Developing high-performance software – from modeling to programming.

Torsten Hoefler
Department of Computer Science
ETH Zurich

Invited talk at Multicore@Siemens conference, Nuremberg, Feb. 2018
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute

$\Delta x = 35 \text{ m} \ (1x)$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute.

$\Delta x = 70 \text{ m} \ (10x)$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute

$\Delta x = 140 \text{ m (100x)}$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute.

$\Delta x = 280 \text{ m (1,000x)}$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute.

$\Delta x = 550 \text{ m} \ (10,000x)$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute

Operational model of MeteoSwiss today!

$\Delta x = 1100 \text{ m (100,000x)}$
MeteoSwiss New Weather Supercomputer

World's First GPU-Accelerated Weather Forecasting System

2x Racks
48 CPUs
192 Tesla K80 GPUs
> 90% of FLOPS from GPUs
Operational in 2016

A factor of $2^x$ in resolution roughly corresponds to a factor of $10^x$ to compute.

We're a factor of $100,000$ away!

Breakthrough Advance in Swiss Weather Forecasting

BEFORE GPUS

24-Hour Forecasts
2.2km Resolution
8 Simulations per Day

Medium Range Forecasts
3 Day Forecasts
6.6km Resolution
3 Simulations per Day

AFTER GPUS

24-Hour Forecasts
1.1km Resolution (2x Higher)
8 Simulations per Day

Medium Range Forecasts
5 Day Forecasts (2 Days Longer)
2.2km Resolution (3x Higher)
42 Simulations per Day (14x More)

Image credit: Oliver Fuhrer, MeteoSwiss
Changing hardware constraints and the physics of computing

Three Ls of modern computing:
- Spatial Locality
- Temporal Locality
- Control Locality

Transistors (thousands)
Parallel Performance
Sequential Performance

Moore’s law really is dead this time

Frequency (MHz)
Typical Power (Watts)
Number of Cores

Data partially collected: M. Horowitz, F. Laborie, G. Ghasham, K. Choukroun, K. Hammond

Electrical data movement loss ~ resistivity \times \text{length} \times \text{cross-section area}

- 0.9 V [1]
- 32-bit FP ADD: 0.9 pJ
- 32-bit FP MUL: 3.2 pJ
- 2x32 bit from L1 (8 KiB): 10 pJ
- 2x32 bit from L2 (1 MiB): 100 pJ
- 2x32 bit from DRAM: 1.3 nJ

[1]: Marc Horowitz, Computing’s Energy Problem (and what we can do about it), ISSC 2014, plenary
Load-store vs. Dataflow architectures

Load-store ("von Neumann")

\[ x = a + b \]

Energy per instruction: 70pJ

Static Dataflow ("non von Neumann")

\[ y = (a + b) \cdot (c + d) \]

Energy per operation: 1-3pJ

Turing Award 1977 (Backus): "Surely there must be a less primitive way of making big changes in the store than pushing vast numbers of words back and forth through the von Neumann bottleneck."

Source: Mark Horowitz, ISSC‘14
Single Instruction Multiple Data/Threads (SIMD - Vector CPU, SIMT - GPU)

(High Performance) Computing really became a data management challenge

[1]: Marc Horowitz, Computing's Energy Problem (and what we can do about it), ISSC 2014, plenary
But memory architectures are becoming more and more complex

Xeon Phi KNL: 3 memory models, 5 configuration modes each \( \rightarrow \) 15 options for configuring the system!
How do we optimize codes for these complex architectures?

- **Performance engineering**: “encompasses the set of roles, skills, activities, practices, tools, and deliverables applied at every phase of the systems development life cycle which ensures that a solution will be designed, implemented, and operationally supported to meet the non-functional requirements for performance (such as throughput, latency, or memory usage).”

- **Manually profile codes and tune** them to the given architecture
  - Requires highly-skilled performance engineers
  - Need familiarity with
    - NUMA (topology, bandwidths etc.)
    - Caches (associativity, sizes etc.)
    - Microarchitecture (number of outstanding loads etc.)
    …
An engineering example – Tacoma Narrows Bridge
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
The latency of Piz Dora is 1.77us!

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

~1.77us

How did you get this number?

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

~1.2ms

Example: Simple ping-pong latency benchmark
Dealing with variation

The 99.9% confidence interval is 1.765us to 1.775us

Did you assume normality?

Can we test for normality?

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

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
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 its design is well motivated and described and if the results are meaningful without the complete source code or input data.

Rule 1
When publishing parallel speedup, report if the base case is a single parallel process or best serial execution, as well as the absolute execution performance of the base case.

Rule 2
Specify the reason for only reporting subsets of standard benchmarks or applications or not using all system resources.

Rule 3
Use the arithmetic mean only for summarizing costs. Use the harmonic mean for summarizing rates.

Rule 4
Avoid summarizing ratios; summarize the costs or rates that the ratios base on instead. Only if these are not available use the geometric mean for summarizing ratios.

Rule 5
Report if the measurement values are deterministic. For nondeterministic data, report confidence intervals of the measurement.

Rule 6
Do not assume normality of collected data (e.g., based on the number of samples) without diagnostic checking.

Rule 7
Carefully investigate if measures of central tendency such as mean or median are useful to report. Some problems, such as worst-case latency, may require other percentiles.

Rule 8
Carefully investigate if measures of central tendency such as mean or median are useful to report. Some problems, such as worst-case latency, may require other percentiles.

Rule 9
Document all varying factors and their levels as well as the complete experimental setup (e.g., software, hardware, techniques) to facilitate reproducibility and provide interpretability.

Rule 10
For parallel time measurements, report all measurement, (optional) synchronization, and summarization techniques.

Rule 11
If possible, show upper performance bounds to facilitate interpretability of the measured results.

Rule 12
Plot as much information as needed to interpret the experimental results. Only connect measurements by lines if they indicate trends and the interpolation is valid.
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 sz */
            LSB_Rec(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
\[ t(n) = a \cdot n^3 \]

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.
The adjusted $R^2$ of the model fit is 0.99
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

Capability Model

Performance Model

Systems Expertise

Requirements Model

Application Expertise

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

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

Number of processes

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

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

\[ I, J \subset Q \]

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

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

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

\[ c_1 \cdot \log(p) + c_2 \cdot p \]
\[ c_1 \cdot \log(p) + c_2 \cdot p \cdot \log(p) \]
\[ c_1 \cdot \log(p) + c_2 \cdot p^2 \]
\[ c_1 \cdot \log(p) + c_2 \cdot p \cdot \log(p) \]
\[ c_1 \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p) \]
\[ c_1 \cdot p + c_2 \cdot p \cdot \log(p) \]
\[ c_1 \cdot p + c_2 \cdot p^2 \]
\[ c_1 \cdot p + c_2 \cdot p^2 \cdot \log(p) \]
\[ c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \]
\[ c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p) \]
\[ c_1 \cdot p^2 + c_2 \cdot p^2 \cdot \log(p) \]

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

```plaintext
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**

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

**Loop extraction**

**Number of iterations**

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

---

[1]: TH, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, ACM SPAA'14
TH: Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC

Performance Modeling

Capability Model

Requirements Model

Application Expertise

- Describe application kernels
- Communication pattern
- Communication parameters
- Fit sequential baseline

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

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

A Practical Model of Parallel Computation

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

Performance Model
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: \( R = \frac{2 \text{flop} \times n^3}{(76 \text{ps} \times n^3)} = 27.78 \text{ Gflop/s} \)
  - Flop peak: \( 3.864 \text{ GHz} \times 8 \text{ flops} = 30.912 \text{ Gflop/s} \)
    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 $d = 2$

$K_1 = 2$

$K_2 = 3$

Tree depth

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

Level size

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

Tree cost

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

Reached threads

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

Measured results – small broadcast and reduction

Performance Modeling

Capability Model

Performance Model

Requirements Model

Application Expertise

Input parameters

Describe application kernels

Communication pattern

Communication/computation overlap

Fit sequential baseline

Systems Expertise

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 → cost
  - number of endpoints, latency, global bandwidth → capabilities
- **Model**: system as graph
- **Understand**: degree-diameter graphs
- **Build**: Slim Fly topology
- Result: non-trivial topology that is 1/3rd cheaper than all existing

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
DAPPy – Data-centric Parallel Programming for Python

- Memory access decoupled from computation

- Programs are composed of Tasklets and Memlets
  - Tasklets wrapped by simple primitives: Map, Iterate, Reduce
  - Hide communication, caching and data-movement

- Easy-to-integrate Python programming interface

- Graph-based compilation pipeline

```python
@dapp.program
def gemm(A, B, C):
    # local definitions
    @dapp.map(_[0:M, 0:K, 0:N])
def multiplication(i, j, k):
in_A << A[i,k]
in_B << B[k,j]
out >> tmp[i,j,k]

out = in_A * in_B

@dapp.reduce(tmp, C, axis=2)
def sum(a,b): return a+b
```
DAPPy Compilation Infrastructure

**Code**

- `@dapp_program
  def program(A, B):
    @dapp_map(_[B:N,0:M])
    def transpose(i, j):
      a << A[i,j]
      b >> B[j,i]
    ...

dappy Program

- Domain Programmer

**Specialization**

- DAPP Framework
  - Custom Patterns
  - Custom Models
  - Subgraph Matching
  - Performance Models
  - Stateful Dataflow Graph (SDFG)

**Runtime**

- Partitioning, Scheduling
- System Probe
- Microbenchmarks

- CPU Library
- GPU Library
- FPGA Modules

- Performance Engineer

- Expert
Performance

SDFG

Naïve
Performance
Performance

SDFG

LoopReorder
MapReduceFusion

Naïve
Performance

SDFG

BlockTiling
LoopReorder
MapReduceFusion

Naïve
Performance

SDFG

RegisterTiling
BlockTiling
LoopReorder
MapReduceFusion
Naïve
Performance

SDFG

LocalStorage
RegisterTiling
BlockTiling
LoopReorder
MapReduceFusion
Naïve
Performance

- PromoteTransient
- LocalStorage
- RegisterTiling
- BlockTiling
- LoopReorder
- MapReduceFusion

SDFG
Performance

![Graph showing performance comparison between Intel MKL, DAPP, and OpenBLAS. The graph indicates a 25% difference between Intel MKL and DAPP.]
Generated DAPP/C++ Code (Excerpt)

```c
void program_gemm(  
    int sym_0, int sym_1, int sym_2, double __restrict__ A, double __restrict__ B, double __restrict__ C) {  
    // State s0  
    for (int tile_k = 0; tile_k < sym_2; tile_k += 128) {  
      #pragma omp parallel for  
      for (int tile_i = 0; tile_i < sym_0; tile_i += 64) {  
        for (int tile_j = 0; tile_j < sym_1; tile_j += 240) {  
          for (int regtile_j = 0; regtile_j < (min(240, sym_1 - tile_j)); regtile_j += 12) {  
            vec<double, 4> local_B_s0_0[128 * 3];  
            Global2Stack_2D_FixedWidth<double, 4, 3>(&B[tile_k*sym_1 + (regtile_j + tile_j)], sym_1,  
                local_B_s0_0, min(sym_2 - tile_k, 128));  
          }  
          for (int regtile_i = 0; regtile_i < (min(64, sym_0 - tile_i)); regtile_i += 4) {  
            vec<double, 4> regtile_C_s0_1[4 * 3];  
            for (int i = 0; i < 4; i += 1) {  
              for (int j = 0; j < 3; j += 1) {  
                double in_A = A[(i + regtile_i + tile_i)*sym_2 + tile_k];  
                vec<double, 4> in_B = local_B_s0_0[0*3 + j];  
                // Tasklet code (mult)  
                auto out = (in_A * in_B);  
                regtile_C_s0_1[i*3 + j] = out;  
              }  
            }  
          }  
        }  
      }  
    }  
  }  
```

Backup