Holistic node-level performance engineering for maximum resource efficiency on modern multi-core CPUs

Georg Hager
Erlangen Regional Computing Center (RRZE)

ParisTech TELECOM
September 7, 2015
References


Further references

  DOI: 10.1142/S0129626410000296

  DOI: 10.1177/1094342012442424

  DOI: 10.1142/S0129626414410047

Analytical performance modeling:

“Constructing a simplified model for the interaction between software and hardware in order to understand lowest-order performance behavior”

Basic questions addressed by analytic performance models

- What is the bottleneck? → optimization technique
- What is the next bottleneck? → performance potential of the optimization
- Impact of processor frequency and socket scalability
  → Appropriate execution parameters, energy-optimized operating point

If the model fails, we learn something!
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 “classic” Roofline Model\(^1,2,3\)

1. \( P_{\text{max}} = \text{Applicable peak performance of a loop} \) (this is not necessarily \( P_{\text{peak}} \))

2. \( I = \text{Operational intensity (“work” per byte transferred) over the slowest data path utilized (“the bottleneck”) } \)

3. \( b_S = \text{Applicable peak bandwidth of the slowest data path utilized (hardware feature)} \) optimistic model (light speed)

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

---


Agenda

- ECM model
  - Basic rules, non-overlap
  - Notation
  - Saturation and comparison with Roofline
  - Case study 1: Kahan-enhanced scalar product
  - Case study 2: 3-D long-range stencil

- A simple power model for multicores
  - Case study: lattice-Boltzmann flow solver

- Summary
THE ECM MODEL

Registers

L1

L2

L3

MEM

The ECM Model

Throughput Analysis Report

<table>
<thead>
<tr>
<th>Buffers</th>
<th>Throughput</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>140.0</td>
<td>6.0</td>
</tr>
<tr>
<td>L2</td>
<td>240.0</td>
<td>12.0</td>
</tr>
<tr>
<td>L3</td>
<td>300.0</td>
<td>15.0</td>
</tr>
<tr>
<td>Mem</td>
<td>240.0</td>
<td>30.0</td>
</tr>
</tbody>
</table>

Graph showing MLUP/s vs. # cores
1. LOADs in the L1 cache do not overlap with any other data transfer in the memory hierarchy

2. Everything else in the core overlaps perfectly with data transfers (STOREs may show some non-overlap)

3. The scaling limit is set by the ratio of 

\[
\frac{\text{# cycles per CL overall}}{\text{# cycles per CL at the bottleneck}}
\]

Example:

Single-core (data in L1): 8 cy (ADD) 
Single-core (data in memory): 
6+9+9+19 cy = 43 cy 

Scaling limit: 43 / 19 = 2.3 cores
ECM model – composition

ECM predicted time

\[ T_{ECM} = \text{maximum of overlapping time and sum of all other contributions} \]

\[ T_{core} = \max(T_{nOL}, T_{OL}) \]

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

Shorthand notation for time contributions:

\[ \{ T_{OL} \parallel T_{nOL} \mid T_{L1L2} \mid T_{L2L3} \mid T_{L3Mem} \} \]

# cy invariant to clock speed

# cy changes w/ clock speed

Example from previous slide:

\[ \{ 8 \parallel 6 | 9 | 9 | 19 \} \text{ cy} \]
ECM model – prediction

Notation for cycle predictions in different memory hierarchy levels:

\[
\{ T_{ECM}^{L1} \mid T_{ECM}^{L2} \mid T_{ECM}^{L3} \mid T_{ECM}^{Mem} \}
\]

\[
T_{ECM}^{L1} = T_{core} = \max(T_{nOL}, T_{OL})
\]

\[
T_{ECM}^{L2} = \max(T_{nOL} + T_{L1L2}, T_{OL})
\]

\[
T_{ECM}^{L3} = \max(T_{nOL} + T_{L1L2} + T_{L2L3}, T_{OL})
\]

\[
T_{ECM}^{Mem} = \max(T_{nOL} + T_{L1L2} + T_{L2L3} + T_{L3Mem}, T_{OL})
\]

Example: \( \{ 8 \mid 15 \mid 24 \mid 43 \} \) cy

Experimental data (measured) notation: \( 8.6 \mid 16.2 \mid 26 \mid 47 \) cy
ECM model – from time to time to performance with varying clock speed

$f_0$: base clock speed [cy/s]
$f$: actual clock speed [cy/s]

Performance is work ($W$) over time:

$$P_{ECM} = \frac{W \cdot f}{\left\{ \frac{T_{ECM}^{L1}}{T_{ECM}^{L3}}, \frac{T_{ECM}^{L2}}{T_{ECM}^{L3}}, \frac{T_{ECM}^{L3}}{T_{ECM}^{L3}} \right\} \max\left( T_{ECM}^{L3} + T_{L3Mem} \cdot \frac{f}{f_0}, T_{OL} \right)}$$

- in-cache performance is proportional to $f$
- ratio $T_{ECM}^{L3}/T_{L3Mem}$ quantifies $f$-sensitivity of serial in-memory performance

Example:

$$P = \frac{32 \text{ flops} \cdot f}{\{8, 15, 24, 24 + 19 \cdot \frac{f}{f_0}\} \text{ cy}}$$
ECM model – saturation

Main assumption: Performance scaling is linear until a bandwidth bottleneck ($b_S$) is hit

Performance vs. cores (Memory BN):

$$P_{ECM}(n) = \min \left( nP_{ECM}^{Mem}, \frac{b_S^{Mem}}{B_C^{Mem}} \right)$$

Number of cores at saturation:

$$n_S = \left[ \frac{b_S/B_C}{P_{ECM}^{Mem}} \right] = \left[ \frac{T_{ECM}^{Mem}}{T_{L3Mem}} \right]$$

Example:

\[
\begin{align*}
\{8 \| 6 \| 9 \| 9 \| 19\} \text{cy},
\{8 \mid 15 \mid 24 \mid 43\} \text{cy} & \implies n_S = \left[ \frac{43}{19} \right] = 3
\end{align*}
\]
ECM vs. Roofline

Roofline assumes **full overlap** of all execution and transfer times

Roofline requires **measured baseline bandwidth limits** for all memory levels $i$ (L2…Memory) at all core counts $n$: $b_S^i(n)$

\[
\begin{align*}
T_{\text{Roof}}^{L1} &= T_{\text{core}} = \max(T_{nOL}, T_{OL}) \\
&\quad \vdots \\
T_{\text{Roof}}^{\text{Mem}} &= \max(T_{nOL}, T_{L1L2}, T_{L2L3}, T_{L3\text{Mem}}, T_{OL})
\end{align*}
\]

\[
P_{\text{Roof}}(n) = \min_i \left( nP_{ECM}^{L1}, \frac{b_S^i(n)}{B_C^i} \right)
\]

Roofline $\approx$ ECM if:

- $T_{\text{core}} \gg T_{\text{data}}$: non-overlapping data transfers are insignificant
- Loop kernel is similar to streaming benchmark used to obtain $b_S^i(n)$
How do we get the numbers?

Basic information about hardware capabilities:

- **In-core limitations**
  - Throughput limits: µops, LD/ST, ADD/MULT per cycle
  - Pipeline depths

- **Cache hierarchy**
  - **ECM**: Cycles per CL transfer
  - **RL**: measured max bandwidths for all cache levels, core counts

- **Memory interface**
  - **ECM**: measured saturated BW
  - **RL**: measured max bandwidths for all core counts

\[\begin{align*}
T_{core} & : \text{Code analysis, Intel IACA} \\
T_{L1L2}, T_{L2L3}, T_{L3Mem}, B_C^i & : \text{Data flow analysis}
\end{align*}\]
CASE STUDY: KAHAN-ENHANCED SCALAR PRODUCT

Is the Kahan scalar product harmful for performance?

Kahan-enhanced scalar product

Does it harm performance to augment the dot kernel in this way?
Is there a difference between single-threaded and multi-threaded?
SP vs. DP? Influence of architecture?
**ECM modeling of sdot on 10-core Ivy Bridge EP 2.2 GHz**

**Naive sdot (AVX):**

\[
\begin{array}{c}
\{2 \| 4 \| 4 \| 4 \} 6.1 + 2.9 \text{ cy} \\
\{4 \| 8 \| 12 \} 18.1 + 2.9 \text{ cy}
\end{array}
\]

- Latency penalty
- saturation at 4 cores

**Kahan sdot, scalar mode:**

\[
\begin{array}{c}
\{64 \| 16 \| 4 \| 4 \} 6.1 + 2.9 \text{ cy} \\
\{64 \| 64 \| 64 \| 64 \} \text{ cy}
\end{array}
\]

- saturation at 11 cores
Comparing optimal AVX implementations on four Intel architectures (SP)

<table>
<thead>
<tr>
<th></th>
<th>ECM model [cy]</th>
<th>Prediction [cy/CL]</th>
<th>Pred. performance [GUP/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNB</td>
<td>{8 \parallel 4\vert 4\vert 4\vert 7.9 + 5.1}</td>
<td>{8 \parallel 8\vert 12\vert 19.9 + 5.1}</td>
<td>{5.40 \parallel 5.40\vert 3.60\vert 1.73}</td>
</tr>
<tr>
<td>IVB</td>
<td>{8 \parallel 4\vert 4\vert 4\vert 6.1 + 2.9}</td>
<td>{8 \parallel 8\vert 12\vert 18.1 + 2.9}</td>
<td>{4.40 \parallel 4.40\vert 2.93\vert 1.68}</td>
</tr>
<tr>
<td>HSW</td>
<td>{8 \parallel 2\vert 2\vert 4\vert 4.9 + 4.5}</td>
<td>{8 \parallel 8\vert 8\vert 12.9 + 4.5}</td>
<td>{4.60 \parallel 4.60\vert 4.60\vert 2.11}</td>
</tr>
<tr>
<td>BDW</td>
<td>{8 \parallel 2\vert 2\vert 4\vert 7 + 1}</td>
<td>{8 \parallel 8\vert 8\vert 15 + 1}</td>
<td>{3.60 \parallel 3.60\vert 3.60\vert 1.8}</td>
</tr>
</tbody>
</table>

- Kahan without consequence if AVX is applied starting from L2 cache
- BDW latency penalty is rather low (== pure ECM model works well)
Saturation

- SP: Saturation possible if any kind of SIMD is applied
- DP: Saturates always

- Compiler is not able to generate decent code
#pragma omp parallel for
for(int k=4; k < N-4; k++) {
    for(int j=4; j < N-4; j++) {
        for(int i=4; i < N-4; i++) {
            float lap = c0 * V[k][j][i]
            + c1 * ( V[k][j][i+1] + V[k][j][i-1])
            + c1 * ( V[k][j+1][i] + V[k][j-1][i])
            + c1 * ( V[k+1][j][i] + V[k-1][j][i])
            ...
            + c4 * ( V[k][j][i+4] + V[k][j][i-4])
            + c4 * ( V[k][j+4][i] + V[k][j-4][i])
            + c4 * ( V[k+4][j][i] + V[k-4][j][i]);

            U[k][j][i] = 2.0 * V[k][j][i] - U[k][j][i]
            + ROC[k][j][i] * lap;
        }
    }
}
3D long-range SP stencil ECM model

Layer condition in L3 at problem size $N_i \times N_j \times N_k$:

$$9 \cdot N_i \cdot b_j \cdot n_{threads} \cdot 4 \cdot B < \frac{C_3}{2}$$

ECM Model: \{ 68 || 62 | 24 | 24 | 17 \} cy \rightarrow \{ 68 | 86 | 110 | 127 \} cy

Saturation at $n_s = \left\lfloor \frac{127}{17} \right\rfloor = 8$ cores.

Consequences:
- Temporal blocking will not yield substantial speedup
- Improve low-level code first (semi-stencil…?)
3D long-range SP stencil results (SNB)

Roofline too optimistic due to overlapping assumption
A simple power model for multicore CPUs
A simple power model for multicore chips

Model assumptions:

1. Power is a quadratic polynomial in the clock frequency: \( W = W_0 + w_1 f + w_2 f^2 \)

2. Dynamic power is linear in the number of active cores: \( W_{\text{dyn}} = (W_1 f + W_2 f^2)n \)

3. Performance is linear in the number of cores until it hits a bottleneck

4. Performance is linear in the clock frequency unless it hits a bottleneck (simplification from performance models!)

5. Energy to solution is power dissipation divided by performance

Model:

\[
E = \frac{\text{Power}}{\text{Performance}} = \frac{W_0 + (W_1 f + W_2 f^2)n}{P(n, f)}
\]
Two simple examples

\[ W_0 = 73 \text{ W} \quad \text{base} = 2 \text{ GHz} \]
\[ W_2 = 1 \text{ W} / \text{GHz}^2 \quad \text{Turbo} = 3 \text{ GHz} \]

**LINPACK type**

- Use all cores and high clock speed!

**STREAM type**

- Run all cores at clock speed that still saturates performance
Energy vs. Performance ("Z-plot")

"Isoline" of constant energy delay product \((E \times \Delta t)\)
Case study: Lattice Boltzmann

- Sparse representation lattice-Boltzmann flow solver
- Well suited for highly porous geometries, MPI parallel
- „AA pattern“ propagation → SIMD friendly, 304-376 bytes/LUP
- Saturating performance for vectorized code on modern Intel chips

Energy to solution vs. Performance on the socket (SNB) for a lattice-Boltzmann flow solver

(a) model

Bandwidth barrier

(b) measured

Optimization region

1.2 GHz AVX
2.0 GHz AVX
2.7 GHz AVX
Turbo AVX

open symbols: scalar, full socket

PPC=1
PPC=2
PPC=3
PPC=4

Performance [MFLUP/s]

Performance [MFLUP/s]
Single node → multi node

How does that change when going multi-node with substantial communication overhead?

- Dependence on socket-level concurrency?
- Dependence on clock speed?

Observations:

- Optimal PPC is crucial for lowest energy!
- Higher clock speed yields better performance without energy penalty!
Summary

- Analytical models are extremely helpful in understanding performance behavior and guiding optimizations
  - Roofline
  - ECM

- Simple power models can qualitatively describe the power consumption characteristics of code on the chip level
  - Saturating vs. scalable code
  - Z-plot
  - Energy-Delay-Product
  - Coupling with performance model
Thank You.

Holger Stengel
Johannes Hofmann
Julian Hammer
Jan Eitzinger
Gerhard Wellein