Making sense of temporally blocked stencil performance via analytic modeling

Georg Hager¹, Tareq Malas²

¹Erlangen Regional Computing Center (RRZE), Erlangen, Germany
²National Energy Research Scientific Computing Center (NERSC), Berkeley, CA

7th AICS International Symposium
Emerging Numerical Techniques for Exascale and Post-Moore Era
February 23-24, 2016, Kobe, Japan
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 $\rightarrow$ energy efficiency

What if the model fails?
- We learn something
- We may still be able to use the model in a less predictive way
Potential hardware capabilities, a.k.a. bottlenecks

```c
!$OMP PARALLEL DO
do j = 1, 7000
do i = 1, 80000
  y(i,j) = b*(x(i-1,j) + x(i+1,j) + x(i,j-1) + x(i,j+1))
enddo
dndo
!$OMP END PARALLEL DO
```

What you need:
Application requirements

What you get:
HW capability
Bottleneck

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Memory transfer</th>
<th>Pipeline latency 3 cycles (ADD)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50 GB/s</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computation</th>
<th>200 GF/s</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Inter-cache transfers</th>
<th>32 B/cycle</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Inter-LD transfer</th>
<th>8 GT/s</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Superscalarity</th>
<th>IPC=4 instr./cycle</th>
</tr>
</thead>
</table>

Solution / Runtime
Start with the basics: Resources for program execution

\[ T_{\text{exec}} = \frac{W_i}{\text{IPC}_{\text{max}}} \]

\[ T_{\text{data}} = \frac{V}{b_s} \]

Optimal ("balanced") co-design if we have equality


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

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

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

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

\[
n_S = \left\lfloor \frac{b_S/B_C}{P(1)} \right\rfloor = \left\lfloor \frac{T_{\text{ECM}}^{\text{Mem}}}{T_{L3\text{Mem}}} \right\rfloor
\]

Example:

\[
\{ 8 \| 6 \| 9 \| 9 \| 19 \} \text{ cy}, \quad \{ 8 \| 15 \| 24 \| 43 \} \text{ cy} \Rightarrow n_S = \left\lfloor \frac{43}{19} \right\rfloor = 3
\]
Example: 2D 5-point stencil & layer conditions

Cache $k$ has size $C_k$

Layer condition (height $r$ stencil):

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

2D 5-pt: $r = 1$
Predictive modeling: ECM Model for 2D 5-pt w/AVX on SNB 2.7 GHz

Radius-1 stencil \(\rightarrow\) 3 layers have to fit

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

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;)</th>
<th>(n_S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>{6</td>
<td></td>
<td>8</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>L2</td>
<td>{6</td>
<td></td>
<td>8</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>L3</td>
<td>{6</td>
<td></td>
<td>8</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>{6</td>
<td></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

Temporal blocking from the resources/modeling point of view

Major step: remove contribution from L3-Mem

→ Bottleneck is gone
→ Scalable resources
→ Other effects dominate scaling
→ Very limited single-core effect
From spatial blocking to temporal blocking: Naïve stencil update order

Data dependency
Wavefront temporal blocking

- Data pipelining in the cache:
  - Bring new element to the front
  - Write back result from the back
- Provides maximum data reuse in space-time blocks
Diamond tiling

Maximum data reuse of loaded block, reducing memory accesses

Provides unified tile shape

Provides independent space-time blocks to reduce synchronization between threads
Single-core Wavefront temporal blocking + Diamond tiling (1WD)

From 1WD to MWD: Cache block sharing

- Reduce the required number of tiles
- Increase the tile size
- More in-cache data reuse
- Less memory bandwidth pressure

Varying grid size (all frameworks) on 18-core Haswell

Performance

GLUPS

7pt constant coef.

7 Flops/LUP

7pt variable coef.

13 Flops/LUP

25pt constant coef.

33 Flops/LUP

25pt variable coef.

37 Flops/LUP

Sustained mem. BW

MEM GB/s

MWD
1WD
Spt.bik.
PLUTO
Pchoir

Measured Bytes/LUP

MEM Bytes/LUP

Size in each dimension

Size in each dimension

Size in each dimension

Size in each dimension
Thread scaling (all frameworks) on 18-core Haswell

7pt constant coefficient stencil

Phenomenological modeling

Roadblocks for predictive modeling

- Short(ish) loops → broken steady-state assumption → inaccurate in-core prediction
- Finite block sizes → boundary effects → inaccurate data traffic prediction

Remedy for traffic prediction: don’t predict but measure the traffic

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7-pt const. coeff.</td>
<td>{12</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>7-pt var. coeff.</td>
<td>{14</td>
<td>21</td>
<td>14</td>
</tr>
<tr>
<td>25-pt const. coeff.</td>
<td>{12</td>
<td>56</td>
<td>20</td>
</tr>
<tr>
<td>25-pt var. coeff.</td>
<td>{12</td>
<td>38</td>
<td>56</td>
</tr>
</tbody>
</table>
What use is phenomenological modeling?

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** into interaction of data transfer & execution
- Model failure points to inaccurate in-core model
- Refinements lead to better **insights**

Curve-fitting “analytic” model
- Yields **scalability** predictions
- Model failure indicates shortcomings of fitting approach
- Refinements by using more fit parameters

\[ \hat{F} = ma \]
Further pointers

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

- **LIKVID 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**
  - [https://rrze-hpc.github.io/layer-condition/](https://rrze-hpc.github.io/layer-condition/)

- **Girihi test harness for temporally blocked stencil algorithms**
  - [https://github.com/ecrc/girih](https://github.com/ecrc/girih)
Thank You.

Holger Stengel
Julian Hammer
Jan Eitzinger
Gerhard Wellein
Further references


Further references


Abstract

Many techniques have been devised to improve the performance of stencil algorithms on cache-based multicore CPUs. The main goal is to decouple from the scarce resource of main memory bandwidth, but this is just where the real challenges begin: How can the equally scarce cache space be used most effectively? What is the next bottleneck beyond memory bandwidth? Does it make sense to block for higher-level caches? What is the role of low-level code quality? What is a good parallelization strategy? Some of these questions can be answered by auto-tuning techniques, but others require deeper analysis with the help of analytic performance models. Such models enable us to pinpoint relevant performance issues and sometimes lead to surprising insights. The talk gives an introduction to useful analytic performance models of streaming kernels and how they can be applied to temporally blocked stencil algorithms. Using relevant corner cases we demonstrate how far these models can take us and where they stop being purely predictive. Beyond this point one can still use the principles behind the models to learn more about the bottlenecks or shortcomings of running code by constructing the models not from first principles but from performance counter measurements. We call this approach “phenomenological modeling.” Backed by these concepts we present an analysis and comparison among state-of-the-art stencil frameworks.