Multicore Scaling: The ECM Model

Single-core performance prediction
The saturation point
Stencil code examples:
  2D Jacobi in L1 and L2 cache
  3D Jacobi in memory
  3D long-range stencil

Assumptions and shortcomings of the roofline model

- Assumes one of two bottlenecks
  1. In-core execution
  2. Bandwidth of a single hierarchy level
- Latency effects are not modeled → pure data streaming assumed
- In-core execution is sometimes hard to model
- Saturation effects in multicore chips are not explained
  - ECM model gives more insight

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

Roofline predicts full socket BW

(c) RRZE 2014
The Execution-Cache-Memory (ECM) model
**ECM Model**

- ECM = “Execution-Cache-Memory”

- **Assumptions:**
  - Single-core execution time is composed of:
    1. In-core execution
    2. Data transfers in the memory hierarchy
  - Data transfers may or may not overlap with each other or with in-core execution
  - Scaling is linear until the relevant bottleneck is reached

- **Input:**
  - Same as for Roofline
  - + data transfer times in hierarchy

(c) RRZE 2014
Example: Schönauber Vector Triad in L2 cache

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

Machine characteristics:

- **Registers**
  - L1
  - L2
  - 1 LD/cy + 0.5 ST/cy
  - 32 B/cy (2 cy/CL)

Triad analysis (per CL):

- **Registers**
  - L1
  - L2
  - 6 cy/CL
  - 10 cy/CL

Arithmetic:
- 1 ADD/cy + 1 MULT/cy
- AVX: 2 cy/CL

Timeline:
- 16 F/CL (AVX)

Roofline prediction: 16/10 F/cy

Measurement: 16F / ≈17cy
Example: ECM model for Schönauer Vector Triad

\[ A(\cdot) = B(\cdot) + C(\cdot) \times D(\cdot) \] on a Sandy Bridge Core with AVX

```
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}
```

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

(c) RRZE 2014

ECM and power models
Testing different overlap hypotheses

No overlap

All caches single-ported

Full overlap beyond L2

0

6.04
L1

17.2
L2

26.3
L3

34

52.3
Memory

(c) RRZE 2014

Results suggest no overlap!
Multicore scaling in the ECM model

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

\[ P(n) = \min(nP_0, I \cdot b_S) \]

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

Model: Scales until saturation sets in

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

General observation:
- If the L1 cache is 100% occupied by LD, there is no overlap throughout the hierarchy
- If there is “slack” at the L1, there is overlap in the hierarchy
Example 1: A 2D Jacobi stencil in DP with SSE2 on Sandy Bridge
Example 1: 2D Jacobi in DP with SSE2 on SNB

```c
// Jacobi 2D line update
for(int j=start; j<end; j++){
    t1[i][j] = ( t0[i-1][j] +
                t0[i+1][j] +
                t0[i][j+1] +
                t0[i][j-1] ) * 0.25;
}
```

4-way unrolling → 8 LUP / iteration

Instruction count
- 13 LOAD
- 4 STORE
- 12 ADD
- 4 MUL
Example 1: 2D Jacobi in DP with SSE2 on SNB

**Processor characteristics**  
(SSE instructions per cycle)

- 2 LOAD  || (1 LOAD + 1 STORE)
- 1 ADD
- 1 MUL

**Code characteristics**  
(SSE instructions per iteration)

- 13 LOAD
- 4 STORE
- 12 ADD
- 4 MUL

---

core execution: 12 cy

(c) RRZE 2014
Example 1: 2D Jacobi in DP with SSE2 on SNB

- **Situation 1: Data set fits into L1 cache**
  - ECM prediction:
    \[(8 \text{ LUP} / 12 \text{ cy}) \times 3.5 \text{ GHz} = 2.3 \text{ GLUP/s}\]
  - Measurement: 2.2 GLUP/s

- **Situation 2: Data set fits into L2 cache (not into L1)**
  - 3 additional transfer streams from L2 to L1 (data delay)
  - Prediction:
    \[(8 \text{ LUP} / (12+6) \text{ cy}) \times 3.5 \text{ GHz} = 1.5 \text{ GLUP/s}\]
  - Measurement: 1.9 GLUP/s
Example 1: 2D Jacobi in DP with SSE2 on SNB

<table>
<thead>
<tr>
<th>LD</th>
<th>LD</th>
<th>LD</th>
<th>LD</th>
<th>2LD</th>
<th>2LD</th>
<th>2LD</th>
<th>2LD</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST</td>
<td>ST</td>
<td>ST</td>
<td>ST</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

LOAD bottleneck: 8.5 cy

core execution: 12 cycles

L2 delay: 6 cycles

L1 „single ported“
\[ \rightarrow \text{no overlap during LD/ST} \]

- ECM prediction w/ overlap:
  \[ (8 \text{ LUP} / (8.5+6) \text{ cy}) * 3.5 \text{ GHz} = 1.9 \text{ GLUP/s} \]
- Measurement: 1.9 GLUP/s

“\text{If the model fails, we learn something}”

(c) RRZE 2014

ECM and power models
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.

3. The scaling limit is set by the ratio of
   
   \[
   \frac{\text{# cycles per CL overall}}{\text{# cycles per CL at the bottleneck}}
   \]

4. The Roofline Model is recovered when assuming full overlap of all contributions.

Example:

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

Scaling limit: 43 / 19 = 2.3 cores

(c) RRZE 2014
Performance Modeling of Stencil Codes

Applying the ECM model to stencil updates:
- 3D Jacobi smoother (DP, AVX)
- Long-range stencil (SP, AVX)

(H. Stengel, RRZE)
Example 2: A 3D Jacobi smoother with AVX vectorization on an Intel Ivy Bridge processor
Jacobi 3D Manual Analysis

Operation Count
(1 LUP)
- MUL 1
- ADD 5
- LOAD 6
- STORE 1

Cycle Count
(4x unroll + AVX = 16 LUP)
- MUL 4
- ADD 20
- LOAD 24
- STORE 8

(c) RRZE 2014
Interlude: Intel Architecture Code Analyzer (IACA)

- Performs architecture-specific code analysis

- Prerequisite: Mark start and end of dominant work loop
  - In high-level code (documented)
  - In assembly code (see iacaMarks.h)
    - Does not influence code optimization (e.g. vectorization)
    - Assembly loop might perform multiple updates per iteration (unrolling, SIMD)

- Important reports (throughput mode):
  - Block throughput: runtime of one loop iteration (→ core-time)
  - Throughput bottleneck: limiting resource for code execution
  - Port pressure: dominant pipeline port
16 updates (4x unroll + AVX) = 2 cache lines per loop iteration

```plaintext
#pragma vector aligned
```

---

**Block Throughput:** 24.10 Cycles

**Throughput Bottleneck:** Port2_DATA, Port3_DATA

<table>
<thead>
<tr>
<th>Port</th>
<th>0 - DV</th>
<th>1</th>
<th>2 - D</th>
<th>3 - D</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>4.0</td>
<td>0.0</td>
<td>20.0</td>
<td>14.0</td>
<td>24.0</td>
<td>14.0</td>
</tr>
</tbody>
</table>

- ECM and power models
  - (c) RRZE 2014
Jacobi 3D ECM

Non-LD/ST time

Data transfers

FrontEnd stalls
0.5*(24.1 - 24)
=0.05cy

IACA throughput:
24.1cy/16LUP

Single-core performance
3.0GHz / (44cy/ 8LUP) = 545MLUP/s
Measurement (N=400): 542MLUP/s (~44cy)

Times [cy] for 8 LUP (DP)
= 1 CL update
= 0.5 loop iterations (ASM)
= 0.5 * IACA output

(c) RRZE 2014
Socket Scaling

Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz
Memory Bandwidth 47 GB/s

Jacobi 3D, N=400^3, double precision

Intel(R) Xeon(R) CPU E5-2690 v2 (1 socket, ivyep1), OMP_SCHEDULE=static

ECM Model 3.0GHz, MemBW 47GB/s

Measurement 3.0GHz (vector aligned)
Example 3: 3D long-range stencil in single precision with AVX on Sandy Bridge
Example 3: 3D long-range stencil in SP with AVX on SNB

Core execution

- 4 neighbors per direction
- Operations per update (code)
  - 27 LOAD (25 V, 1 ROC, 1 U)
  - 1 STORE (U)
  - 26 ADD
  - 15 MUL

- Core time & actual LOAD count → IACA

```
// 3D long-range line update (single precision)
for(i=4; i<nnx-4; i++) {
    lap = coef0 * V(i,j,k)
    + coef[1] * ( V(i+1,j,k) + V(i-1,j,k) )
    + coef[2] * ( V(i+2,j,k) + V(i-2,j,k) )
    + coef[3] * ( V(i+3,j,k) + V(i-3,j,k) )
    + coef[4] * ( V(i+4,j,k) + V(i-4,j,k) )
    + coef[3] * ( V(i,j+3,k) + V(i,j-3,k) )
    + coef[4] * ( V(i,j+4,k) + V(i,j-4,k) )
    + coef[4] * ( V(i,j,k+4) + V(i,j,k-4) )

    U(i,j,k) = 2.f * V(i,j,k) - U(i,j,k) + ROC2(i,j,k) * lap;
}
```

Collaboration with D. Keyes & T. Malas (KAUST)
### Core Execution time (16 LUP) = 2*34.25 cy = 68.5 cy

#### Data transfer: LOAD ports
\[ \Rightarrow \text{REG – L1: } 2*30.5 \text{ cy} = 61 \text{ cy} \]

#### 128 Bit Loads

- AVX vectorization, no unrolling: One iteration updates 8 SP (float) elements
- Multiply all numbers by 2X to get time for updating 1 CacheLine (16 floats)
Example 3: Data delay

- **Problem size:** $260^3$ (single precision) – cy/CL
- **Spatial blocking → Layer condition at L3 and row condition in L1: OK**

**From IACA analysis**
- 8 LOADS to $V$ can be served directly by L3 cache + 1 from main memory

<table>
<thead>
<tr>
<th>Registers</th>
<th>MemBW=40 GB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>256 bit LD &amp; 128 bit ST</td>
</tr>
<tr>
<td>L2</td>
<td>256 bit</td>
</tr>
<tr>
<td>L3</td>
<td>256 bit</td>
</tr>
<tr>
<td>Memory</td>
<td>119 bit (@2.7 GHz)</td>
</tr>
</tbody>
</table>

Minimum data transfer to main memory:
4 WORD/LUP (LD: U,V,ROC – ST:U)
Example 3: Putting it all together

**Core execution (Non-LD/ST cycles)**

- ADD: 52 cy
- MULT: 38 cy
- Reg-Reg transfers: 48 cy

**Data delay**

- L1-REG (Load): 61 cy
- L2-L1: 24 cy
- L3-L2: 24 cy
- M-L3: 17 cy
- Stores: 4 cy

IACA throughput: 68.5 cy / CL (sp)

FrontEnd stalls overlap: (68.5 - 61) cy = 7.5 cy

Single-core performance (ECM Model)

2.7GHz / (126 cy / 16LUP) = 343 MLUP/s

Measurement: 320 MLUP/s

(c) RRZE 2014

ECM and power models
Socket scaling

3D long-range stencil
1 socket, Sandy Bridge, 2.7GHz, intel64/13.1up03

memory bandwidth limit

MLUP/s

cores

(c) RRZE 2014
Saturation effects are ubiquitous; understanding them gives us opportunity to

- Find out about optimization opportunities
- Save energy by letting cores idle → see power model later on
- Putting idle cores to better use → communication, functional decomposition

Simple models work best. Do not try to complicate things unless it is really necessary!

Possible extensions to the ECM model

- Accommodate latency effects
- Model simple “architectural hazards”
A simple power model for the Sandy Bridge processor
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: \( W_{dyn} = (W_1 f + W_2 f^2) n \)
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 (simplification from the ECM model)
5. Energy to solution is power dissipation divided by performance

Model:

\[
E = \frac{\text{Power}}{\text{Performance}} = \frac{W_0 + (W_1 f + W_2 f^2) n}{\min(n P_0 f / f_0, P_{max})}
\]
Model predictions

\[ E = \frac{W_0 + (W_1 f + W_2 f^2)n}{\min(nP_0 f/f_0, P_{max})} \]

1. **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

![Graph showing performance vs. number of cores for different CPU frequencies and configurations.](a)

Better code
- earlier saturation
- smaller E @ saturation

(c) RRZE 2014
Model predictions

2. If there is saturation, $E$ is minimal near the saturation point

\[ E = \frac{W_0 + (W_1 f + W_2 f^2)n}{\min(nP_0 f / f_0, P_{\text{max}})} \]

\[ n_s = \frac{P_{\text{max}}}{P_0 f / f_0} \]
There is an optimal frequency $f_{opt}$ at which $E$ is minimal in the non-saturated case, with

$$f_{opt} = \sqrt{\frac{W_0}{W_2 n}} \quad \Rightarrow \text{depends on the baseline power}$$

$\Rightarrow$ “Clock race to idle” if baseline power is large (e.g., if we take the full system into account)!  

$$E = \frac{W_0 + (W_1 f + W_2 f^2)n}{\min(nP_0 f / f_0, P_{max})}$$
A model for multicore chip power

- 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 validate the power model
Model validation with the benchmark apps: Scalable codes

\[ f_{opt} = \sqrt{\frac{W_0}{W_2 n}} \]

1. Scalable code → optimal frequency \( f_{opt} \) for minimum energy

Trade-off between energy and best time to solution!? → May need more advanced cost functions!

2. More cores → lower frequency for optimum energy

3. Faster code → less energy
Model validation with the benchmark apps: Saturating code

Energy is minimal at the saturation point

Lower frequency $\rightarrow$ need more cores to saturate $\rightarrow$ less energy at saturation point
Introducing the Z-plot: Jacobi smoother (AVX)

Performance & energy to solution (chip-level base power $W_0 = 23W$) @ 2.7 GHz on Sandy Bridge EP

![Graph showing performance and energy to solution for different core counts and processor counts.]
A lattice-Boltzmann flow solver on the Sandy Bridge chip

ECM + Power model vs. measurements (chip level)

Lowest energy for

- best code (AVX)
- low-ish clock speed
- optimal number of cores (at bottleneck)

Optimal PPC is just as important as optimal clock speed!

optimization space: energy-performance trade-off

(c) RRZE 2014
Conclusions

- **Good, “fast” code is the zeroth-order energy saving strategy**

- **Energy behavior is quite different between saturating and scalable code**
  - **Saturating**: Set frequency as low as possible to still saturate, select optimal PPC!
  - **Scalable**: Use all cores, select \( f_{opt} \), consider energy-time trade-off

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

(c) RRZE 2014