The curses and blessings of analytic performance modeling

Georg Hager
Erlangen Regional Computing Center (RRZE), Erlangen, Germany

PPAM’2017, The 12th International Conference on Parallel Processing and Applied Mathematics
September 10-13, 2017, Lublin, Poland
Motivation

- Analytic 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?
  - What is the next bottleneck after optimization?
  - Impact of processor frequency and socket scalability → energy efficiency

- What if the model fails?
  - We learn something
  - We may still be able to use the model in a less predictive way
“Black box” vs. “white box” models

input data

black box

output data

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

predictions

Y

model OK?

N

insight

adjust model → insight

simplified description of system

white box

Load

Leaf Arm

Effort Arm

Load

Leaf Arm

Effort Arm

input data

modeling

predictions

validation

≈
What data/knowledge can a model be based on?

1. Only documented hardware properties + hypotheses
   - Purely analytic model

2. Hardware properties + (some) microbenchmark results + hypotheses
   - (Partly) phenomenological model

3. Measured performance/speedup data + hypotheses
   - Curve-fitting analytic model
Examples for white-/gray-box models

Amdahl’s Law with communication

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

Program speedup
Serial fraction

Hockney model for message transmission time

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

Latency
Message length
Bandwidth

Roofline model for loop code execution time

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

Time for computation
Time for data transfer

ECM model for loop code execution time

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

Non-overlapping execution
Time for data transfer

Overlapping execution
An example from physics

\[ \vec{F}_{12} = \frac{\gamma m_1 m_2}{r_{12}^3} \vec{r}_{12} \]

\[ h(t) = h_0 - \frac{1}{2} g t^2 \]
Models and insights

- Purely predictive analytic model
  - Direct **insight** into bottlenecks from first principles
  - Model failures challenge model assumptions or input data
  - Refinements lead to better **insights**

- Phenomenological analytic model
  - **Insight** with some “uncharted territory”
  - Model failure points to inaccurate or unsuitable measurements

- Curve-fitting “analytic” model
  - Yields **predictions**
  - Model failure indicates shortcomings of fitting approach
  - Refinements by using more fit parameters
"IF THE MODEL DOES NOT WORK, WE LEARN SOMETHING"

Three examples
Example: The Haswell port 7 AGU mystery

- Schönauer Vector Triad \( A(\cdot) = B(\cdot) + C(\cdot) \times D(\cdot) \)
- Haswell Xeon E5 core

Expectation: 8 iterations in 3 cycles

\[ P_{\text{max}} = 12.26 \text{ GF/s @ 2.3 GHz} \]
Generated assembly code (Intel V16up4)

- “Perfect code” at first sight
- Compiler only uses complex addressing mode
- Expected performance limit from port model:
  \[ P_{\text{max}} = 12.3 \text{ GFlop/s} \]

\begin{verbatim}
vmovupd (%r12,%r10,8),%ymm2
vmovupd 0x20(%r12,%r10,8),%ymm4
vmovupd 0x40(%r12,%r10,8),%ymm6
vmovupd 0x60(%r12,%r10,8),%ymm8
vmovupd (%r15,%r10,8),%ymm1
vmovupd 0x20(%r15,%r10,8),%ymm3
vmovupd 0x40(%r15,%r10,8),%ymm5
vmovupd 0x60(%r15,%r10,8),%ymm7
vfmadd213pd (%rsi,%r10,8),%ymm1,%ymm2
vfmadd213pd 0x20(%rsi,%r10,8),%ymm3,%ymm4
vfmadd213pd 0x40(%rsi,%r10,8),%ymm5,%ymm6
vfmadd213pd 0x60(%rsi,%r10,8),%ymm7,%ymm8
vmovupd %ymm2,0x0(%r13,%r10,8)
vmovupd %ymm4,0x20(%r13,%r10,8)
vmovupd %ymm6,0x40(%r13,%r10,8)
vmovupd %ymm8,0x60(%r13,%r10,8)
add $0x10,%r10
cmp %r9,%r10
jb 4016a0 <do_triad+_0x360>
\end{verbatim}
“If the model does not work, we learn something”

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

Haswell E5 2.3 GHz

Port 7 utilized by STORE instructions
Example: Weird scaling behavior of a stencil code

double precision, dimension(1:imx,1:kmax,0:1) :: phi
integer :: t0,t1
  t0 = 0 ; t1 = 1

!$OMP PARALLEL PRIVATE(it)
  do it = 1,imx
!$OMP DO SCHEDULE(STATIC)
    do k = 2,kmax-1
      do i = 2,imax-1
        phi(i,k,t1) = ( phi(i+1,k,t0) + phi(i-1,k,t0)
                       + phi(i,k+1,t0) + phi(i,k-1,t0) ) * 0.25d0
      enddo
    enddo
!$OMP END DO
!$OMP SINGLE
  i = t0 ; t0 = t1 ; t1 = i
!$OMP END SINGLE
  enddo
!$OMP END PARALLEL
Scaling behavior changes with inner dimension

Intel Broadwell Xeon E5 non-CoD
Working set 1 GiB
Solution: layer conditions!

Cache $k$ has size $C_k$

Layer condition (height $r$ stencil):

\[(2r + 1) \cdot N_i \cdot 8 \cdot B < \frac{C_k}{2}\]

2D 5-pt: $r = 1$
Validating the LC hypothesis

Measurement via likwid-perfctr (or PAPI)

(a) Performance [MLUP/s] vs. # cores for different values of \( \text{imax} \):
- \( \text{imax} = 8000 \)
- \( \text{imax} = 150000 \)

(b) Measured code balance \( B_c \) [byte/LUP] vs. # cores for different values of \( \text{imax} \):
- \( \text{imax} = 8000 \)
- \( \text{imax} = 150000 \)
Another riddle

Throughput $A(\cdot) = B(\cdot) + C(\cdot) \times D(\cdot)$

AMD Ryzen 1700X @ 3.0 GHz, Intel 17.0 up02

L2 faster than light???

https://blogs.fau.de/hager/archives/7810
FROM ROOFLINE TO ECM

Node-level white-/gray-box analytic modeling
Lowest order model: Roofline model (RLM)
Williams, Waterman, Patterson (2009), DOI: 10.1145/1498765.1498785

Limited resources impose upper (lower) performance (runtime) limits

\[ T = \max\{i,j\} \left( T_{\text{exec},j}, \frac{V_i}{b_{s,i}} \right) \]

Single-bottleneck view: perfect overlap!

The bottleneck

Notation:
\[ \{T_{OL}, T_{nOL}, T_{L2}, T_{L3}, T_{Mem}\} \xrightarrow{\text{prediction}} \{\max(T_{OL}, T_{nOL})\} \]

Data in L1

\[ \{\max(T_{OL}, T_{nOL}, T_{L2})\} \ldots \]

Data in L2
Next to lowest order: Execution Cache Memory (ECM) Model

Simple bottleneck picture does not hold for non-overlap scenarios: → ECM single core model for Intel x86 architectures

\[
\{T_{OL} \parallel T_{nOL} \mid T_{L1L2} \mid T_{L2L3} \mid T_{L3Mem} \} \xrightarrow{prediction} \{\max(T_{OL}, T_{nOL}) \mid \max(T_{OL}, T_{nOL} + T_{L1L2}) \mid \ldots\} \]

Software optimization targets (in this case)
What about multiple cores?

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

Performance vs. cores (Memory BN):

\[
P(n) = \min \left( nP(1), \frac{b_S^{\text{Mem}}}{B_C^{\text{Mem}}} \right)
\]

Number of cores at saturation:

\[
n_S = \left\lfloor \frac{b_S/B_C}{P(1)} \right\rfloor = \left\lfloor \frac{T_{\text{ECM}}^{\text{Mem}}}{T_{\text{L3Mem}}} \right\rfloor
\]

Example:

\[
\{ 8 \parallel 6 \mid 9 \mid 9 \mid 19 \} \text{ cy, } \{ 8 \mid 15 \mid 24 \mid 43 \} \text{ cy} \Rightarrow n_S = \left\lfloor \frac{43}{19} \right\rfloor = 3
\]
Predictive modeling: ECM Model for 2D 5-pt w/AVX on SNB 2.7 GHz

Radius-1 stencil $\rightarrow$ 3 layers have to fit

\[
\begin{align*}
\text{for}(j=1; & j < Nj-1; ++j) \\
\text{for}(i=1; & i < Ni-1; ++i) \\
& b[j][i] = (a[j][i-1] + a[j][i+1] \\
& \quad + a[j-1][i] + a[j+1][i]) * s;
\end{align*}
\]

8 iterations (DP):

<table>
<thead>
<tr>
<th>LC</th>
<th>ECM Model [cy]</th>
<th>prediction [cy]</th>
<th>$P_{\text{ECM}}^{\text{mem}}$ [MLUPS]</th>
<th>$N_i &lt; n_S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>${6</td>
<td>8</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>L2</td>
<td>${6</td>
<td>8</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>L3</td>
<td>${6</td>
<td>8</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>---</td>
<td>${6</td>
<td>8</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

LC = layer condition satisfied in ...
2D 5-pt serial in-memory performance and layer conditions

AUTOMATING ANALYTIC MODELS

Kerncraft
Kerncraft

user-input
- kernel code
- constants

compiler
- 

binary
- marked for IACA

IACA throughput analysis

in-core
- T_Ol, T_nOl

ECM/Roofline model
- data transfers
- T_L1L2, T_L2L3, T_L3MEM

pycparser

AST
- abstract syntax tree

data pattern

machine file
- documentation
- likwid-bench

Input
- intermediate
- output

#define N 1000
#define M 2000

for (j=1; j < N-1; ++j)
    for (i=1; i < M-1; ++i)
        b[i][j] = (a[i][j-1] + a[i][j+1] + a[i-1][j] + a[i+1][j]) * s;

vnovsd (%rsi,%rbx,8), %xmm1
vaddsd 16(%rsi,%rbx,8), %xmm1, %xmm2
vaddsd 8(%rdx,%rbx,8), %xmm2, %xmm3
vaddsd 8(%rdx,%rbx,8), %xmm3, %xmm4
vaddsd 8(%rdx,%rbx,8), %xmm4, %xmm5
vaddsd 8(%rdx,%rbx,8), %xmm5, %xmm6
vunpckd (%xmm6, %xmm5, %xmm4)

name | offsets ...
------------------------
| a ("rel", ", 0), ("rel", ", 1) | 
| ("rel", ", 0), ("rel", ", 1) | 
| ("rel", ", -1), ("rel", ", 0) | 
| ("rel", ", 1), ("rel", ", 0) | 
| s ("dir"); | 

clock: 2.7 GHz
- cacheline size: 64 B
memory hierarchy:
- [cores per group: 1, cycles per cacheline: 2, level: L2, size per group: 32 kB]
- [cores per group: 1, cycles per cacheline: 2, level: L2, size per group: 256 kB]
- [cores per group: 8, bandwidth: 40 GB/s, level: L3, size per group: 20 MB]
Machine File

- **model type:** Intel Core SandyBridge EP processor
- **model name:** Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
- **clock:** 2.7 GHz
- **sockets:** 2
- **cores per socket:** 8
- **threads per core:** 2
- **cacheline size:** 64 B
- **micro-architecture:** SNB
- **FLOPs per cycle:**
  - **SP:** {total: 16, ADD: 8, MUL: 8}
  - **DP:** {total: 8, ADD: 4, MUL: 4}
- **overlapping ports:** ["0", "0DV", "1", "2", "3", "4", "5"]
- **non-overlapping ports:** ["2D", "3D"]
- **compiler:** icc
- **compiler flags:** [-O3, -xAVX, -fno-alias]

**memory hierarchy:** [...]

**benchmarks:**
- **kernels:** [...]
- **measurements:** [...]

September 13, 2017 | PPAM 2017 | Georg Hager
Machine File – Memory Hierarchy

```python
memory hierarchy:
  - level: L1
    cache per group: {'sets': 64, 'ways': 8, 'cl_size': 64, # 32 kB
                      'replacement_policy': 'LRU', 'write_allocate': True, 'write_back': True,
                      'load_from': 'L2', 'store_to': 'L2'}
    cores per group: 1
    threads per group: 2
    groups: 16
    cycles per cacheline transfer: 2
  - level: L2
    cache per group: {'sets': 512, 'ways': 8, 'cl_size': 64, # 256 kB
                      'replacement_policy': 'LRU', 'write_allocate': True, 'write_back': True,
                      'load_from': 'L3', 'store_to': 'L3'}
    cores per group: 1
    threads per group: 2
    groups: 16
    cycles per cacheline transfer: 2
  - level: L3
    cache per group: {'sets': 20480, 'ways': 16, 'cl_size': 64, # 20 MB
                      'replacement_policy': 'LRU', 'write_allocate': True, 'write_back': True}
    cores per group: 8
    threads per group: 16
    groups: 2
    cycles per cacheline transfer: null
  - level: MEM
    cores per group: 8
    threads per group: 16
```

[...]
Machine File – Benchmark Infos

[...]
benchmarks:
  kernels:
    copy:
      FLOPs per iteration: 0
      read streams: {bytes: 8.00 B, streams: 1}
      read+write streams: {bytes: 0.00 B, streams: 0}
      write streams: {bytes: 8.00 B, streams: 1}
    daxpy:
      FLOPs per iteration: 2
      read streams: {bytes: 16.00 B, streams: 2}
      read+write streams: {bytes: 8.00 B, streams: 1}
      write streams: {bytes: 8.00 B, streams: 1}
    load:
      FLOPs per iteration: 0
      read streams: {bytes: 8.00 B, streams: 1}
      read+write streams: {bytes: 0.00 B, streams: 0}
      write streams: {bytes: 0.00 B, streams: 0}
    triad:
      FLOPs per iteration: 2
      read streams: {bytes: 24.00 B, streams: 3}
      read+write streams: {bytes: 0.00 B, streams: 0}
      write streams: {bytes: 8.00 B, streams: 1}
    update:
      FLOPs per iteration: 0
      read streams: {bytes: 8.00 B, streams: 1}
      read+write streams: {bytes: 8.00 B, streams: 1}
      write streams: {bytes: 8.00 B, streams: 1}
  measurements: [...]

measurements: [...]

September 13, 2017   |   PPAM 2017   |   Georg Hager
Machine File – Benchmark Results

benchmarks:
  kernels: [...]
measurements:
  L1:
    cores: [1, 2, 3, 4, 5, 6, 7, 8]
    results:
      copy: [81.98 GB/s, 163.75 GB/s, 245.62 GB/s, 327.69 GB/s, 409.41 GB/s, 489.83 GB/s, 571.67 GB/s, 653.50 GB/s]
      daxpy: [71.55 GB/s, 143.01 GB/s, 214.86 GB/s, 286.26 GB/s, 355.60 GB/s, 426.71 GB/s, 497.45 GB/s, 568.97 GB/s]
  [...]
size per core: [16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB]
size per thread: [16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB]
threads: [1, 2, 3, 4, 5, 6, 7, 8]
threads per core: 1
total size: [16.00 kB, 32.00 kB, 48.00 kB, 64.00 kB, 80.00 kB, 96.00 kB, 112.00 kB, 128.00 kB]
  [...]
MEM:
  cores: [1, 2, 3, 4, 5, 6, 7, 8]
  results:
    copy: [11.60 GB/s, 21.29 GB/s, 25.94 GB/s, 27.28 GB/s, 27.47 GB/s, 27.36 GB/s, 27.21 GB/s, 27.12 GB/s]
Kerncraft – Output

ECM model: \{ T_{OL} \parallel T_{nOL} \parallel T_{L1-L2} \parallel T_{L2-L3} \parallel T_{L3-MEM} \}

```
$ kerncraft --machine snb.yaml 2d-5pt.c --pmodel ECM -D N 5000 -D M 500

kernels/2d-5pt.c

{ 9.0 \parallel 8.0 | 10 | 6 | 12.74 } = 36.74 cy/CL
{ 9.0 \backslash 18.00 \backslash 24.00 \backslash 36.74 } cy/CL
saturating at 3 cores
$

$ kerncraft --machine snb.yaml 2d-5pt.c --pmodel Roofline --unit cy/CL -D N 5000 -D M 500

kernels/2d-5pt.c

Cache or mem bound with 1 core(s)
29.79 cy/CL due to L3-MEM transfer bottleneck (bw from copy benchmark)
Arithmetic Intensity: 0.17 FLOP/b
$```
Kerncraft – Results

2d-5pt.c in memory on single Intel Xeon E5-2680 (SandyBridge) core

- Roofline
- ECM
- L3-MEM
- L2-L3
- L1-L2
- OL
- nOL
Kerncraft – Results

2d-5pt.c in memory on single Intel Xeon E5-2680 (SandyBridge) core

- Spatial blocking
- Temporal blocking

- Roofline
- ECM

- L3-MEM
- L2-L3
- L1-L2
- nOL
- OL

N (of N*M matrix)

September 13, 2017 | PPAM 2017 | Georg Hager
Kerncraf – Spatial Blocking

2d-5pt.c in memory on single Intel Xeon E5-2680 (SandyBridge) core

- Roofline
- ECM
- Inner-dim. block:
  - 20032
  - 2048
  - 768

N (of N*M matrix)
So the problem is solved, or is it not?

- We would really like to see (some of) this in compilers
  - Work in progress (LLVM)
  - Problem/block sizes unknown at compile time
- IACA is a closed-source, Intel-only, unclear-future component
  - Work in progress (OS-ACA)
- Deriving machine models from automated benchmarks is dangerous
  - Compiler is an unpredictable component
- How to validate?
  - Kerncraft “Benchmark” mode
  - Hard to do within a compiler
- Coupling with energy model?
  - See references
  - Manual checking is mandatory
Thank You.

Julian Hammer
Holger Stengel
Jan Eitzinger
Gerhard Wellein
Further pointers

- (Semi-) Automatic modeling of streaming kernels with Kerncraft
  - [https://github.com/RRZE-HPC/kerncraft](https://github.com/RRZE-HPC/kerncraft)

- LIKWID toolkit for HPM measurements (and much more)
  - [https://github.com/RRZE-HPC/likwid](https://github.com/RRZE-HPC/likwid)

- Layer condition and block size calculator for stencil codes
  - [https://rrze-hpc.github.io/layer-condition/](https://rrze-hpc.github.io/layer-condition/)

- Girih test harness for temporally blocked stencil algorithms
  - [https://github.com/ecrc/girih](https://github.com/ecrc/girih)
Further references


Further references


