters $x_1, \ldots, x_n$

\[
t = f (x_1, \ldots, x_n)
\]

Machine learning

Alternative metrics: FLOPs, data volume…
Challenges

Applications

Run-to-run variation / noise

System

Cost of the required experiments
How to deal with noisy data

- Introduce prior into learning process
  - Assumption about the probability distribution generating the data

![Diagram showing a clock and a wall with labels for time and effort, and a set of tasks including computation, memory access, communication, and I/O.](image)
Typical algorithmic complexities in HPC

**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 (PMNF)

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

Single parameter
[Calotoiu et al., SC13]

\[ f(x_1, \ldots, x_m) = \sum_{k=1}^{n} c_k \prod_{l=1}^{m} x_{l}^{i_{kl}} \cdot \log_{2}^{j_{kl}}(x_{l}) \]

Multiple parameters
[Calotoiu et al., Cluster’16]

Heuristics to reduce search space
Extra-P 3.0

New BSD license
http://www.scalasca.org/software/extra-p/download.html
## MPI implementations

[Shudler et al., IEEE TPDS 2019]

<table>
<thead>
<tr>
<th>Platform</th>
<th>Juqueen</th>
<th>Juropa</th>
<th>Piz Daint</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Allreduce [s]</strong></td>
<td>Expectation: $O (\log p)$</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>Comm_dup [B]</strong></td>
<td>Expectation: $O (1)$</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>
**Kripke - example w/ multiple parameters**

**SweepSolver**

Main **computation** kernel

Expectation – Performance depends on **problem size**

\[ t \sim d \cdot g \]

Actual model:

\[ t = 5 + d \cdot g + 0.005 \cdot \sqrt[3]{p} \cdot d \cdot g \]

*Coefficients have been rounded for convenience*

**MPI_Testany**

Main **communication** kernel: 3D wave-front communication pattern

Expectation – Performance depends on **cubic root of process count**

\[ t \sim \sqrt[3]{p} \]

Actual model:

\[ t = 7 + \sqrt[3]{p} + 0.005 \cdot \sqrt[3]{p} \cdot d \cdot g \]

**Smaller compounded effect discovered**
Lightweight requirements engineering for (exascale) co-design

- Collect portable requirement metrics
- Derive requirement models
- Extrapolate to new system

<table>
<thead>
<tr>
<th>Resource</th>
<th>Metric (per process)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory footprint</td>
<td># Bytes used (resident memory size)</td>
</tr>
<tr>
<td>Computation</td>
<td># Floating-point operations (#FLOP)</td>
</tr>
<tr>
<td>Network communication</td>
<td># Bytes sent / received</td>
</tr>
<tr>
<td>Memory access</td>
<td># Loads / stores; stack distance</td>
</tr>
</tbody>
</table>

Counters often more noise resilient than time
Application demands for different resources scale differently

<table>
<thead>
<tr>
<th>Metric</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Bytes used</td>
<td>$10^5 \cdot n \log n$</td>
</tr>
<tr>
<td>#FLOP</td>
<td>$10^5 \cdot n \log n \cdot p^{0.25} \log p$</td>
</tr>
<tr>
<td>#Bytes sent &amp; received</td>
<td>$10^3 \cdot n \cdot p^{0.25} \log p$</td>
</tr>
<tr>
<td>#Loads &amp; stores</td>
<td>$10^5 \cdot n \log n \cdot \log p$</td>
</tr>
<tr>
<td>Stack distance</td>
<td>Constant</td>
</tr>
</tbody>
</table>

Models are per process
- $p$ – Number of processes
- $n$ – Problem size per process

Calculate relative changes of resource demand by scaling $p$ and $n$
- $n$ is a function of the memory size
- $p$ is a function of the number of cores / sockets
Co-design

Given a budget and a set of applications, how can we best invest in upgrades for a given hardware system?

Examples
- Double the racks
- Double the sockets
- Double the memory

[Calotoiu et al., Cluster’18]
### Three upgrades – summary

<table>
<thead>
<tr>
<th>Ratios</th>
<th>Apps</th>
<th>Kripke</th>
<th>LULESH</th>
<th>MILC</th>
<th>Relearn</th>
<th>icoFoam</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Upgrade A: Double the racks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem size per process</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Overall problem size</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Computation</td>
<td>1</td>
<td>1.2</td>
<td>1.2</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>1</td>
<td>1.2</td>
<td>1.2</td>
<td>1</td>
<td>0.7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Memory accesses</td>
<td>2</td>
<td>1.2</td>
<td>2.8</td>
<td>2</td>
<td>0.7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>System Upgrade B: Double the sockets</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem size per process</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>Overall problem size</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
<td>0.6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Computation</td>
<td>0.5</td>
<td>0.6</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>0.5</td>
<td>0.6</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>Memory accesses</td>
<td>0.5</td>
<td>1.0</td>
<td>1.4</td>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td><strong>System Upgrade C: Double the memory</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem size per process</td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>1.4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Overall problem size</td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>1.4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Computation</td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>1.7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>1.4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Memory accesses</td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>4</td>
<td>1.4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Best option:** LULESH

**Worst option:**
Task-graph modeling
[Shudler et al., PPoPP’17]

- Nodes – tasks, edges – dependencies
- $p,n$ – processing elements, input size
- $T_1(n)$ – all the task times (work)
- $T_\infty(n)$ – longest path (depth)
- $\pi(n) = \frac{T_1(n)}{T_\infty(n)}$ – average parallelism

![Task-graph diagram]

$T_1 = 45$

$T_\infty = 25$
Experiments can be expensive
Need $5^{(m+1)}$ experiments, $m = \#\text{parameters}$
Multi-parameter modeling in Extra-P

Find best single-parameter model

Combine them in the most plausible way (+, *, none)

Generation of candidate models and selection of best fit
How many data points do we really need?

<table>
<thead>
<tr>
<th>Processes $p$</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem size per process $s$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Learning cost-effective sampling strategies
[Ritter et al., IPDPS’20]

\[ f(x_1, \ldots, x_m) = \sum_{k=1}^{n} c_k \prod_{l=1}^{m} x_l^{i_{kl}} \cdot \log_2^{j_{kl}} (x_l) \]

*Function generator*

*Noise module*

*Synthetic measurement*

*Reinforcement learning agent*

*Empirical model*

*Evaluation*

*Ground truth*

*Feedback*

*Prediction*

*Extra-P*

*Selected parameter values*
Heuristic parameter-value selection strategy

1. Measure min. amount points required for modeling
2. Create a model using Extra-P
3. Final model
4. If cost < budget
   - yes: Create new model
   - no: Gather on additional measurement (assumed to be cheapest)
Synthetic data evaluation

![Graph showing synthetic data evaluation](image)
Synthetic evaluation results

Measurements used / Percentage of cost

1 parameter, 5% noise

<table>
<thead>
<tr>
<th>Repetitions</th>
<th>Measurements used / Percentage of cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5 / 100%</td>
</tr>
<tr>
<td></td>
<td>9 / 12%</td>
</tr>
<tr>
<td></td>
<td>11 / 13%</td>
</tr>
<tr>
<td></td>
<td>15 / 17%</td>
</tr>
<tr>
<td></td>
<td>25 / 100%</td>
</tr>
</tbody>
</table>

2 parameters, 5% noise

<table>
<thead>
<tr>
<th>Repetitions</th>
<th>Measurements used / Percentage of cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5 / 100%</td>
</tr>
<tr>
<td></td>
<td>9 / 12%</td>
</tr>
<tr>
<td></td>
<td>11 / 13%</td>
</tr>
<tr>
<td></td>
<td>15 / 17%</td>
</tr>
<tr>
<td></td>
<td>25 / 100%</td>
</tr>
</tbody>
</table>
Synthetic evaluation results

3 parameters, 5% noise

Percentage of accurate models*

Repetitions
2  4  6

Measurements used / Percentage of cost
13 / 1.7%  15 / 1.8%  25 / 2.2%  75 / 11%  125 / 100%

13 / 1.7%  15 / 1.8%  25 / 2.2%  75 / 11%  125 / 100%
Synthetic evaluation results

4 parameters, 5% noise

Percentage of accurate models*

Repetitions
2 4 6

Measurements used / Percentage of cost

- 17 / 0.1%
- 18 / 0.12%
- 25 / 0.17%
- 125 / 1.2%
- 250 / 4.2%
- 625 / 100%

2 4 6
Synthetic evaluation results

4 parameters, 1% noise

Percentage of accurate models*

Repetitions
2 4 6

Measurements used / Percentage of cost

17 / 0.1% 18 / 0.12% 25 / 0.17% 125 / 1.2% 250 / 4.2% 625 / 100%

2 / 0.17%
## Case studies

<table>
<thead>
<tr>
<th>Application</th>
<th>#Parameters</th>
<th>Extra points</th>
<th>Cost savings [%]</th>
<th>Prediction error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>FASTEST</td>
<td>2</td>
<td>0</td>
<td>70</td>
<td>2</td>
</tr>
<tr>
<td>Kripke</td>
<td>3</td>
<td>3</td>
<td>99</td>
<td>39</td>
</tr>
<tr>
<td>Relearn</td>
<td>2</td>
<td>0</td>
<td>85</td>
<td>11</td>
</tr>
</tbody>
</table>
Parameter selection

- The more parameters the more experiments
- Modeling parameters without performance impact is harmful

Taint analysis

Input parameters → Program

Which parameter influences which function?

Taint labels
PerfTaint – Taint-based performance modeling

Annotate parameters
```
register_variable("size", &size);
```

Static loop analysis

Dynamic taint analysis

DataFlowSanitizer
+ control-flow taint propagation

- Parameter effects & dependencies
- Constant functions

[Copik et al., submitted, Code: spcl/perf-taint@ GitHub]
PerfTaint - White-box performance modeling

<table>
<thead>
<tr>
<th>Parameter identification</th>
<th>Black box (before)</th>
<th>White box (now)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Manual</td>
<td>Taint coverage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment design</th>
<th>Vary all parameters blindly</th>
<th>Exploit knowledge of parameter influence and dependencies</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Instrumentation</th>
<th>All functions</th>
<th>Only functions with parameter influence</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Model generation</th>
<th>All functions</th>
<th>Only functions with parameter influence</th>
</tr>
</thead>
</table>
# Case study – LULESH & MILC

Influence of program parameters

<table>
<thead>
<tr>
<th>LULESH</th>
<th>Total</th>
<th>p</th>
<th>size</th>
<th>regions</th>
<th>iters</th>
<th>balance</th>
<th>cost</th>
<th>p, size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td>349</td>
<td>2</td>
<td>40</td>
<td>15</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>Loops</td>
<td>275</td>
<td>2</td>
<td>78</td>
<td>29</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MILC</th>
<th>Total</th>
<th>p</th>
<th>size</th>
<th>trajecs</th>
<th>warsms</th>
<th>steps</th>
<th>nrest.</th>
<th>niter</th>
<th>mass, beta</th>
<th>nfl.</th>
<th>u0</th>
<th>p, size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td>621</td>
<td>54</td>
<td>53</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loops</td>
<td>874</td>
<td>187</td>
<td>161</td>
<td>39</td>
<td>31</td>
<td>15</td>
<td>1</td>
<td>7</td>
<td>196</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PerfTaint – Taint-based performance modeling

**Overhead**
- 50% less overhead (rel. to Score-P default filter)

**Quality**
- Constant functions
- Perturbation
  - $2.4 \times 10^{-8} p^{0.25} s^3$

**Validity**
- Hardware contention
- Segmented behavior

```c
int foo(int a) {
    if (a < 4) {
        kernel_linear(a);
    } else {
        kernel_log(a);
    }
}
```
Noise-resilient adaptive modeling

DNNs often better at guessing models in the presence of noise
Noise-resilient adaptive modeling
Synthetic evaluation

Relative error
(at unseen point, two ticks in each dimension)

- Adaptive
- Sparse

Lead exponents within 1/3 of ground truth

- Adaptive
- Sparse

2 parameters
Gaussian processes

Goal: better tradeoff between accuracy and cost for specific models

---

Source: https://scikit-learn.org/stable/_images/sphx_glr_plot_gpr_noisy_targets_001.png
Buitinck et al.: API design for machine learning software: experiences from the scikit-learn project, 2013.
New version of Extra-P in Q4 2020

• Includes the new sparse modeler
• Available as a Python package
• No interfaces or external dependencies
• Support for Windows and Linux (Ubuntu)
• Easy installation via pip
• BSD 3-Clause License
## Selected papers

<table>
<thead>
<tr>
<th>Topic</th>
<th>Bibliography</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task-graph modeling</td>
<td>Sergei Shudler, Alexandru Calotoiu, Torsten Hoefler, Felix Wolf: Isoefficiency in Practice: Configuring and Understanding the Performance of Task-based Applications. <strong>PPoPP 2017</strong>.</td>
</tr>
<tr>
<td>Taint-based performance modeling</td>
<td>Marcin Copik, Alexandru Calotoiu, Tobias Grosser, Nicolas Wicki, Felix Wolf, Torsten Hoefler: Extracting Clean Performance Models from Tainted Programs. <strong>Submitted</strong>.</td>
</tr>
</tbody>
</table>
Thank you!

Q&A