Theory and practice in HPC: Modeling, Programming, and Networking
2⁴ years ago ....
High **Performance** Cluster Computing

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
- Experimental design
- Collect data
- Examine documentation
- Gather statistics
- Document process
- 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!

How did you get this number?

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

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

~1.77us

~1.2ms
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!
Scientific benchmarking of parallel computing systems

ACM/IEEE Supercomputing 2015 (SC15)

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
belli@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

Reproducing 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 it can be repeated with the same or similar results; reproducible if it can be repeated with the exact same results.
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_Rec(sz);
        }
    }
    LSB_Finalize();
    MPI_Finalize();
    return 0;
}
```

S. Di Girolamo, TH: http://spcl.inf.ethz.ch/Research/Performance/LibLSB/
We have the (statistically sound) data, now what?

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

Matrix Multiply
\[ t(n) = a \times n^3 \]

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

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

- Input parameters
- Describe application kernels
- Communication parameters
- Fit sequential baseline
- Communication / computation overlap
- 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

\[
f(p) = \sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot \log_2^{j_k}(p)
\]

Number of processes

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

\begin{align*}
  c_1 & \quad c_1 \times \log(p) \\
  c_1 \times p & \quad c_1 \times p \times \log(p) \\
  c_1 \times p^2 & \quad c_1 \times p^2 \times \log(p)
\end{align*}

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

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

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

\[ f(p) = c_1 + c_2 \times p \]
\[ f(p) = c_1 + c_2 \times p^2 \]
\[ f(p) = c_1 + c_2 \times \log(p) \]
\[ f(p) = c_1 + c_2 \times p \times \log(p) \]
\[ f(p) = c_1 + c_2 \times p^2 \times \log(p) \]

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

Talk: Fast Multi-Parameter Performance Modeling
A. Calotoiu, et al.

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

Sunday, November 13th
1:30pm - 5pm
Room 355-C

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

```
for (j = 1; j <= n; j = j*2)
for (k = j; k <= n; k = k++)
  OperationInBody(j,k);
```

\[ N = (n+1) \log_2 n - n + 2 \]

Parallel program

```
do i = 1, procCols;
call mpi_send(buff, i, mp_type, reduce_each_proc[i],
  i, mpi_comm_world, request, ierr);
call mpi_send(buff0, i, mp_type, reduce_each_proc[i],
  i, mpi_comm_world, ierr);
call mpi_wait(request, status, ierr);
enddo
```

Loop extraction

```
Number of iterations

\[ N = \sum_{i_1=0}^{n_1(x_0,1)} \sum_{i_2=0}^{n_2(x_0,2)} \cdots \sum_{i_r=0}^{n_r(x_0,r-1)} n_r(x_0,r) \]

Requirements Models

\[ W = N \big|_{p=1} \]

\[ D = N \big|_{p \to \infty} \]

[1]: TH, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, ACM SPAA’14
Performance Modeling

Capability Model

Requirements Model

Application Expertise

TH: Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC
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

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
  - \( t(n) = 76 \text{ps} \times n^3 \)
  - Flop rate \( R = 2 \text{flop} \times n^3 / (76 \text{ps} \times n^3) = 27.78 \text{ Gflop/s} \)
  - Flop peak: 3.864 GHz \times 8 \text{ flops} = 30.912 \text{ Gflop/s}
    - Achieved \( \sim 90\% \text{ of peak (IBM Power 7 IH @3.864GHz)} \)

- 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

\[
\begin{align*}
\text{depth } d &= 2 \\
k_1 &= 2 \\
k_2 &= 3
\end{align*}
\]

Tree depth

\[
T_{\text{tree}} = \sum_{i=1}^{d} T_C(k_i) = \sum_{i=1}^{d} (c \cdot k_i + b)
\]

Level size

\[
= \sum_{i=1}^{d} (R_R + R_L + c \cdot (k_i - 1))
\]

Tree cost

\[
T_{\text{bcast}} = \min_{d, k_i} \left( T_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} T_{nb}(k_i + 1) \right)
\]

Reached threads

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

Measured results – small broadcast and reduction

Intel Xeon Phi 5110P (60 cores at 1052 MHz), Intel MPI v.4.1.4 – each operation timed separately, reporting maximum across processes

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
A BRIEF HISTORY OF NETWORK TOPOLOGIES

- Mesh
- Butterfly
- Clos/Benes
- Kautz
- Hypercube
- Fat Trees
- Torus
- Trees
- Dragonfly
- Slim Fly
- Flat Fly
- Random

1980’s - 2000’s - ~2005

- copper cables, small radix switches
- fiber, high-radix switches

2007 - 2008 - 2014

????
A BRIEF HISTORY OF NETWORK TOPOLOGIES

- Mesh
- Torus
- Butterfly
- Clos/Benes
- Kautz
- Dragonfly
- Slim Fly
- Hypercube
- Trees
- Fat Trees
- Flat Fly
- Random

Bandwidth = \(2 \sqrt[N]{d-1}\)
Latency = \(\frac{d}{2} \sqrt[N]{d}\)
Radix = \(2d\)

1980s: Copper cables, small radix switches
2000s: Fiber, high-radix switches
~2005: Transition period
2007: 2008: Future developments
2014: ???

35
A BRIEF HISTORY OF NETWORK TOPOLOGIES

- **Mesh**
  - 1980's
  - Copper cables, small radix switches

- **Butterfly**
  - 2000's
  - Bandwidth $= \frac{N}{2}$
  - Latency $= 2 \log_2 N$
  - Radix $= 4$

- **Torus**
  - ~2005

- **Clos/Benes**
  - 2000's

- **Kautz**
  - ~2005

- **Dragonfly**
  - 2008

- **Slim Fly**
  - 2008

- **Hypercube**
  - 2001

- **Fat Trees**
  - 2008

- **Trees**
  - 2014

- **Flat Fly**
  - 2014

- **Random**
  - 2014
A BRIEF HISTORY OF NETWORK TOPOLOGIES

Bandwidth \( \approx \frac{N}{4} \)
Latency \( = 3 - 5 \)
Radix \( = 48 - 64 \)

1980's

- Copper cables, small radix switches
- Fiber, high-radix switches

2007
- 2008
- 2008
- 2014

???

John Kim
Northwestern University
Evanston, IL 60208
johnkim@northwestern.edu

Abstract

The PERCS system was designed by IBM in response to a HPC’s challenge that called for a high-performance interconnect. A key innovation in the PERCS design is the network that is built using high-radix switches that are integrated into the compute nodes. Each high-radix switch is about 580 mm in size, has over 2750 signal I/Os, and is packaged in a module that also contains LGA-attached optical devices.

The PERCS high-performance interconnect is a modular implementation of high-radix switches, with multiplexed links, that is proven to be effective in high-performance computing environments. The module contains a high-radix internal crossbar switch and is connected to external links with a high-performance interconnect. The module is designed to be compatible with existing high-radix switches and is expected to reduce the cost and complexity of high-radix switch implementations.
A BRIEF HISTORY OF NETWORK TOPOLOGIES

Key insight:

“It’s the diameter, stupid”

Lower diameter:
→ Fewer cables traversed
→ Fewer cables needed
→ Fewer routers needed

Cost and energy savings:
→ Up to 50% over Fat Tree
→ Up to 33% over Dragonfly

Bandwidth \( \approx \frac{N}{4} \)
Latency \( = 2 - 4 \)
Radix \( = k \)


**Key method**

**Optimize towards the Moore Bound [1]:** the upper bound on the number of vertices in a graph with given diameter $D$ and radix $k$.

$$MB(D, k) = 1 + k + k(k - 1) + k(k - 1)^2 + \cdots$$

$$MB(D, k) = 1 + k \sum_{i=0}^{D-1} (k - 1)^i$$

DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

Example Slim Fly design for diameter = 2: MMS graphs [1]

DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

Groups form a fully-connected bipartite graph
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

1. Select a prime power $q$
   
   $q = 4w + \delta$;
   
   $w \in \mathbb{N}$  $\delta \in \{-1,0,1\}$,
   
   A Slim Fly based on $q$ :
   
   Number of routers: $2q^2$
   
   Network radix: $(3q - \delta)/2$

2. Construct a finite field $\mathbb{F}_q$.
   
   Assuming $q$ is prime:
   
   $\mathbb{F}_q = \mathbb{Z}/q\mathbb{Z} = \{0,1,\ldots,q-1\}$
   
   with modular arithmetic.

E. Example: $q = 5$

   50 routers
   
   network radix: 7

   $\mathbb{F}_5 = \{0,1,2,3,4\}$
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

3 Label the routers

Set of routers:
\( \{0,1\} \times \mathcal{F}_q \times \mathcal{F}_q \)

Example: \( q = 5 \)

...
DESIGNING AN EFFICIENT NETWORK TOPOLOGY
CONNECTING ROUTERS: DIAMETER 2

4 Find primitive element $\xi$
$\xi \in \mathcal{F}_q$ generates $\mathcal{F}_q$:
All non-zero elements of $\mathcal{F}_q$ can be written as $\xi^i$, $i \in \mathbb{N}$

5 Build Generator Sets
$X = \{1, \xi^2, \ldots, \xi^{q-3}\}$
$X' = \{\xi, \xi^3, \ldots, \xi^{q-2}\}$

E Example: $q = 5$
$\mathcal{F}_5 = \{0, 1, 2, 3, 4\}$
$\xi = 2$
$1 = \xi^4 \mod 5 = 2^4 \mod 5 = 16 \mod 5$
$X = \{1, 4\}$
$X' = \{2, 3\}$
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

**Intra-group connections**

Two routers in one group are connected iff their "vertical Manhattan distance" is an element from:

- \( X = \{1, \xi^2, \ldots, \xi^{q-3}\} \) (for subgraph 0)
- \( X' = \{\xi, \xi^3, \ldots, \xi^{q-2}\} \) (for subgraph 1)

**Example:** \( q = 5 \)

Take Routers \((0,0,\ldots)\)

\[ X = \{14\} \]
DESIGNING AN EFFICIENT NETWORK TOPOLOGY

CONNECTING ROUTERS: DIAMETER 2

Intra-group connections

Two routers in one group are connected iff their “vertical Manhattan distance” is an element from:

\[ X = \{1, \xi^2, \ldots, \xi^{q-3}\} \quad \text{(for subgraph 0)} \]
\[ X' = \{\xi, \xi^3, \ldots, \xi^{q-2}\} \quad \text{(for subgraph 1)} \]

Example: \( q = 5 \)

Take Routers (1,4,.)

\[ X' = \{2,3\} \]
Designing an Efficient Network Topology

Connecting Routers: Diameter 2

7 Inter-group connections

Router \((0, x, y) \leftrightarrow (1, m, c)\)

iff \(y = mx + c\)

Example: \(q = 5\)

- Take Router \((1,0,0)\)
  - \(m = 0, c = 0\)
  - \((1,0,0) \leftrightarrow (0, x, 0)\)

- Take Router \((1,1,0)\)
  - \(m = 1, c = 0\)
  - \((1,0,0) \leftrightarrow (0, x, x)\)
**Cost Comparison**

![Graph showing cost comparison across different network topologies.](image)

- **Topologies**:
  - Long Hop
  - Hypercube
  - Torus 5D
  - Fat Tree
  - Torus 3D
  - Random Topology
  - Flat. Butterfly
  - Dragonfly
  - Slim Fly

- **Legend**
  - Total cost in millions of $ vs. Number of endpoints (in thousands).

A LOWEST-DIAMETER TOPOLOGY

- Approaching the Moore Bound
- Resilient

A COST & POWER EFFECTIVE TOPOLOGY

- 25% less expensive than Dragonfly,
- 26% less power-hungry than Dragonfly

A HIGH-PERFORMANCE TOPOLOGY

- Lowest latency
- Full global bandwidth
How to continue from here?

Transformation System

- User-supported, compile- and run-time

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
Advancing with Scientific Performance Engineering

1) Observe

2) Model

Online Data Reduction

- Collect nonparametric statistics online
- Reduce to small number of parameters

3) Understand

Optimal Collectives
- Large broadcast and many others, unclear in LogGOPS even LogGP [1]

4) Build
Advancing with Scientific Performance Engineering

1) Observe

2) Model

Model Accuracy Tradeoffs

- Modeling vs. Simulation [1]

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Cycle-accurate simulation</th>
<th>Model-based simulation</th>
<th>Analytical Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Parameters/Complexity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Model Error

3) Understand

- Optimal Collectives
  - Large broadcast ... and many others, unclear in LogGOPS even LogGP [1]

4) Build
Advancing with Scientific Performance Engineering

1) Observe

2) Model

3) Understand

4) Build

Optimal Collectives

- Large broadcast ... and many others, unclear in LogGOPS, even LogGP [1]
Advancing with Scientific Performance Engineering

1) Observe

2) Model

- Model Accuracy Tradeoffs
  - Benchmark Cycle - accurate simulation
  - Analytical Model

3) Understand

- Model vs. Simulation [1]
- Number of Parameters/Complexity
- Model Error

4) Build

- Collect nonparametric statistics online
- Reduce to small number of parameters

Modular Low-Diameter Topologies

- How to combine smaller pieces into a single cheap topology?
Advancing with Scientific Performance Engineering

1) Observe

2) Model

3) Understand

4) Build

Modeling vs. Simulation

Model Accuracy Tradeoffs

Benchmark Cycle - accurate simulation

Model - based simulation

Analytical Model

Number of Parameters/Complexity

Model Error

Collect nonparametric statistics online

Reduce to small number of parameters

Online Data Reduction

Large broadcast, … and many others, unclear in LogGOPS, even LogGP [1]

Optimal Collectives

How to combine smaller pieces into a single cheap topology?

Modular Low-Diameter Topologies
Backup