Update on the Performance-Modeling Tool Extra-P

Felix Wolf, TU Darmstadt
Acknowlegdement

- David Beckingsale
- Alexandru Calotoiu
- Christopher W. Earl
- Torsten Hoefler
- Kashif Ilyas
- Ian Karlin
- Daniel Lorenz
- Patrick Reisert
- Martin Schulz
- Sergei Shudler
- Andreas Vogel
Latent scalability bugs

System size

Wall time
Motivation

**Performance model** = formula that expresses relevant performance metrics as a function of one or more execution parameters

Manual creation challenging

- Incomplete coverage
- Laborious, difficult

Performance model formula: $3 \cdot 10^{-4} p^2 + c$
Automatic empirical performance modeling

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

Small-scale measurements

Performance model normal form (PMNF)

Generation of candidate models and selection of best fit

**Kernel [2 of 40]**

**Model [s]**

<table>
<thead>
<tr>
<th>sweep</th>
<th>MPI_Recv</th>
<th>4.03\sqrt{p}</th>
</tr>
</thead>
<tbody>
<tr>
<td>sweep</td>
<td>582.19</td>
<td></td>
</tr>
</tbody>
</table>
Extra-P 3.0

- GUI improvements, better stability, additional features
- Tutorials available through VI-HPS and upon request

http://www.scalasca.org/software/extra-p/download.html
Recent developments

1. Performance models with multiple parameters
2. Automatic configuration of the search space
3. Segmented models
4. Iso-efficiency modeling
5. Lightweight requirements engineering for co-design
Models with more than one parameter

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

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

$I = \{0, \frac{1}{4}, \frac{3}{4}, \ldots, \frac{12}{4}\}$

$J = \{0, 1, 2\}$

Search space explosion
- Total number of hypotheses to search: 34,786,300,841,019
- Too slow for any practical purpose
Search space reduction through heuristics

- **Hierarchical search** – Assumes the best multi-parameter model is created out of the combination of the best single parameter hypothesis for each parameter.

- **Modified golden section search** – Speeds up the single parameter search by ordering the hypothesis space and then using a variant of binary search to find the model in logarithmic time rather than linear time.

Calotoiu et al.
Search space reduction

- Assuming 300,000 hypotheses searched per second*
- 3-parameter models

\[ n = 3 \]
\[ m = 3 \]
\[ I = \left\{ \frac{0}{4}, \frac{1}{4}, ..., \frac{12}{4} \right\} \]
\[ J = \{0,1,2\} \]
Search space reduction

• Assuming 300,000 hypotheses searched per second*
• 3-parameter models

*This is optimistic

Exhaustive search

34,786,300,841,019 hypotheses searched
~1 model / 3.5 years

\[ n = 3 \]
\[ m = 3 \]
\[ I = \left\{ \frac{0}{4}, \frac{1}{4}, \ldots, \frac{12}{4} \right\} \]
\[ J = \{0,1,2\} \]
Search space reduction

- Assuming 300,000 hypotheses searched per second*
- 3-parameter models

*This is optimistic

\[ n = 3, \quad m = 3, \quad I = \left\{ \frac{0}{4}, \frac{1}{4}, ..., \frac{12}{4} \right\}, \quad J = \{0,1,2\} \]

Exhaustive search

- 34,786,300,841,019 hypotheses searched
- ~1 model / 3.5 years

- 27,929 hypotheses searched
- ~11 models / second
Search space reduction

- Assuming 300,000 hypotheses searched per second*
- 3-parameter models

*This is optimistic

Exhaustive search

34,786,300,841,019 hypotheses searched
~1 model / 3.5 years

27,929 hypotheses searched
~11 models / second

590 hypotheses searched
~508 models / second

\(n = 3\)
\(m = 3\)
\(I = \left\{ \frac{0}{4}, \frac{1}{4}, \ldots, \frac{12}{4} \right\}\)
\(J = \{0, 1, 2\}\)
Evaluation with synthetic data (100,000 models with two parameters)

Distribution of generated models [%]

- Optimal model identified
- Lead-order term identified
- Lead-order term not identified

- Exhaustive search - 107 hours
- Heuristics - 1.5 hours
Evaluation with application data

Distribution of generated models [%]

- Identical models
- Lead-order terms identical
- Different lead-order terms

Blast (full)  |  Blast (partial)  |  CloverLeaf  |  Kripke

0  |  10  |  20  |  30  |  40  |  50  |  60  |  70  |  80  |  90  |  100
Case study – Kripke

- Neutron transport proxy code
- Three parameters considered
  - Process count – \( p \)
  - Number of directions – \( d \)
  - Number of groups – \( g \)
Expected behavior

**SweepSolver**

Main *computation* kernel

Expectation – Performance depends on problem size

\[ t \sim d \cdot g \]

**MPI_Testany**

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

Expectation – Performance depends on cubic root of process count

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

SweepSolver

Main computation kernel

Expectation – Performance depends on problem size

$t \sim d \cdot g$

Kernels must wait on each other

Actual model:

$t = 5 + d \cdot g + 0.005 \cdot 3p \cdot d \cdot g$

Smaller compounded effect discovered

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:

$t = 7 + \sqrt[3]{p} + 0.005 \cdot \sqrt[3]{p} \cdot d \cdot g$
How to find good PMNF parameters?

Option (1): Rely on default parameters
   → But what if they don't fit the problem?

Option (2): Try those parameters that you expect to fit
   → Requires prior expertise!
      Also, what if your expectation is wrong?

Option (3): Try very large sets $I$, $J$
   → Requires more resources (especially bad for multiple parameters)!

Option (4): Let Extra-P automatically refine the search space based on previous results.
Simplified PMNF

• Use only constant and “lead order” term

\[ f(p) = c_0 + c_1 \cdot p^\alpha \cdot \log_2^\beta p \]

• Want to find values for \( c_0, c_1, \alpha, \) and \( \beta, \) such that model error is minimized
  • \( c_0 \) and \( c_1 \) are determined by regression
  • What about \( \alpha \) and \( \beta? \)
Simplified PMNF

\[ f(p) = c_0 + c_1 \cdot p^\alpha \cdot \log_2^\beta p \]

We define four slices:
- \( \beta = 0, \alpha = ? \)
- \( \beta = 1, \alpha = ? \)
- \( \beta = 2, \alpha = ? \)
- \( \alpha = 0, \beta = ? \)

**Goal:**

*Unimodal* error distribution along each slice
## Evaluation

### Data from previous case studies
- Sweep3D
- MILC
- UG4
- MPI collective operations
- BLAST
- Kripke
- 5–9 points available
- Last data point (largest p) not used for modeling, but to evaluate prediction accuracy

### Results
- 4453 models
- 49% remain unchanged
- 39% get better
- 12% get worse
- Mean relative prediction down from 45.7% to 13.0%
- Improvements in every individual case study

Reisert et al.
Segmented behavior

First behaviour:
\[ p^2 \]

Model predicted by Extra-P:
\[ \log_2^2(p) \]

Second behaviour:
\[ 30 + p \]
Divide data into subsets

![Graph showing runtime versus number of processors for subsets 1, 2, 3, and 6. The graph includes lines for $p^2$ and $30 + p$.](image)
Model each subset and compute nRSS

## Normalized RSS

<table>
<thead>
<tr>
<th>Subset</th>
<th>Model</th>
<th>nRSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 = {1, 4, 9, 16, 25}$</td>
<td>$p^2$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>$s_2 = {4, 9, 16, 25, 36}$</td>
<td>$p^2$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>$s_3 = {9, 16, 25, 36, 37}$</td>
<td>$-49.41 + 33.45 \cdot \sqrt{p}$</td>
<td>0.18</td>
</tr>
<tr>
<td>$s_4 = {16, 25, 36, 37, 38}$</td>
<td>$-28.53 + 23.17 \cdot \log_2(p)$</td>
<td>0.19</td>
</tr>
<tr>
<td>$s_5 = {25, 36, 37, 38, 39}$</td>
<td>$-6.19 + 14.82 \cdot \log_2(p)$</td>
<td>0.16</td>
</tr>
<tr>
<td>$s_6 = {36, 37, 38, 39, 40}$</td>
<td>$30 + p$</td>
<td>$\approx 0$</td>
</tr>
</tbody>
</table>

### High nRSS values

- $s_3$
- $s_4$
- $s_5$

### Heterogeneous subsets

- $s_3$
- $s_4$
- $s_5$
Identify change point

<table>
<thead>
<tr>
<th>Subset</th>
<th>nRSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 = {1, 4, 9, 16, 25}$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>$s_2 = {4, 9, 16, 25, 36}$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>$s_3 = {9, 16, 25, 36, 37}$</td>
<td>0.18</td>
</tr>
<tr>
<td>$s_4 = {16, 25, 36, 37, 38}$</td>
<td>0.19</td>
</tr>
<tr>
<td>$s_5 = {25, 36, 37, 38, 39}$</td>
<td>0.16</td>
</tr>
<tr>
<td>$s_6 = {36, 37, 38, 39, 40}$</td>
<td>$\approx 0$</td>
</tr>
</tbody>
</table>

$nRSS \geq 0.1$

0011110
Identify change point

Valid Patterns

.....0000011100000...

.....000001111100000...

Just Noise

.....01000110010...
Identifying the change point

<table>
<thead>
<tr>
<th>Subset</th>
<th>nRSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 = {1, 4, 9, 16, 25}$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>$s_2 = {4, 9, 16, 25, 36}$</td>
<td>$\approx 0$</td>
</tr>
<tr>
<td>$s_3 = {9, 16, 25, 36, 37}$</td>
<td>0.18</td>
</tr>
<tr>
<td>$s_4 = {16, 25, 36, 37, 38}$</td>
<td>0.19</td>
</tr>
<tr>
<td>$s_5 = {25, 36, 37, 38, 39}$</td>
<td>0.16</td>
</tr>
<tr>
<td>$s_6 = {36, 37, 38, 39, 40}$</td>
<td>$\approx 0$</td>
</tr>
</tbody>
</table>

$nRSS \geq 0.1$

Change Point

001110
HOMME

- Dynamic core of Community Atmosphere Model (CAM)
- Run for $p \in \{600; 1,176; \ldots; 54,150\}$
- 25 out of 664 kernels found segmented
- Change point found between 15,000 and 16,224
- Example: \textit{laplace\_sphere\_wk}

Non-segmented model:

$$f(p) = 27.7 + 2.23 \cdot 10^{-7} \cdot p^2$$

Segmented model:

$$f_{seg}(p) = \begin{cases} 
49.36 & p \leq 15,000 \\
20.8 + 2.3 \cdot 10^{-7} \cdot p^2 & p \geq 16,224 
\end{cases}$$
System upgrade

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

Given a budget and a set of applications, how can we best invest in upgrades for a given hardware system?
Lightweight requirements engineering for (exascale) co-design

<table>
<thead>
<tr>
<th>Resource</th>
<th>Metric</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>
## 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
Response of workload to system upgrades

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>System upgrade A:</strong> Double the racks</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></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></td>
</tr>
<tr>
<td>Computation</td>
<td>1</td>
<td>1.2</td>
<td>1</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</td>
<td>1</td>
<td>0.7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Memory access</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:</strong> Double the sockets</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>0.6</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 access</td>
<td>0.5</td>
<td>1</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:</strong> Double the memory</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>2.8</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>2.8</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>2.8</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>2.8</td>
<td>1.4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Memory access</td>
<td>2</td>
<td>1.4</td>
<td>2</td>
<td>2.8</td>
<td>1.4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
# Publications

7th Workshop on Extreme Scale Programming Tools (ESPT'18)

- Performance tools
- Debugging and correctness tools
- Program development tool chains (incl. IDEs)
- Performance engineering
- Tool technologies for extreme-scale challenges (e.g., scalability, resilience, power)
- Tool support for accelerated architectures
- Tools for networks and I/O
- Tool infrastructures and environments
- Application developer experiences

Author stipends!