Performance and Power Engineering on Multicore Processors

Georg Hager, Jan Treibig, Gerhard Wellein
Erlangen Regional Computing Center (RRZE)
University of Erlangen-Nuremberg
Erlangen, Germany

GRS-Sim Aachen
March 11, 2013
Outline

- Performance Modeling and Engineering
  - Motivation
  - “White Box” models: Roofline

- Example: Sparse MVM

- “If the model doesn’t work, we learn something”
  - A starting point for refining Roofline

- The ECM multi-core model

- A simple power model for multicore
An example from physics

Newtonian mechanics

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

Fails @ small scales!

Nonrelativistic quantum mechanics

\[ \frac{\partial}{\partial t} \psi(r, t) = i\hbar \left( \nabla \cdot \psi(r, t) - H \psi(r, t) \right) \]

Fails @ even smaller scales!

Relativistic quantum field theory

\[ U(1)_Y \otimes SU(2)_L \otimes SU(3)_c \]
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
The Performance Engineering (PE) process

- Runtime profiling
- Algorithm/Code analysis
- Machine characteristics
- Traces/HW metrics
- Code optimization
- Microbenchmarking
- Performance model
“White Box” Performance Models on the Chip Level

Roofline model
ECM model
An example: The Roofline Model$^{1,2}$

1. $P_{\text{max}} = $ Applicable peak performance of a loop, assuming that data comes from L1 cache

2. $I = $ Computational intensity ("work" per byte transferred) over the slowest data path utilized ("the bottleneck")

3. $b_S = $ Applicable peak bandwidth of the slowest data path utilized

Expected performance:

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


A simple Roofline example

Example: 

\[
\text{do } i=1,N; \ s=s+a(i); \ \text{enddo}
\]

in double precision on hypothetical 3 GHz CPU, 4-way SIMD, N large

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

Performance and Power Engineering

ADD peak (half of full peak)

4-cycle latency per ADD if not unrolled

no SIMD

Computational intensity
Roofline Model assumptions

- 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
- Attainable bandwidth of code = input parameter!
  - Microbenchmarging may be required

- Data transfer and core execution overlap perfectly!
- “Applicable peak” can be calculated accurately
- Bottleneck is modeled only; all others are assumed to be infinitely fast

- If data transfer is the limiting factor, the bandwidth of the slowest data path can be utilized to 100% (“saturation”)
Using Roofline in a More Complex Setting

Sparse matrix-vector multiply (spMVM)
Example: SpMVM node performance model

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

\[
\text{do } i = 1, N_{r}
\]
\[
\text{do } j = \text{row}_\text{ptr}(i), \text{row}_\text{ptr}(i+1) - 1
\]
\[
C(i) = C(i) + \text{val}(j) \times \text{B(col_idx(j))}
\]
\text{enddo}
\text{enddo}

- DP CRS comp. intensity
  - \(\kappa\) quantifies extra traffic for loading RHS more than once

\[
I_{\text{CRS}} = \frac{2}{12 + \frac{24}{N_{nzr}} + \kappa} \frac{\text{Flops}}{\text{Byte}}
\]
\[
= \left(6 + \frac{12}{N_{nzr}} + \frac{\kappa}{2}\right)^{-1} \frac{\text{Flops}}{\text{Byte}}
\]

- Predicted Performance = \text{streamBW} \cdot I_{\text{CRS}}

- Determine \(\kappa\) by measuring performance and actual memory bandwidth
Test matrices: Sparsity patterns

- Analysis for HMeP matrix on Nehalem EP socket
  - BW used by spMVM kernel = 18.1 GB/s → should get ≈ 2.66 Gflop/s
  - spMVM performance if $\kappa = 0$
  - Measured spMVM performance = 2.25 Gflop/s
  - Solve $2.25 \text{ Gflop/s} = \text{BW} \cdot I_{\text{CRS}}$ for $\kappa \approx 2.5$

→ 37.5 extra bytes per row
→ RHS is loaded 6 times from memory
→ about 33% of BW goes into RHS

- Conclusion: Even if the roofline model does not work 100%, we can still learn something from the deviations
“If the model fails, we learn something”

In-core analysis of the Schönauer triad on Sandy Bridge
Example: Schönauer Vector Triad in L2 cache

- **REPEAT** [\[A(:) = B(:) + C(:) * D(:)\]] @ double precision
- Analysis for Sandy Bridge core w/ AVX (unit of work: 1 cache line)

Machine characteristics:

**Registers**

\[ \text{L1} \]

- 1 LD/cy + 0.5 ST/cy
- 32 B/cy (2 cy/CL)

**Arithmetic**

- 1 ADD/cy+ 1 MULT/cy

**L2**

**Triad analysis (per CL):**

\[ \text{L1} \]

- 6 cy/CL
- 10 cy/CL

**Arithmetic**

- AVX: 2 cy/CL
- SSE: 4 cy/CL

**Timeline:**

- 16 F/CL (AVX)

**Roofline prediction:** 16/10 F/cy

**Measurement:** 16F / ≈17cy
Schönauer Vector Triad in L2 cache

- No overlap of evict/refill with LD/ST in L1
  - L1 is “single ported”
- Other cache levels similar?
- How about overlap further down the hierarchy?
  - May be possible to get lower/upper performance bounds

→ Model for single-core execution with data from all levels of the hierarchy!
An Improved Performance Model for Multicore

The ECM Model
Roofline sometimes fails for multicore

- Assumes one of two bottlenecks
  1. In-core execution
  2. Bandwidth of a single hierarchy level

- Saturation effects in multicore chips are not explained

\[ A(\cdot) = B(\cdot) + C(\cdot) \times D(\cdot) \]

Roofline predicts full socket BW
The multicore saturation mystery

- Why is serial performance “too low?”
  - Non-overlapping contributions from data transfers and in-cache execution to overall runtime

- What determines the saturation point?
  - Important for energy efficiency
  - Putting cores to better use
  - Saturation == Bandwidth pressure on relevant bottleneck exhausts the maximum BW capacity

- Requirements for an appropriate multicore performance model
  - Should predict single-core performance
  - Should predict saturation point

→ ECM (Execution – Cache – Memory) model
Example: ECM model for Schönauer Vector Triad
\[ A(\cdot) = B(\cdot) + C(\cdot) \times D(\cdot) \] on a Sandy Bridge Core with AVX

Per-cycle transfer widths

256 bit LD
& 128 bit ST

256 bit

256 bit

107 bit
(@ 2.7 GHz)

Registers

\[ \text{max}(2(B) + 2(C) + 2(D), 4(A)) \text{ cy} = 6 \text{ cy} \]

\[ (2(B) + 2(C) + 2(D) + 4(A)) \text{ cy} = 10 \text{ cy} \]

\[ (2(B) + 2(C) + 2(D) + 4(A)) \text{ cy} = 10 \text{ cy} \]

\[ (5 \times 64 \text{ B} \times 2.7 \text{ Gcy/s}) / (36 \text{ GB/s}) = 24 \text{ cy} \]

Achievable full-socket BW \( (b_s) \)
Results suggest no overlap!
Multicore scaling in the ECM model

- Identify relevant **bandwidth bottlenecks**
  - L3 cache
  - Memory interface
- **Scale** single-thread performance \( (P_0) \) until first bottleneck is hit:

\[
P(n_t) = \min(n_t P_0, P_{\text{roof}}), \quad \text{with} \quad P_{\text{roof}} = \min(P_{\text{max}}, I \cdot b_S)
\]

**Example:** Scalable L3 on Sandy Bridge
ECM prediction vs. measurements for $A(:)=B(:)+C(:)\times D(:)$ on a Sandy Bridge socket (no-overlap assumption)

Saturation point (# cores) well predicted

Measurement: scaling not perfect

Caveat: This is specific for this architecture and this benchmark!

Check: Use “overlappable” kernel code
ECM prediction vs. measurements for $A(\cdot)=B(\cdot)+C(\cdot)/D(\cdot)$ on a Sandy Bridge socket (full overlap assumption)

In-core execution is dominated by divide operation
(44 cycles with AVX, 22 scalar)

→ Almost perfect agreement with ECM model
Example: Lattice-Boltzmann flow solver

- D3Q19 model
- Empty channel, \(228^3\) fluid lattice sites (3.7 GB of memory)
- AVX implementation with compiler intrinsics

**ECM model input**
- Core execution from Intel IACA tool
- Max. memory bandwidth from multi-stream measurements

\[
\begin{align*}
\text{Registers} & \quad 432 \text{ cy (IACA throughput)} \\
\text{L1D} & \quad 3 \cdot 19 \cdot 2 \text{ cy} = 114 \text{ cy} \\
\text{L2} & \quad 3 \cdot 19 \cdot 2 \text{ cy} = 114 \text{ cy} \\
\text{L3} & \quad (3 \cdot 19 \cdot 64 \cdot 2.7 / 32.3) \text{ cy} = 305 \text{ cy (@ 2.7 GHz)} \\
& \quad \quad \text{or} \\
& \quad \quad (3 \cdot 19 \cdot 64 \cdot 1.6 / 30.6) \text{ cy} = 191 \text{ cy (@ 1.6 GHz)}
\end{align*}
\]

**BW \(b_s\)** degradation @ lower frequencies and large # of streams

3/11/2013 Performance and Power Engineering
Lattice-Boltzmann solver: ECM (no-overlap) vs. measurements

Saturation point again predicted accurately

Saturation performance matches multi-stream benchmarks (by construction)

No-overlap assumption seems a little pessimistic

Not all execution is LD and ST (IACA predicts ADD bottleneck)
Why the fuss about the saturation point?

(1) Putting cores to good use

(2) Energy consumption
A simple power model for the Sandy Bridge processor

A model for multicore chip power

- Goal: Establish model for chip power and program energy consumption with respect to
  - Clock speed
  - Number of cores used
  - Single-thread program performance

- Choose different characteristic benchmark applications to measure a chip’s power behavior
  - Matrix-matrix-multiply ("DGEMM"): “Hot” code, well scalable
  - Ray tracer: Sensitive to SMT execution (15% speedup), well scalable
  - 2D Jacobi solver: 4000x4000 grid, strong saturation on the chip
    - AVX variant
    - Scalar variant

- Measure characteristics of those apps and establish a power model
App scaling behavior (DGEMM omitted)
Chip power and performance vs. clock speed on full socket & single core

Sandy Bridge EP (8-core) processor:

---

(a) Power [W] vs. Frequency [GHz] with all cores used.
(b) Relative performance vs. Frequency [GHz] with single core ignored.
Chip power and cycles per instruction (CPI) vs. # of cores

Sandy Bridge EP (8-core) processor:

CPI and power correlated, but not proportional
A simple power model for multicore chips

Assumptions:

1. Power is a quadratic polynomial in the clock frequency
2. Dynamic power is linear in the number of active cores $t$
3. Performance is linear in the number of cores until it hits a bottleneck ($\leftarrow$ ECM model)
4. Performance is linear in the clock frequency unless it hits a bottleneck
5. Energy to solution is power dissipation divided by performance

Model:

$$E = \frac{W_0 + (W_1 f + W_2 f^2) t}{\min ((1 + \Delta \nu) t P_0, P_{\text{max}})}$$

where $f = (1 + \Delta \nu) f_0$
Model predictions

$$E = \frac{W_0 + (W_1 f + W_2 f^2) t}{\min \left((1 + \Delta v) t P_0, P_{\text{max}}\right)}$$

1. If there is no saturation, use all available cores to minimize $E$

![Graph showing performance vs. number of cores]

Minimum $E$ here

$$\frac{\partial E}{\partial t} = -\frac{W_0}{(1 + \Delta v) t^2 P_0} < 0$$
Model predictions

\[ E = \frac{W_0 + (W_1 f + W_2 f^2)t}{\min ((1 + \Delta v)tP_0, P_{\text{max}})} \]

2. There is an optimal frequency \( f_{\text{opt}} \) at which \( E \) is minimal in the non-saturated case, with

\[ f_{\text{opt}} = \sqrt{\frac{W_0}{W_2 t}}, \quad \text{hence it depends on the baseline power} \]

\( \rightarrow \) “Clock race to idle” if baseline power accommodates whole system!
\( \rightarrow \) If \( f_{\text{opt}} < f_0 \), may have to look at other metrics, e.g., \( C = \frac{E}{P} \)

\[ \frac{\partial C}{\partial \Delta v} = -\frac{2W_0 + W_1 ft}{(f/f_0)^3 P_0^2} < 0 \]
Model predictions

\[ E = \frac{W_0 + (W_1 f + W_2 f^2) t}{\min((1 + \Delta v) t P_0, P_{\text{max}})} \]

3. If there is saturation, \( E \) is minimal at the saturation point

Minimum \( E \) here

\[ t_s = \frac{P_{\text{max}}}{(1 + \Delta v) P_0} \]
Model predictions

\[
E = \frac{W_0 + (W_1 f + W_2 f^2) t}{\min((1 + \Delta v)tP_0, P_{\text{max}})}
\]

4. If there is saturation, absolute minimum \(E\) is reached if the saturation point is at the number of available cores

- Slower clock
  - \(\rightarrow\) more cores to saturation
  - \(\rightarrow\) smaller \(E\)
5. Making code execute faster on the core saves energy since
   - The time to solution is smaller if the code scales ("Code race to idle")
   - We can use fewer cores to reach saturation if there is a bottleneck

\[ E = \frac{W_0 + (W_1 f + W_2 f^2)t}{\min((1 + \Delta v)tP_0, P_{\text{max}})} \]

Better code
→ earlier saturation
→ smaller E @ saturation
Model validation with the benchmark apps

![Graphs showing energy consumption vs. frequency and number of cores.]

- **Figure (a)**: Energy to solution [J] vs. Frequency [GHz] for DGEMM 8C, DGEMM 4C, RAY 8C, and RAY SMT 8C.
  - Key Points: 1, 2, 5

- **Figure (b)**: Energy to solution [J] vs. # cores for Jacobi AVX 2.7 GHz, Jacobi AVX 2.0 GHz, and Jacobi AVX 1.4 GHz.
  - Key Points: 3

---

3/11/2013

Performance and Power Engineering
Need for a cost model!? 

\[ C = \frac{E}{P} \]

$\rightarrow$ Lower cost for slower clock speed (in this case)
Conclusions

- Performance Engineering == Performance Modeling with “bells and whistles”

- PE is a structured process which gives insight into the interaction of hardware and software

- Saturation effects are ubiquitous; understanding them gives us opportunity to
  - Find out about optimization opportunities
  - Put cores to good use
  - Save energy

- Possible extensions to the power model
  - Allow for per-core frequency setting (coming with Intel Haswell)
  - Accommodate load imbalance & sync overhead
Thank you.
References

