Analytical Tool-Supported Modeling of Streaming and Stencil Loops

Georg Hager, Julian Hammer
Erlangen Regional Computing Center (RRZE)

Scalable Tools Workshop
August 3-6, 2015, Lake Tahoe, CA
<table>
<thead>
<tr>
<th>RRZE</th>
</tr>
</thead>
</table>
| - LIKWID  
tiny.cc/LIKWID |
| - GHOST  
tiny.cc/GHOST |
| - Performance Engineering  
http://blogs.fau.de/...  
hager/talks/nlpe |
Motivation

DAXPY on Sandy Bridge core

Loop length

Inner dimension w/ in-memory data

2D-5pt stencil on Sandy Bridge core
THE ECM MODEL

Registers

L1

L2

L3

MEM
ECM model – the rules

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 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:

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

# cy invariant to clock speed
# cy changes w/ clock speed

Example from previous slide:

\[ \{ 8 \parallel 6 \parallel 9 \parallel 9 \parallel 19 \} \text{ cy} \]
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 – saturation

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

Performance vs. cores (Memory BN):

\[
P_{ECM}(n) = \min \left( np^{Mem}_{ECM}, \frac{b_S^{Mem}}{b_C^{Mem}} \right)
\]

Number of cores at saturation:

\[
n_S = \left[ \frac{b_S/B_C}{p^{Mem}_{ECM}} \right] = \left[ \frac{T^{Mem}_{ECM}}{T_{L3Mem}} \right]
\]

Example:

\[
\{8 \parallel 6 \mid 9 \mid 9 \mid 19\} \text{ cy, } \{8 \mid 15 \mid 24 \mid 43\} \text{ cy} \implies n_S = \left[ \frac{43}{19} \right] = 3
\]
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
2D 5-PT JACOBI STENCIL
(DOUBLE PRECISION)

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

Unit of work (1 CL): 8 LUPs

Data transfer per unit:
- 5 CL if layer condition violated in higher cache level
- 3 CL if layer condition satisfied
ECM Model for 2D Jacobi (AVX) on SNB 2.7 GHz

Radius-\( r \) stencil \( \rightarrow (2r+1) \) layers have to fit

\[
\text{for}(j=1; j < N_j-1; ++j) \\
\text{for}(i=1; i < N_i-1; ++i) \\
\quad b[j][i] = (a[j][i-1] + a[j][i+1] \\
\quad \quad + a[j-1][i] + a[j+1][i]) \times s;
\]

Cache \( k \) has size \( C_k \)

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

2D 5-pt: \( r = 1 \)

<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|8|6|6|13}</td>
<td>{8|14|20|33}</td>
<td>659</td>
<td>683</td>
<td>3</td>
</tr>
<tr>
<td>L2</td>
<td>{6|8|10|6|13}</td>
<td>{8|18|24|37}</td>
<td>587</td>
<td>5461</td>
<td>3</td>
</tr>
<tr>
<td>L3</td>
<td>{6|8|10|10|13}</td>
<td>{8|18|28|41}</td>
<td>529</td>
<td>436900</td>
<td>4</td>
</tr>
<tr>
<td>—</td>
<td>{6|8|10|10|22}</td>
<td>{8|18|28|50}</td>
<td>438</td>
<td>N/A</td>
<td>3</td>
</tr>
</tbody>
</table>

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

SNB 2.7 GHz

- LC in L2
- LC in L3
- no LC

Performance [MLUP/s]

- $B_{C_{L2}} = 24 \text{ B/LUP}$
- $B_{C_{L3}} = 40 \text{ B/LUP}$
- $B_{C_{mem}} = 24 \text{ B/LUP}$
- $B_{C_{mem}} = 40 \text{ B/LUP}$

Leading dimension ($N_1$)
#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.f * 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 $n_s = \left[\frac{127}{17}\right] = 8$ 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
First steps towards automated model construction
kerncraft: ECM/Roofline modeling toolkit

Loop Kernel Analysis and Performance Modeling Toolkit

This tool allows automatic analysis of loop kernels using the Execution Cache Memory (ECM) model, the Roofline model and actual benchmarks. kerncraft provides a framework to investigate the data reuse and cache requirements by static code analysis. In combination with the Intel IACA tool kerncraft can give a good overview of both in-core and memory bottlenecks and use that data to apply performance models.

Installation

Run: pip install kerncraft

Additional requirements are:
- Intel IACA tool, with (working) isca.sh in PATH environment variable (used by ECM, ECMCPU and Roofline models)
- likwid (used in Benchmark model and by likwid_bench_auto.py)
Towards automated model generation

Manual

- Code inspection and/or IACA
- Traffic analysis w/ layer conditions
- HW limits: micro-benchmarking & docs

Automated

- Registers
- IACA or direct analysis
- Reuse distance analysis, cache simulation
- HW limits: likwid-bench & docs
```c
#define N 1000
#define M 2000

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

**Compiler**

**AST**

**Cache simulator/reuse distance analysis**

**Traffic volumes**

\[ T = \frac{V}{b} \]

**Machine description (yaml file)**

**Likwid-bench**

**IACA TP/CP**

**Roofline / ECM model**

\[ T_{OL}, T_{nOL}, T_{L1L2}, \ldots, T_{L3Mem} \]

**Vmovsd (%rsi,%rbx,8), %xmm1**

**Vaddsd 16(%rsi,%rbx,8), %xmm1, %xmm2**

**Vaddsd 8(%rdx,%rbx,8), %xmm2, %xmm3**

**Vaddsd 8(%rcx,%rbx,8), %xmm3, %xmm4**

**Vaddsd 8(%r8,%rbx,8), %xmm4, %xmm5**

**Vaddsd 8(%r9,%rbx,8), %xmm5, %xmm6**

**Vmulsd %xmm6, %xmm0, %xmm7**
Restrictions on code input (selection)

- Only doubles and ints supported
- Array declarations may use fixed sizes or constants, with an optional offset (e.g., double u1[M+3][N-2][23], but not double u[M*N])
- Only the innermost loop may contain assignment statements
- Array references must either use index variables from for-loops, with optional addition or subtraction, constant or fixed values
- All for-loops must use a declaration as initial statement and an increment or a decrement assignment operation as the next statement (e.g., i++, i -= 2)
- Function calls and the use of pointers is not allowed anywhere in the kernel code
- Write access to any data is assumed to use “normal” STORE instructions (e.g., no non-temporal stores)
Operating modes

- **ECM**
  - Full ECM model including in-core analysis
- **ECMData**
  - Data traffic analysis only (works on any system)
- **ECMCPU**
  - In-core part of ECM model (IACA)
- **Roofline**
  - Full Roofline model using CPU peak performance as in-core limit
- **RooflineIACA**
  - Full Roofline model using IACA analysis for in-core
- **Benchmark**
  - Run the actual benchmark for model validation
Machine file example: 8-core SNB EP node

clock: 2.7 GHz
cores per socket: 8
model type: Intel Core SandyBridge EP processor
model name: Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
sockets: 2
threads per core: 2
cacheline size: 64 B
icc architecture flags: [-xAVX]
micro-architecture: SNB
FLOPs per cycle:
   SP: {total: 8, ADD: 4, MUL: 4}
   DP: {total: 4, ADD: 2, MUL: 2}
overlapping ports: ["0", "0DV", "1", "2", "3", "4", "5"]
non-overlapping ports: ["2D", "3D"]
memory hierarchy:
- {cores per group: 1, cycles per cacheline transfer: 2,
  groups: 16, level: L1, bandwidth: null, size per group: 32.00 kB, threads per group: 2}
- {cores per group: 1, cycles per cacheline transfer: 2,
  groups: 16, level: L2, bandwidth: null, size per group: 256.00 kB, threads per group: 2}
- {bandwidth per core: 18 GB/s, cores per group: 8, cycles per cacheline transfer: null,
  groups: 2, level: L3, bandwidth: 40 GB/s, size per group: 20.00 MB, threads per group: 16}
- {cores per group: 8, cycles per cacheline transfer: null,
  level: MEM, bandwidth: null, size per group: null, threads per group: 16}
[...]
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

[...]
measurements:

[...]

MEM:

1:

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]
daxpy: [17.33 GB/s, 31.89 GB/s, 38.65 GB/s, 40.50 GB/s, 40.81 GB/s, 40.62
GB/s, 40.59 GB/s, 40.26 GB/s]
load: [12.01 GB/s, 23.04 GB/s, 32.79 GB/s, 40.21 GB/s, 43.39 GB/s, 44.14
GB/s, 44.42 GB/s, 44.40 GB/s]
triad: [12.73 GB/s, 24.27 GB/s, 30.43 GB/s, 31.46 GB/s, 31.77 GB/s, 31.74
GB/s, 31.65 GB/s, 31.52 GB/s]
update: [18.91 GB/s, 32.43 GB/s, 37.28 GB/s, 39.98 GB/s, 40.99 GB/s, 40.92
GB/s, 40.61 GB/s, 40.34 GB/s]

size per core: [40.00 MB, 20.00 MB, 13.33 MB, 10.00 MB, 8.00 MB, 6.67 MB,
5.71 MB, 5.00 MB]

size per thread: [40.00 MB, 20.00 MB, 13.33 MB, 10.00 MB, 8.00 MB, 6.67 MB,
5.71 MB, 5.00 MB]

threads: [1, 2, 3, 4, 5, 6, 7, 8]

threads per core: 1

total size: [40.00 MB, 40.00 MB, 40.00 MB, 40.00 MB, 40.00 MB, 40.00 MB, 40.00 MB, 40.00 MB, 40.00 MB, 40.00 MB]
Cache reuse analysis

- **Cached only in last level cache**
- **Cached in second (L2) and last level cache**
- **Cached in first (L1), second and last level cache**
- **Miss on all cache levels**
- **Miss in L1 and hit in L2**
- **Hit in L1**
- **Data for cache line update**
- **Loop center**
kerncraft usage

$ kerncraft -h
usage: kerncraft [-h] [-v[v]] --machine MACHINE
       --pmodel{ECM,ECMData,ECMCPU,Roofline,RooflineIACA,Benchmark}
       [-D KEY VALUE] [--testcases] [--testcase-index INDEX]
       [--verbose] [--asm-block BLOCK] [--store PICKLE]
       [--ecm-plot ECM_PLOT]
       FILE [FILE ...]

Examples:

$ kerncraft -vv -p ECM -m phinally.yaml 2d-5pt.c -D N 10000 -D M 10000

$ kerncraft -v -p Roofline -m phinally.yaml 2d-5pt.c -D N 10000 -D M 10000
kerncraft example (ECM)

```
$ kerncraft -vv -p ECM -m phinally.yaml 2d-5pt.c -D N 10000 -D M 10000
```

```
double a[M][N];
double b[M][N];
double s;

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

variables:

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>double</td>
<td>(10000, 10000)</td>
</tr>
<tr>
<td>s</td>
<td>double</td>
<td>None</td>
</tr>
<tr>
<td>b</td>
<td>double</td>
<td>(10000, 10000)</td>
</tr>
</tbody>
</table>
kerncraft example (ECM) continued

<table>
<thead>
<tr>
<th>loop stack:</th>
<th>idx</th>
<th>min</th>
<th>max</th>
<th>step</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>j</td>
<td>1</td>
<td>9999</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td>i</td>
<td>1</td>
<td>9999</td>
<td>+1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data sources:</th>
<th>name</th>
<th>offsets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>('rel', 'j', 0), ('rel', 'i', -1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>('rel', 'j', 0), ('rel', 'i', 1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>('rel', 'j', -1), ('rel', 'i', 0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>('rel', 'j', 1), ('rel', 'i', 0)</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>('rel', 'j', 0), ('rel', 'i', 0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data destinations:</th>
<th>name</th>
<th>offsets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>('rel', 'j', 0), ('rel', 'i', 0)</td>
</tr>
</tbody>
</table>
FLOPs: | op  | count |
------|------|-------|
+    |      | 3     |
*    |      | 1     |

4

constants: | name | value |
-----------|------|-------|
M          | 10000|
N          | 10000|

Ports and cycles: {'1': 6.0, '0DV': 0.0, '2D': 8.0, '0': 5.05, '3': 9.0, '2': 9.0, '5': 5.95, '4': 4.0, '3D': 8.0}
Uops: 37.0
Throughput: 9.45cy per CL
T_nOL = 8.0cy
T_OL = 9.0cy
Trace length per access in L1: 982
Hits in L1: 30 {'a': {'ji': [10006, 10005, 10004, 10003, 10002, 10001, 10000, 7, 6, 5, 4, 3, 2, 1, 0, -1, -9994, -9995, -9996, -9997, -9998, -9999, -10000]}, 's': {}, 'b': {'ji': [6, 5, 4, 3, 2, 1, 0]}}
Misses in L1: 4 (4CL): {'a': {'ji': [10007, 8, -9993]}, 's': {}, 'b': {'ji': [7]}}
Evicts from L1 8 (1CL): {'a': {}, 's': {}, 'b': {'ji': [7, 6, 5, 4, 3, 2, 1, 0]}}

L1-L2 = 10cy
L2-L3 = 10cy
L3-MEM = 12.96cy
{ 9.0 | | 8.0 | 10 | 10 | 12.96 } cy
{ 9.0 \ 18 \ 28 \ 41 } cy
### kerncraft example (Roofline)

```bash
$ kerncraft -v -p Roofline -m phinally.yaml 2d-5pt.c -D N 10000 -D M 10000
```

... 

#### Bottlenecks:

<table>
<thead>
<tr>
<th>level</th>
<th>a. intensity</th>
<th>performance</th>
<th>bandwidth</th>
<th>bandwidth kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
<td>21.60 GFLOP/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU-L1</td>
<td>0.083 FLOP/b</td>
<td>8.50 GFLOP/s</td>
<td>102.01 GB/s</td>
<td>triad</td>
</tr>
<tr>
<td>L1-L2</td>
<td>0.1 FLOP/b</td>
<td>5.12 GFLOP/s</td>
<td>51.15 GB/s</td>
<td>triad</td>
</tr>
<tr>
<td>L2-L3</td>
<td>0.1 FLOP/b</td>
<td>3.15 GFLOP/s</td>
<td>31.48 GB/s</td>
<td>triad</td>
</tr>
<tr>
<td>L3-MEM</td>
<td>0.17 FLOP/b</td>
<td>2.90 GFLOP/s</td>
<td>17.40 GB/s</td>
<td>copy</td>
</tr>
</tbody>
</table>

Cache or mem bound

**2.90 GFLOP/s due to L3-MEM transfer bottleneck** (bw with from copy benchmark)

**Arithmetic Intensity:** 0.17 FLOP/b
Interpretation of predictions: 3D long-range stencil

Automated loop performance model construction | G. Hager
Layer conditions in the 3D long-range stencil

1D layer-condition:

2D layer-condition:

3D layer-condition:

- stencil center-point
- stream head (uncached)
- cached element
Comparison of measurements with predictions: 3D long-range stencil
Summary & remarks

- No silver bullet
  - Tool output must be checked
  - Validation is absolutely mandatory
  - If the model does not work, we learn something

- Future work
  - Lift some of the restrictions on the C formulation of the loop code
  - Include saturation analysis
  - Become more independent of external tools
    - IACA, icc
  - Improve simplistic reuse analysis
References


Further references


Thank You.

Julian Hammer
Johannes Hofmann
Holger Stengel
Jan Eitzinger