TORSTEN HOEFLER

Automatic Performance Models for the Masses
Static and dynamic techniques for application performance modeling

in collaboration with Alexandru Calotoiu and Felix Wolf @ RWTH Aachen
with students Arnamoy Bhattacharyya and Grzegorz Kwasniewski @ SPCL
presented at Co-Design 2015, HPC China, Wuxi, China
Use-cases for performance modeling

1. **Scalability bug prediction**
   - Find latent scalability bugs early on (before machine deployment)
   
   *SC13: A. Calotoiu, TH, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes*

2. **Automated performance (regression) testing**
   - Performance modeling as part of a software engineering discipline in HPC

   *ICS’15: S. Shudler, A. Calotoiu, T. Hoefler, A. Strube, F. Wolf: Exascaling Your Library: Will Your Implementation Meet Your Expectations?*

3. **Guided or automated performance optimization**
   - E.g., near-optimal job scheduling

   *Nan Ding, Wei Xue, et al. (forthcoming)*

4. **Hardware/Software co-design (how to architect systems)**
   - Zhiwei Xu’s “efficiency first” design

   \[ 1.5 \times 10^{-4} x^3 - 2.6 \times 10^{-2} x^2 \]
   \[ R^2 = 0.93 \]

   **VS.**

   \[ 1.5 \times 10^{-4} x^3 - 2.6 \times 10^{-2} x^2 \]
But how to measure and report performance?

- We all think we know it but it’s harder than I thought!
  - How many measurements?
  - How to summarize data?
  - How to summarize many processes?
  
- Attempt to establish a rigorous practice
  - Clarify common problems e.g., Which mean to use when, common statistics issues, ...
  
- Good start for students
  - 12 simple concise rules

- My thesis: give up on (performance) reproducibility?

---

Scientific Benchmarking of Parallel Computing Systems
Twelve ways to tell the masses when reporting performance results

Torsten Hoeffer
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
htoro@inf.ethz.ch

Roberto Belli
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
bellrlab@inf.ethz.ch

ABSTRACT

Measuring and reporting performance of parallel computers constitutes the basis for scientific advancement of high-performance computing (HPC). Most scientific reports show performance improvements of new techniques and are thus obliged to ensure reproducibility or at least interpretability. Our investigation of a stratified sample of 120 papers across three top conferences in the field shows that the state of the practice is lacking. For example, it is often unclear if reported improvements are deterministic or observed by chance. In addition to distilling best practices from existing work, we propose statistically sound analysis and reporting techniques and simple guidelines for experimental design in parallel computing and codify them in a portable benchmarking library. We aim to improve the standards of reporting research results and initiate a discussion in the HPC field. A wide adoption of our minimal set of rules will lead to better interpretability of performance results and improve the scientific culture in HPC.

Categories and Subject Descriptors
D.2.8 [Software Engineering]: Metrics—complexity measures, performance measures

Keywords
Benchmarking, parallel computing, statistics, data analysis

1. INTRODUCTION

Correctly designing insightful experiments to measure and report performance numbers is a challenging task. Yet, there is surprisingly little agreement on standard techniques for measuring, reporting, and interpreting computer performance. For example, common questions such as “How many iterations do I have to run per measurement?”, “How many measurements should I run?”, “Once...
Manual analytical performance 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
  - Error-prone, may overlook unscalable code

Our first step: scalability bug detector

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

Instrumentation
- All functions

Performance measurements (profiles)

\[ p_1 = 128 \]
\[ p_2 = 256 \]
\[ p_3 = 512 \]
\[ p_4 = 1,024 \]
\[ p_5 = 2,048 \]
\[ p_6 = 4,096 \]

Input

Output

Ranking:
1. Asymptotic
2. Target scale \( p_t \)

Automated modeling

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

Weak scaling
Primary focus on scaling trend

Our ranking

1. $F_1$
2. $F_3$
3. $F_2$

Common performance analysis chart in a paper
Primary focus on scaling trend

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. \(F_2\)
How to mechanize the expert? → Survey!

**Computation**
- LU: $t(p) \sim c$
- FFT: $t(p) \sim \log_2(p)$
- Naïve N-body: $t(p) \sim p$
- Samplesort: $t(p) \sim p^2 \log_2(p)$

... 

**Communication**
- LU: $t(p) \sim c$
- FFT: $t(p) \sim \log_2(p)$
- Naïve N-body: $t(p) \sim p$
- Samplesort: $t(p) \sim p^2$
Survey result: performance model normal form

\[ f(p) = \sum_{k=1}^{n} c_k \times p^{i_k} \times \log^j_{2} (p) \]

\( n = 1 \)
\( I = \{0, 1, 2\} \)
\( J = \{0, 1\} \)

- \( c_1 \times p \)
- \( c_1 \times p \times \log(p) \)
- \( c_1 \times p^2 \)
- \( c_1 \times p^2 \times \log(p) \)

A. Calotoiu, T. Hoefler, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes, SC13
Survey result: performance model normal form

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

- \( n = 2 \)
- \( I = \{0, 1, 2\} \)
- \( J = \{0, 1\} \)

\[
\begin{align*}
\text{for } n = 2, I = \{0, 1, 2\}, J = \{0, 1\}: \\
&c_1 + c_2 \times p \\
&c_1 + c_2 \times p^2 \\
&c_1 + c_2 \times \log(p) \\
&c_1 + c_2 \times p \times \log(p) \\
&c_1 + c_2 \times p^2 \times \log(p)
\end{align*}
\]

\[
\begin{align*}
&c_1 \cdot \log(p) + c_2 \cdot p \\
&c_1 \cdot \log(p) + c_2 \cdot p \cdot \log(p) \\
&c_1 \cdot \log(p) + c_2 \cdot p^2 \\
&c_1 \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p) \\
&c_1 \cdot p + c_2 \cdot p \cdot \log(p) \\
&c_1 \cdot p + c_2 \cdot p^2 \\
&c_1 \cdot p + c_2 \cdot p^2 \cdot \log(p) \\
&c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \\
&c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p) \\
&c_1 \cdot p^2 + c_2 \cdot p^2 \cdot \log(p)
\end{align*}
\]
Our automated generation workflow

- Statistical quality assurance
- Performance measurements
- Performance profiles
- Model generation
- Scaling models
- Performance extrapolation
- Ranking of kernels
- Model refinement

A. Calotoiu, T. Hoefler, M. Poke, F. Wolf: Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes, SC13

Come to our SC15 tutorial!
Model refinement

Input data

Hypothesis generation; hypothesis size $n$

Hypothesis evaluation via cross-validation

Computation of $R^2$ for best hypothesis

No

$n = 1; R_0 = \ldots$

Yes

$n = n_{\text{max}}$

Scaling model

\begin{align*}
\{(p_1, t_1), \ldots, (p_6, t_6)\} \\
\{c_1, c_1 \times \log(p)\} \\
\{c_1 \times p, c_1 \times p \times \log(p)\} \\
\{c_1 \times p^2, c_1 \times p^2 \times \log(p)\} \\
c_1 \times \log(p)
\end{align*}

\[
R^2 = 1 - \frac{\text{residualSumSquares}}{\text{totalSumSquares}}
\]

\[
\bar{R}^2 = 1 - (1 - R^2) \cdot \frac{n}{6 - n - 1}
\]

$I = \{0, 1, 2\}; J = \{0, 1\}; n_{\text{max}} = 2$
Evaluation overview

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

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

\[ n = 5 \]
HOMME

- 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>( 0.026 + 2.53 \times 10^{-6} p \sqrt{p} + 1.24 \times 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>

\( p_i \ 15k \)
HOMME

- 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>$3.63 \times 10^{-6} p \sqrt{p} + 7.21 \times 10^{-13} p^3$</td>
<td>30.34</td>
</tr>
<tr>
<td>vlaplace_sphere_vk</td>
<td>$24.44 + 2.26 \times 10^{-7} p^2$</td>
<td>4.28</td>
</tr>
<tr>
<td>compute_and_apply_rhs</td>
<td></td>
<td>49.09</td>
</tr>
</tbody>
</table>

$p_i \ 43k$
It works, great! Or not?

- We face several problems:
  - Multiple models – when did we collect enough data?
    
    \[
    \text{if}(np < 1.000) \ a(); \ \text{else} \ b();
    \]
  - Multiparameter modeling – search space explosion
    
    Interesting instance of the curse of dimensionality
  - Modeling overheads – traces do not scale
    
    Cross validation (leave-one-out) is slow and
    
    Our current profiling requires a lot of storage (>TBs)
First step: simple compiler analyses

- **Automatic kernel detection in LLVM**
  - Loop call graph – each loop/function as kernel (recursively)
  - Determine relevant input parameters for each kernel
    
    Massive pruning possible!

- **Online model generation (using online LASSO)**

  Constant (little) amount of data stored

- **Automatic profiling rate limiting**

  Profile less as models gain confidence

Quality: NAS UA and Mantevo MiniFE

Overhead: Mantevo

A. Bhattacharyya, TH: PEMOGEN: Automatic Adaptive Performance Modeling during Program Runtime, PACT’14
Second step: counting loop iterations

\[
\text{for } (j = 1; \ j \leq n; \ j = j \times 2) \\
\quad \text{for } (k = j; \ k \leq n; \ k = k++) \\
\quad \text{veryComplicatedOperation}(j,k);
\]

Polyhedral model

\[
N = (n + 1) \log_2 n - n + 2
\]
Counting arbitrary affine loop nests

- **Affine loops**

  ```plaintext
  x = x_0; // Initial assignment
  while (c^T x < g) // Loop guard
     x = Ax + b; // Loop update
  ```

- **Perfectly nested affine loops**

  ```plaintext
  while (c_1^T x < g_1) {
     x = A_1 x + b_1;
     while (c_2^T x < g_2) {
        ...
        x = A_{k-1} x + b_{k-1};
        while (c_k^T x < g_k) {
           x = A_k x + b_k;
           while (c_{k+1}^T x < g_{k+1}) { ... }  
           x = U_k x + v_k; }
        x = U_{k-1} x + v_{k-1};    
    ...
    x = U_1 x + v_1;
  }
  ```

  \[ A_k, U_k \in \mathbb{R}^{m \times m}, \ b_k, v_k, c_k \in \mathbb{R}^m, \ g_k \in \mathbb{R} \text{ and } k = 1 \ldots r. \]
Counting arbitrary affine loop nests

- Example

```c
for (j=1; j < n/p + 1; j= j*2)
    for (k=j; k < m; k = k + j )
        veryComplicatedOperation(j,k);
```
Counting arbitrary affine loop nests

- Example

```plaintext
for (j=1; j < n/p + 1; j= j*2) 
    for (k=j; k < m; k = k + j )
        veryComplicatedOperation(j,k);
```

```plaintext
while(c_1^T x < g_1) {
    x = A_1 x + b_1;
    while(c_2^T x < g_2) {
        ...
        x = A_{k-1} x + b_{k-1};
        while(c_k^T x < g_k) {
            x = A_k x + b_k;
            while(c_{k+1}^T x < g_{k+1}) {
                ...
                x = U_k x + v_k; }
            x = U_{k-1} x + v_{k-1};
        ...
    x = U_1 x + v_1;}
```
Counting arbitrary affine loop nests

- Example

```plaintext
for (j=1; j < n/p + 1; j= j*2)
    for (k=j; k < m; k = k + j )
        veryComplicatedOperation(j,k);

\[
\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix};
\]
```

```plaintext
while(c_1^T x < g_1) {
    x = A_1 x + b_1;
    while(c_2^T x < g_2) {
        ...
        x = A_{k-1} x + b_{k-1};
        while(c_k^T x < g_k) {
            x = A_k x + b_k;
            while(c_{k+1}^T x < g_{k+1}) {
                ...
            }
            x = U_k x + v_k;  }
        x = U_{k-1} x + v_{k-1};
    ...
} 
    x = U_1 x + v_1;}
```
Counting arbitrary affine loop nests

- Example

```
for (j=1; j < n/p + 1; j= j*2)
  for (k=j; k < m; k = k + j)
    veryComplicatedOperation(j,k);

while(c^T_1 x < g_1) {
  x = A_1 x + b_1;
  while(c^T_2 x < g_2) {
    ...
    x = A_{k-1} x + b_{k-1};
    while(c^T_k x < g_k) {
      x = A_k x + b_k;
      while(c^T_{k+1} x < g_{k+1}) {... }
      x = U_k x + v_k; }
    x = U_{k-1} x + v_{k-1};
    ...
  }
  x = U_1 x + v_1;

  while((1 0)^T \begin{pmatrix} j \\ k \end{pmatrix} < n/p + 1){
```
Counting arbitrary affine loop nests

Example

\[
\begin{align*}
&\text{for } (j=1; j < n/p + 1; j= j*2) \\
&\quad \text{for } (k=j; k < m; k = k + j ) \\
&\quad \text{veryComplicatedOperation}(j,k);
\end{align*}
\]

\[
\begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix};
\]

while \((1 \ 0) \begin{pmatrix} j \\ k \end{pmatrix} < \frac{n}{p} + 1)\{
\begin{align*}
&\quad \begin{pmatrix} j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \\
&\quad \begin{pmatrix} j \\ k \end{pmatrix} < m}\{
\end{align*}
\}

while \((0 \ 1) \begin{pmatrix} j \\ k \end{pmatrix} < m)\{
\}

T. Hoefler, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, SPAA’14
Counting arbitrary affine loop nests

- Example

```plaintext
for (j=1; j < n/p + 1; j = j*2)  
    for (k=j; k < m; k = k + j )  
        veryComplicatedOperation(j,k);
```

```plaintext
while(c^T_1 x < g_1) {  
    x = A_1 x + b_1;
    while(c^T_2 x < g_2) {  
        ...  
        x = A_{k-1} x + b_{k-1};  
    }
    while(c^T_k x < g_k) {  
        x = A_k x + b_k;  
        while(c^T_{k+1} x < g_{k+1}) {... }  
        x = U_k x + v_k;  
    }
    x = U_{k-1} x + v_{k-1};  
    ...}  
    x = U_1 x + v_1;}
```

\[
\begin{align*}
\begin{pmatrix} j \\ k \end{pmatrix} &= \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \\
while(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} < \frac{n}{p} + 1) \{ \\
\begin{pmatrix} j \\ k \end{pmatrix} &= \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \\
while(\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} < m) \{ \\
\begin{pmatrix} j \\ k \end{pmatrix} &= \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \\
\} \begin{pmatrix} j \\ k \end{pmatrix} &= \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}; \}
\end{align*}
\]
Counting arbitrary affine loop nests

- Example

```c
for (j=1; j < n/p + 1; j= j*2)
    for (k=j; k < m; k = k + j )
        veryComplicatedOperation(j,k);
```

```latex
\begin{align*}
    x &= \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} x + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\
    \text{while}(1 \ 0)x &< \frac{n}{p} + 1 \{ \\
    x &= \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\
    \text{while}(0 \ 1)x &< m \{ \\
    x &= \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\
    } \}x &= \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\
}\end{align*}
```

where

```
x = \begin{pmatrix} j \\ k \end{pmatrix}
```
Overview of the whole system

Parallel program

```
for i = 1, procCols
    call mpi_init( buff, .., dp_type, reduce_send_proc(1),
                   i, mpi_comm_world, request, ierr )
    call mpi_send( buff2, .., dp_type, reduce_send_proc(1),
                   i, mpi_comm_world, ierr )
    call mpi_wait( request, status, ierr )
enddo
for i = id * n/p, ( id + 1) * n/p
    do j = 1, nSize
        call compute
    enddo
```

Closed form representation

\[ x(i_1, ..., i_r) = A_{\text{final}}(i_1, ..., i_r) \cdot x_0 + b_{\text{final}}(i_1, ..., i_r) \]

with

\[ i_r = 0, n_r(x_{0,k}), k = 1, ..., r \]

Affine loop synthesis

```
while(c_1 x < g_1)
    x = A_1 x + b_1;
while(c_2 x < g_2)
    x = A_2 x + b_2;
... 
while(c_{k+1} x < g_{k+1})
    x = A_{k+1} x + b_{k+1};
... 
    x = U_k x + v_k; 
    x = U_{k-1} x + v_{k-1};
... 
    x = U_1 x + v_1;
```

Loop extraction

Program analysis

\[ W = N|_{p=1} \]
\[ D = N|_{p \to \infty} \]
Case study: NAS EP

\[ N(m, p) = \left[ \frac{2^m - 16 \cdot (u + 2^{16})}{p} \right] \]

15 applications (NAS/Mantevo/Mibench):
- 100% of loops were treated (with unknowns)
- 9-45% of loops were predicted exact

Arithmetic mean: 18%, median: 18%
What problems are remaining?

- Well, what about non-affine loops?
  - More general abstract interpretation (next step)
  - Not decidable → will always have undefined terms

- Back to PMNF?
  - Generalize to multiple input parameters
    a) Bigger search-space
    b) Bigger trace files

- Combine static (loop counting) and dynamic approach (PMNF)
  - Find number of loop iterations, replace \( u_x \) with \( \text{PMNF}_x \)
  - Find similar kernels (use only one PMNF for similar ones)
  - Remove irrelevant input parameters for each PMNF
  - Other simple optimizations: batch model update, etc.
  - Result: higher accuracy, lower overheads

\[
N = \frac{na \cdot u}{nprows}
\]

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

A. Bhattacharyya, G. Kwasniewski, TH: Using Compiler Techniques to Improve Automatic Performance Modeling, PACT'15
Performance Analysis 2.0 – Automatic Models

A call for action: use performance modeling for rigorous designs
- Especially for co-design (systems and applications)!
  *New architectures, e.g., FPGA assessment*
- High-performance programming as a science
  *Learn from natural sciences!*
- Start with the students: teach rigorous analysis and modeling


A. Bhattacharyya, T. Hoefler: PEMOGEN: Automatic Adaptive Performance Modeling during Program Runtime, *PACT 2014*
