Model-guided performance engineering of numerical kernels

Georg Hager
Erlangen Regional Computing Center (RRZE)
University of Erlangen-Nuremberg
Erlangen, Germany

Bergische Universität Wuppertal
10.7.2015
First of all: How to see more of it

- Regular PRACE tutorials (two-day)
  - December @ LRZ Garching
  - July @ HLRS Stuttgart

- SC Conference tutorial (one-day)
  - Next: November 15, 2015, Austin, TX, USA

- SPPEXA collaboration activities
  - Next: September 29+30, 2015, TU Darmstadt

- ISC15 full-day workshop **Performance Modeling: Methods and Applications**, July 16, 2015, Frankfurt

- … and probably some others
An example from physics

Newtonian mechanics

\[ \vec{F} = m\vec{a} \]

Fails @ small scales!

Nonrelativistic quantum mechanics

\[ \frac{\partial}{\partial t} \psi(\vec{r}, t) = H \psi(\vec{r}, t) \]

Fails @ even smaller scales!

Relativistic quantum field theory

\[ U(1)_Y \otimes SU(2)_L \otimes SU(3)_c \]
"Black box" vs. "white box" models

input data

black box

output data

modeling framework: statistics, fitting, machine learning,...

predictions

model OK?

N

Y

insight

adjust model → insight

input data

white box

simplified description of system

modeling

predictions

validation

model OK?

N

Y

insight
## Overview of Performance Modeling

<table>
<thead>
<tr>
<th>Performance modeling</th>
<th>“Brand X”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relies on understanding of true application behavior</td>
<td>Relies on pattern matching and curve fitting (extrapolation &amp; interpolation)</td>
</tr>
<tr>
<td>White-box approach (application-centric)</td>
<td>Black-box approach (application-oblivious)</td>
</tr>
<tr>
<td><em>Explains</em> performance; detail of explanation correlates with accuracy of prediction</td>
<td>Predicts without providing insight or gauges of accuracy</td>
</tr>
<tr>
<td>Disagreements with measurements challenge assumptions and yield new insights</td>
<td>Disagreements with measurements merely showcase limitations of approach</td>
</tr>
</tbody>
</table>
White box performance engineering

Set up an (analytical) model for a given algorithm/kernel/solver/application on a given architecture

Compare with measurements to validate the model

(Hopefully) identify optimization opportunities and start over
Examples for analytical (“white box”) models

Amdahl’s Law with communication

\[ S(N) = \frac{1}{s + \frac{1-s}{N} + c(N)} \]

Hockney model for message transmission time

\[ T_{PtP} = T_l + \frac{L}{B} \]

Roofline model for loop code execution time

\[ T_{exec} = \max(T_{calc}, T_{data}) \]

ECM model for loop code execution time

\[ T_{exec} = \max(T_{nOL} + T_{data}, T_{OL}) \]

Program speedup

Serial fraction

Latency

Message length

Bandwidth

Time for computation

Time for data transfer

Non-overlapping execution

Time for data transfer

Overlapping execution

July 10, 2015

Performance Engineering
Case study: Sparse Matrix Vector Multiplication on a multicore CPU

Part 1: Basics and observations
Sparse Matrix Vector Multiplication (spMVM)

- Key ingredient in some matrix diagonalization algorithms
  - Lanczos, Davidson, Jacobi-Davidson

- **Store only** $N_{nz}$ **nonzero elements** of matrix and RHS, LHS vectors with $N_r$ (number of matrix rows) entries
- **“Sparse”**: $N_{nz} \sim N_r$

General case: some indirect addressing required!
Popular storage format: CRS scheme

“Compressed Row Storage”

- \texttt{val[]} stores all the nonzeros (length \(N_{nz}\))
- \texttt{col_idx[]} stores the column index of each nonzero (length \(N_{nz}\))
- \texttt{row_ptr[]} stores the starting index of each new row in \texttt{val[]} (length: \(N_r\))
Case study: Sparse matrix-vector multiply

OpenMP-parallel loop kernel

```c
!$OMP parallel do
do  i = 1,N_r
   do  j = row_ptr(i), row_ptr(i+1) - 1
      c(i) = c(i) + val(j) * b(col_idx(j))
   enddo
endo
do i = 1,N_r
do  j = row_ptr(i), row_ptr(i+1) - 1
   c(i) = c(i) + val(j) * b(col_idx(j))
endo
!$OMP end parallel do
```

Usually many spMVMs required to solve a problem
Performance characteristics on Intel Sandy Bridge CPU

- Performance does not scale across the cores on a CPU chip
- Performance seems to depend on the matrix

- Can we explain this?
- Is there a “light speed” for spMVM?
- Optimization?
“White box” performance modeling on the chip level: Roofline

DOI: 10.1016/0743-7315(88)90002-0


Prelude: Modeling customer dispatch in a bank

Rovolving door throughput:
\[ b_s \text{ [customers/sec]} \]

Intensity:
\[ I \text{ [tasks/customer]} \]

Processing capability:
\[ P_{\text{max}} \text{ [tasks/sec]} \]
Prelude: Modeling customer dispatch in a bank

How fast can tasks be processed? \( P \) [tasks/sec]

The bottleneck is either

- The service desks (max. tasks/sec): \( P_{\text{max}} \)
- The revolving door (max. customers/sec): \( I \cdot b_S \)

\[
P = \min(P_{\text{max}}, I \cdot b_S)
\]

This is the “Roofline Model”

- High intensity: \( P \) limited by “execution”
- Low intensity: \( P \) limited by “bottleneck”
- “Knee” at \( P_{\text{max}} = I \cdot b_S \):
  Best use of resources

Roofline is an “optimistic” model (“light speed”)
The Roofline Model: Performance modeling of loops

1. \( P_{\text{max}} \) = Applicable peak performance of a loop, assuming that data comes from the level 1 cache (this is not necessarily \( P_{\text{peak}} \))
   \[ \Rightarrow \text{e.g., } P_{\text{max}} = 176 \text{ GFlop/s} \]

2. \( I \) = Computational intensity (“work” per byte transferred) over the slowest data path utilized (code balance \( B_C = I^{-1} \))
   \[ \Rightarrow \text{e.g., } I = 0.167 \text{ Flop/Byte} \Rightarrow B_C = 6 \text{ Byte/Flop} \]

3. \( b_S \) = Applicable peak bandwidth of the slowest data path utilized
   \[ \Rightarrow \text{e.g., } b_S = 56 \text{ GByte/s} \]

Expected performance:

\[
P = \min(P_{\text{max}}, I \cdot b_S) = \min\left(P_{\text{max}}, \frac{b_S}{B_C}\right)
\]
Example: Dense matrix-vector multiplication in double precision on an Intel Haswell processor

\[
\begin{align*}
\text{do } & i=1,N \\
\text{do } & j=1,N \\
& \quad c(i) = c(i) + A(j,i) \times b(j) \\
\text{enddo} \\
\text{enddo}
\end{align*}
\]

- Assume \( N \approx 5000 \)
- Applicable peak performance?
- Relevant data path?
- Computational Intensity?

\[
\begin{align*}
\text{do } & i=1,N \\
& \quad \text{tmp} = c(i) \\
\text{do } & j=1,N \\
& \quad \text{tmp} = \text{tmp} + A(j,i) \times b(j) \\
\text{enddo} \\
& \quad c(i) = \text{tmp} \\
\text{enddo}
\end{align*}
\]

\( \rightarrow \) does not fit in cache

\( \rightarrow \) half peak (2 LD, 1 ADD, 1 MULT)

\( \rightarrow \) main memory

\( \rightarrow \) 2 flops / 8 bytes = 0.25 F/B
(b() comes from cache)
Example: Dense matrix-vector multiplication in double precision on an Intel Haswell processor

- Haswell CPU hardware characteristics
  - 14 cores
  - 2.5 MByte of cache per core
  - 8 ADDs + 8 MULTs per core per clock cycle
  - Clock speed = 2.3 GHz
    \[ P_{\text{max}} = \frac{1}{2} \times P_{\text{peak}} = \frac{1}{2} \times 14 \times 16 \times 2.3 \text{ GFlop/s} = 258 \text{ GFlop/s} \]
  - Max. memory bandwidth (measured) \( b_S = 56 \text{ GByte/s} \)

- Roofline model: \( P = \min \left( \frac{258 \text{ GFlop}}{s}, 0.25 \frac{\text{Flop}}{\text{Byte}} \times 56 \frac{\text{GByte}}{s} \right) = 14 \text{ GFlop/s} \)
  - Code is memory bound
  - 2.7% of peak performance!

- What can you do to improve it? If the code runs at 14 Gflop/s, nothing.
Roofline Model assumptions ("machine model")

- There is a clear concept of "work" vs. "traffic"
  - "work" = flops, updates, iterations…
  - "traffic" = required data to do "work"
- No latency effects → perfect streaming mode
- One data transfer bottleneck is modeled only; all others are assumed to be infinitely fast

- **Data transfer and core execution overlap perfectly!**
  - This is the main problem in situations where Roofline does not work!
  - Remedy: Execution-Cache-Memory (ECM) model


Case study:
Sparse Matrix Vector Multiplication on a multicore CPU

Part 2: Modeling
Example: SpMVM node performance model

- Sparse MVM in double precision w/ CRS data storage:

  do i = 1, Nr
  do j = row_ptr(i), row_ptr(i+1) - 1
    \[ C(i) = C(i) + \text{val}(j) \times \text{B[col_idx(j)]} \]
  enddo
  enddo

- Computational intensity
  - \( \alpha \) quantifies traffic for loading RHS
    - \( \alpha = 1/N_{nzr} \rightarrow \) RHS loaded once
    - \( \alpha = 1 \rightarrow \) no cache
    - \( \alpha > 1 \rightarrow \) Houston, we have a problem!
  - “Expected” performance = \( b_S \times I_{CRS} \)
  - Determine \( \alpha \) by measuring performance and actual memory traffic
    - Maximum memory BW may not be achieved with spMVM

\[ I_{CRS}^{DP} = \frac{2}{8 + 4 + 8\alpha + 16/N_{nzr}} \text{flops/byte} \]

For large \( N_{nzr}: \)
- \( I_{max} \approx \frac{2 \text{Flops}}{12 \text{Byte}} \)
- \( B_{min} \approx 6 \text{Byte/Flop} \)
Determine RHS traffic

\[ I_{CRS}^{DP} = \frac{2}{8 + 4 + 8\alpha + 16/N_{nzr}} \text{flops} = \frac{N_{nz} \cdot 2 \text{flops}}{V_{meas}} \]

- \( V_{meas} \) is the measured overall memory data traffic (using, e.g., likwid-perfctr)
- Solve for \( \alpha \):
  \[
  \alpha = \frac{1}{4} \left( \frac{V_{meas}}{N_{nz} \cdot 2 \text{ bytes}} - 6 - \frac{8}{N_{nzr}} \right)
  \]

- Example: kkt_power matrix from the UoF collection on one Intel SNB socket
  - \( N_{nz} = 14.6 \cdot 10^6, N_{nzr} = 7.1 \)
  - \( V_{meas} \approx 258 \text{ MB} \)
  - \( \Rightarrow \alpha = 0.43, \alpha N_{nzr} = 3.1 \)
  - \( \Rightarrow \) RHS is loaded 3.1 times from memory
  - and:
    \[
    \frac{I_{CRS}^{DP}(1/N_{nzr})}{I_{CRS}^{DP}(\alpha)} = 1.15
    \]
    \( 15\% \text{ extra traffic} \Rightarrow \text{optimization potential!} \)
Now back to the start…

Hardware & software:

- \( b_S = 39 \text{ GB/s} \)
- \( B_{c \text{min}} = 6 \text{ B/F} \)

Maximum spMVM performance:

- \( P = 6.5 \text{ GF/s} \)

\( \rightarrow \) DLR1 causes minimum code balance!

sAMG matrix code balance:

- \( B_c \leq \frac{b_S}{4.5 \text{ GF/s}} = 8.7 \text{ B/F} \)

Now what next?

- Matrix reordering may improve balance \( \rightarrow \) faster code
- Check other performance-limiting factors (load imbalance, non-streaming)
- Saturation effect cannot be explained by Roofline (in a satisfying way)
Sparse matrix testcases

“DLR1” (A. Basermann, DLR)
Adjoint problem computation (turbulent transonic flow over a wing) with the TAU CFD system of the German Aerospace Center (DLR)
Avg. non-zeros/row ~150

“sAMG” (K. Stüben, FhG-SCAI)
Matrix from FhG’s adaptive multigrid code sAMG for the irregular discretization of a Poisson problem on a car geometry.
Avg. non-zeros/row ~ 7
Conclusion from Roofline analysis

- The roofline model can only deliver an optimistic absolute upper limit for spMVM due to the RHS traffic uncertainties.
- We have “turned the model around” and measured the actual memory traffic to determine the RHS overhead.
- Result indicates:
  1. how much actual traffic the RHS generates
  2. how efficient the RHS access is (compare BW with max. BW)
  3. how much optimization potential we have with matrix reordering

Consequence: Modeling is not always 100% predictive. It’s all about learning more about performance properties!
Typical code optimizations in the Roofline Model

1. Hit the BW bottleneck by good serial code
   (e.g., Perl → Fortran)
2. Increase intensity to make better use of BW bottleneck
   (e.g., loop blocking [see later])
3. Increase intensity and go from memory-bound to core-bound
   (e.g., temporal blocking)
4. Hit the core bottleneck by good serial code
   (e.g., -fno-alias [see later])
5. Shift $P_{\text{max}}$ by accessing additional hardware features or using a different algorithm/implementation
   (e.g., scalar → SIMD)
Best practices for benchmarking and optimization
Basics of benchmarking & optimization

Motivation:
- **Understand** observed performance
- **Learn** about code characteristics and machine capabilities
- Deliberately **decide** on optimizations

Process:
1. Define relevant **test cases** (similar to production runs)
2. Establish a sensible **performance metric** (work/time)
3. Acquire a **runtime profile** (where does the time go?)
4. Identify “hot” loop kernels (hopefully there are any!)
5. Try to build a simple **performance model**
6. **Optimize** the kernel if possible

iterate
Best practices for benchmarking

Preparation
- Care for **reliable timing** (minimum time which can be measured?)
- Document code generation (flags, compiler version)
- Get access to an **exclusive** system
- System state (clock speed, turbo mode, memory, caches)
- Consider automating runs with a script (shell, python, perl)

Doing
- **Affinity control**
- Check: Is the result reasonable? (300 PFlop/s are not)
- Is result **deterministic and reproducible**? (if not, search for reasons)
- Statistics: Mean, Best?
- Basic variants to check
  - Thread count, affinity, **working set size**
Thank you.

http://blogs.fau.de/hager