TORSTEN HOEFLER

Theory and practice in HPC: Modeling, Programming, and Networking
High **Performance** Computing Practice

dgemm("N", "N", 50, 50, 50, 1.0, A, 50, B, 50, 1.0, C, 50);

**Performance is nondeterministic and not modular**

Performance of complex systems is tricky
HPC is used to solve complex problems!

Treat performance-centric programming and system design like physical systems

Image credit: Serena Donnin, Sarah Rauscher, Ivo Kabashow
Scientific **Performance** Engineering

1) Observe

2) Model

3) Understand

4) Build
Part I: Observe

- Measure systems
- Collect data
- Examine documentation
- Gather statistics
- Document process
- Experimental design
- Factorial design
Observe the state of the art in performance measurement

- Stratified random sample of three top HPC conferences for four years
  10 random papers from each (10-50% of population)
  120 total papers, 20% (25) did not report performance (were excluded)

Performance results are often nearly impossible to reproduce! Thus, we need to provide enough information to allow scientists to understand the experiment, draw own conclusions, assess their certainty, and possibly generalize results.
The latency of Piz Dora is 1.77us!

I averaged $10^6$ runs, it must be right!

How did you get this number?

Why do you think so? Can I see the data?

~1.2ms

~1.77us
Dealing with variation

The 99.9% confidence interval is 1.765us to 1.775us

Ugs, the data is not normal at all. The **nonparametric** 99.9% CI is much wider: 1.6us to 1.9us!

Did you assume normality?

Can we test for normality?
Looking at the data in detail

This CI makes me nervous. Let’s check!

Clearly, the mean/median are not sufficient!

Try quantile regression!

Image credit: nersc.gov
Scientific benchmarking of parallel computing systems

Twelve ways to tell the masses when reporting performance results

Torsten Hoefler
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
htor@inf.ethz.ch

Roberto Belli
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
bellr@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 reproduce experiments is one of the main principles of the scientific method. It is well known that the performance of a computer program depends on the application, the input, the compiler, the runtime environment, the machine, and the measurement methodology [20, 43]. If a single one of these aspects of experimental design is not appropriately motivated and described, presented results can hardly be reproduced and may even be misleading or incorrect.

The complexity and uniqueness of many supercomputers makes reproducibility a hard task. For example, it is practically impossible to recreate most hero-runs that utilize the world’s largest machines because these machines are often unique and their software configurations changes regularly. We introduce the notion of interpretability, which is weaker than reproducibility. We call an experiment interpretable if someone who looks at the data and not the code can understand the results. We describe guidelines for reproducibility that are based on interpretability and analysis tools that are needed to improve interpretability.
Simplifying Measuring and Reporting: LibSciBench

- Simple MPI-like C/C+ interface
- High-resolution timers
- Flexible data collection
- Controlled by environment variables
- Tested up to 512k ranks
- Parallel timer synchronization
- R scripts for data analysis and visualization

```c
#include <mpi.h>
#include <liblsb.h>
#include <stdlib.h>

#define N 1024
#define RUNS 10

int main(int argc, char *argv[]){
    int i, j, rank, buffer[N];

    MPI_Init(&argc, &argv);
    LSB_Init("test_bcast", 0);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* Output the info (i.e., rank, runs) in the results file */
    LSB_Set_Rparam_int("rank", rank);
    LSB_Set_Rparam_int("runs", RUNS);

    for (sz=1; sz<N; sz+=2){
        for (j=0; j<RUNS; j++){
            /* Reset the counters */
            LSB_Res();

            /* Perform the operation */
            MPI_Bcast(buffer, sz, MPI_INT, 0, MPI_COMM_WORLD);

            /* Register the j-th measurement of sz size */
            LSB_Record(sz);
        }
    }

    LSB_Finalize();
    MPI_Finalize();
    return 0;
}
```
We have the (statistically sound) data, now what?

The 99% confidence interval is within 1% of the reported median.

Matrix Multiply

$t(n) = a \cdot n^3$

The 99% confidence interval is within 1% of the reported median.
We have the (statistically sound) data, now what?

The 99% confidence interval is within 1% of the reported median.
The adjusted $R^2$ of the model fit is 0.99

Performance Modeling = Performance Analysis v 2.0

(time the next logical step)
Part II: Model

Burnham, Anderson: “A model is a simplification or approximation of reality and hence will not reflect all of reality. ... Box noted that “all models are wrong, but some are useful.” While a model can never be “truth,” a model might be ranked from very useful, to useful, to somewhat useful to, finally, essentially useless.”

This is generally true for all kinds of modeling. We focus on performance modeling in the following!
Performance Modeling

TH: Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC
Requirements modeling I: Six-step performance modeling

1. Input parameters
2. Describe application kernels
3. Communication parameters
4. Fit sequential baseline
5. Communication / computation overlap
6. Communication pattern

10-20% speedup [2]

Requirements modeling II: Automated best-fit modeling

- Manual kernel selection and hypothesis generation is time consuming (boring and tricky)
- Idea: Automatically select best (scalability) model from predefined search space

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

Number of processes

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

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

\[ c_1 \times \log(p) \]
\[ c_1 \times p \]
\[ c_1 \times p^2 \times \log(p) \]

Requirements modeling II: Automated best-fit modeling

- Manual kernel selection and hypothesis generation is time consuming (and boring)
- Idea: Automatically select best model from predefined space

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

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

Tool support: Extra-P for automated best-fit modeling [1]

Tutorial: **Insighful Automatic Performance Modeling**
A. Calotoiu, F. Wolf, TH, M. Schulz

Sunday, November 13th
1:30pm - 5pm


Requirements modeling III: Source-code analysis [1]

- Extra-P selects model based on best fit to the data
  - What if the data is not sufficient or too noisy?
- Back to first principles
  - The source code describes all possible executions
  - Describing all possibilities is too expensive, focus on counting loop iterations symbolically

\[
N = (n+1) \log_2 n - n + 2
\]
TH: Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC

Performance Modeling

Capability Model

Systems Expertise

Requirements Model

Application Expertise
TH: Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC
Capability models for network communication

The LogP model family and the LogGOPS model [1]

Finding LogGOPS parameters

Netgauge [2], model from first principles, fit to data using special kernels

Large scale LogGOPS Simulation

LogGOPSim [1], simulates LogGOPS with 10 million MPI ranks

<5% error

Capability models for cache-to-cache communication


Invalid read $R_I = 278\,\text{ns}$
Local read: $R_L = 8.6\,\text{ns}$
Remote read $R_R = 235\,\text{ns}$
Performance Model

TH: Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC
Part III: Understand

- **Use models to**
  1. Proof optimality of real implementations
     - *Stop optimizing, step back to algorithm level*
  2. Design optimal algorithms or systems in the model
     - *Can lead to non-intuitive designs*

- **Proof optimality of matrix multiplication**
  - Intuition: flop rate is the bottleneck
  - \( t(n) = 76\text{ps} * n^3 \)
  - **Flop rate:** \( R = 2\text{flop} * n^3/(76\text{ps} * n^3) = 27.78 \text{ Gflop/s} \)
  - **Flop peak:** 3.864 GHz * 8 flops = 30.912 Gflop/s
    Achieved ~90% of peak (IBM Power 7 IH @3.864GHz)

- **Gets more complex quickly**
  - Imagine sparse matrix-vector
2) Design optimal algorithms – small broadcast in LogP

L=2, o=1, P=7

Design algorithms – bcast in cache-to-cache model

Multi-ary tree example

depth \( d = 2 \)

\( k_1 = 2 \)

\( k_2 = 3 \)

\( \mathcal{T}_{\text{tree}} = \sum_{i=1}^{d} \mathcal{T}_C(k_i) = \sum_{i=1}^{d} (c \cdot k_i + b) = \sum_{i=1}^{d} (R_R + R_L + c \cdot (k_i - 1)) \)

\( \mathcal{T}_{\text{bcast}} = \min_{d, k_i} \left( \mathcal{T}_f + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} \mathcal{T}_n(k_i + 1) \right) \)

\( N \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j, \quad \forall i < j, k_i \leq k_j \)

Measured results – small broadcast and reduction

- Broadcast
- Min-Max Model
- Intel MPI (icc)
- Reduction
- Min-Max Model
- OpenMP (icc)

Intel Xeon Phi 5110P (60 cores at 1052 MHz), Intel MPI v.4.1.4

- Each operation timed separately, reporting maximum across processes

Performance Modeling

Capability Model

Performance Model

Requirements Model

TH: Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC
Part IV: Build

Abstraction is Key

- Enables to focus on essential aspects of a system

Case study: Network Topologies

- **Observe**: optimize for cost, maintain performance:
  - router radix, number of cables, number of routers $\rightarrow$ cost
  - number of endpoints, latency, global bandwidth $\rightarrow$ capabilities
- **Model**: system as graph
- **Understand**: degree-diameter graphs
- **Build**: Slim Fly topology
- **Result**: non-trivial topology that outperforms all existing ones

How to continue from here?

### Transformation System

- User-supported, compile- and run-time

```latex
\text{memlets} + \text{operators} = \text{DCIR}
```

### Parallel Language

- Data-centric, explicit requirements models

### Performance-transparent Platforms

- HTM [1]
- MPI RMA
- foMPI-NA [2]
- NISA [3]

---

[1]: M. Besta, TH: Accelerating Irregular Computations with Hardware Transactional Memory and Active Messages, ACM HPDC'15
Backup