Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL and the COSMO Weather Code
Microarchitectures are becoming more and more complex
How to optimize codes for these complex architectures?

- **Performance engineering**: “encompasses the set of roles, skills, activities, practices, tools, and deliverables applied at every phase of the systems development life cycle which ensures that a solution will be designed, implemented, and operationally supported to meet the non-functional requirements for performance (such as throughput, latency, or memory usage).”

- **Manually profile codes and tune** them to the given architecture
  - Requires highly-skilled performance engineers
  - Need familiarity with
    - NUMA (topology, bandwidths etc.)
    - Caches (associativity, sizes etc.)
    - Microarchitecture (number of outstanding loads etc.)

Trust me, I’m an engineer!
An engineering example – Tacoma Narrows Bridge
Scientific **Performance** Engineering

1) Observe

2) Model

3) Understand

4) Build
Modeling by example: KNL Architecture (mesh)

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS’17
KNL Architecture (memory: Flat & Cache)

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS‘17
KNL Architecture (all to all mode)

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS’17
KNL Architecture (Quadrant or Hemisphere)
KNL Architecture (SNC-4 or SNC-2)

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS’17
How much does this all matter?
What is the real cost of accessing cache?
What is the cost of accessing memory?

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS’17
Step 1: Understand core-to-core transfers – MESIF cache coherence

Write back overhead

Location: only 5-15% difference

Contention effects?

That is curious!

All values are medians within 10% of the 95% nonparametric CI, cf. TH, RB: “Scientific Benchmarking of Parallel Computing Systems”, SC16
Step 2: Understand core-to-memory transfers – DRAM and MCDRAM

- MCDRAM 20% slower!
- MCDRAM 4-6x faster!
- Need to read **and** write for full bandwidth
- Cache mode >20% slower
- Bandwidth suffers a bit

All values are medians within 10% of the 95% nonparametric CI, cf. TH, RB: “Scientific Benchmarking of Parallel Computing Systems”, SC16
Performance engineers optimize your code!

Are you kidding me?

Contention [ns] (1:N copy) $\alpha$
Linear, $T_C(N) = \alpha + \beta \cdot N$

<table>
<thead>
<tr>
<th>Software NUMA</th>
<th>Software UMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNC4</td>
<td>SNC2</td>
</tr>
<tr>
<td>Latency [ns]</td>
<td></td>
</tr>
<tr>
<td>Local (L1)</td>
<td>3.8</td>
</tr>
<tr>
<td>Tile (L2)</td>
<td>34 (M)</td>
</tr>
<tr>
<td>Remote</td>
<td>14 (S,F)</td>
</tr>
<tr>
<td>107-112 (M)</td>
<td>111-125 (M)</td>
</tr>
<tr>
<td>116 (E)</td>
<td>116 (E)</td>
</tr>
<tr>
<td>96-118 (S,F)</td>
<td>104-117 (S,F)</td>
</tr>
<tr>
<td>109-117 (S,F)</td>
<td></td>
</tr>
<tr>
<td>Bandwidth [GB/s] (Read)</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>Remote</td>
<td>7.7</td>
</tr>
</tbody>
</table>

Congestion (P2P pairs) None

Contention [ns] (1:N copy) $\alpha$
Linear, $T_C(N) = \alpha + \beta \cdot N$

<table>
<thead>
<tr>
<th>Software NUMA</th>
<th>Software UMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNC4</td>
<td>SNC2</td>
</tr>
</tbody>
</table>
| Flat Mode | Latency [ns] (BenchT)
| DRAM | 130-140 | 134-146 | 140 | 140 | 139 |
| MCDRAM | 160-175 | 160-170 | 160 | 167 | 160 |
| Bandwidth [GB/s] (Copy NT / STREAM Copy)
| DRAM | 69 / 77 | 70 / 77 | 71 / 77 | 71 / 77 | 306 / 359 |
| MCDRAM | 342 / 418 | 333 / 388 | 333 / 415 | 315 / 372 |
| Bandwidth [GB/s] (Read)
| DRAM | 71 | 71 | 71 | 71 | 71 |
| MCDRAM | 288 | 288 | 288 | 288 |
| Bandwidth [GB/s] (Write)
| DRAM | 33 | 34 | 36 | 36 |
| MCDRAM | 147 | 163 | 165 | 161 |
| Bandwidth [GB/s] (Trial NT / STREAM Trial)
| DRAM | 71 / 82 | 71 / 82 | 71 / 82 | 71 / 82 | 73 / 82 |
| MCDRAM | 371 / 448 | 347 / 441 | 340 / 401 | 332 / 434 |
| Cache Mode | Latency [ns] (BenchT)
| DRAM | 158-178 | 167-171 | 166 | 168 | 172 |
| Bandwidth [GB/s] (Copy NT / STREAM Copy)
| DRAM | 150 / 252 | 130 / 252 | 175 / 255 | 134 / 237 |
| Bandwidth [GB/s] (Read)
| DRAM | 87 | 95 | 124 | 128 | 118 |
| Bandwidth [GB/s] (Write)
| DRAM | 56 | 56 | 72 | 72 | 68 |
| Bandwidth [GB/s] (Trial NT / STREAM Trial)
| DRAM | 296 / 292 | 246 / 294 | 296 / 309 | 273 / 274 | 264 / 269 |
A principled approach to designing cache-to-cache broadcast algorithms

Multi-ary tree example

- Depth $d = 2$
  - $k_1 = 2$
- $k_2 = 3$

Tree depth

Level size

Tree cost

Reached threads

\[ T_{tree} = \sum_{i=1}^{d} T_C(k_i) = \sum_{i=1}^{d} (c \cdot k_i + b) \]

\[ = \sum_{i=1}^{d} (R_R + R_L + c \cdot (k_i - 1)) \]

\[ T_{sbcast} = \min_{d, k_i} \left( T_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} T_{nb}(k_i + 1) \right) \]

\[ N \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j, \quad \forall i < j, k_i \leq k_j \]

S. Ramos, TH: "Cache line aware optimizations for ccNUMA systems (IEEE TPDS’17)."
Model-driven performance engineering for broadcast

Binary or binomial trees?

\[
T_{\text{sbcast}} = \min_{d, k_i} \left( T_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} T_{nb}(k_i + 1) \right)
\]

\[
N \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j, \ \forall i < j, k_i \leq k_j
\]

Oh! But sure I can do that.

S. Ramos and T. Hoefler: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS'17
Model-driven performance engineering for broadcast

Binary or binomial trees?

13x faster, lower variance

Oh! But sure I can do that.

S. Ramos and T. Hoeffer: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL, IPDPS'17
Easy to generalize to similar algorithms

Barrier (7x faster than OpenMP)

Reduce (5x faster than OpenMP)
What about real applications?

Image credit: Oliver Fuhrer, MeteoSwiss
A factor $2\times$ in resolution roughly corresponds to a factor $10\times$ compute

$\Delta x = 35 \text{ m} \ (1\times)$

Image credit: Oliver Fuhrer, MeteoSwiss
A factor 2x in resolution roughly corresponds to a factor 10x compute.
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute

$\Delta x = 140 \text{ m (100x)}$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute.

$\Delta x = 280 \text{ m} \ (1,000x)$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute.

$\Delta x = 550 \text{ m} \ (10,000x)$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute

Operational model of MeteoSwiss today!

$\Delta x = 1100 \text{ m (100,000x)}$
A factor $2x$ in resolution roughly corresponds to a factor $10x$ compute

Operational model of MeteoSwiss before 2016!

$\Delta x = 2200 \text{ m} \ (1,000,000x)$

We’re a factor of $100,000$ away!

Image credit: Oliver Fuhrer, MeteoSwiss
Basic Atmospheric Equations

\[
\begin{align*}
\frac{d\mathbf{v}}{dt} &= -\nabla p + \rho g - 2\Omega \times (\rho \mathbf{v}) - \nabla \cdot (T) \\
\frac{dp}{dt} &= -(c_{pd}/c_{vd})\rho \nabla \cdot \mathbf{v} + (c_{pd}/c_{vd} - 1)Q_h \\
\frac{dT}{dt} &= \frac{dp}{dt} + Q_h \\
\frac{d\rho_e}{dt} &= -\nabla \cdot \mathbf{F}^w - (I^i + I^j) \\
\frac{d\rho_e^w}{dt} &= -\nabla \cdot (\mathbf{P}^{k,j} + \mathbf{F}^{k,j}) + I^{k,j} \\
\rho &= p\{R_d(1 + (R_v/R_d - 1)q^p - q^l - q^f)T_j^{l-1}\}
\end{align*}
\]

Discretized on a compute grid

1 – 10 km
20 – 600 m

ECMWF-Model
16 km Grid
2 x per day
10 days prediction

COSMO-7
6.6 km Grid
3 x per day
72 h prediction

COSMO-1
1.1 km Grid
7 x pro day 33 h prediction
1 x pro day 45 h prediction

Slide adopted from T. Schulthess
The COSMO Code – 300k SLOC Fortran

Two main algorithmic motifs in dynamical core

Finite Difference Stencils

Tridiagonal Solvers

```fortran
  do j = 1, niter
    do i = 1, nwork
      c(i) = a(i) + b(i) * ( a(i+1) - 2.0d0*a(i) + a(i-1) )
    end do
  end do
```
Stencil computations (oh no, another stencil talk)

Motivation:
- Important algorithmic motif (e.g., finite difference method)

Definition:
- Element-wise computation on a regular grid using a fixed neighborhood
- Typically working on multiple input fields and writing a single output field

\[
lap(i,j) = -4.0 \times \text{in}(i,j) + \text{in}(i-1,j) + \text{in}(i+1,j) + \text{in}(i,j-1) + \text{in}(i,j+1)
\]

due to the typically low arithmetic intensity stencil computations are often memory bandwidth limited!
How to tune such stencils (most other stencil talks)

- **LOTS of related work!**
  - Compiler-based (e.g., Polyhedral such as PLUTO [1])
  - Auto-tuning (e.g., PATUS [2])
  - Manual model-based tuning (e.g., Datta et al. [3])
  - ... essentially every micro-benchmark or tutorial, e.g.:

- **Common features**
  - Vectorization tricks (data layout)
  - Advanced communication (e.g., MPI neighbor colls)
  - Tiling in time, space (diamond etc.)
  - Pipelining

- **Much of that work DOES NOT compose well with complex stencil programs in weather/climate**

---

[1]: Uday Bondhugula, A. Hartono, J. Ramanujan, P. Sadayappan. A Practical Automatic Polyhedral Parallelizer and Locality Optimizer , PLDI'08
[3]: Kaushik Datta, et al., Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors, SIAM review
What is a “complex stencil program”? (this stencil talk)

E.g., the COSMO weather code
- is a regional climate model used by 7 national weather services
- contains hundreds of different complex stencils

Modeling stencils formally:
- Represent stencils as DAGs
  - Model stencil as nodes, data dependencies as edges

simplified horizontal diffusion example

\[ a \oplus b = \{ a' + b' \mid a' \in a, b' \in b \} \]
Horizontal Diffusion Stencil Program tuned to Xeon Phi KNL

4th order horizontal diffusion: $-\alpha \nabla^2 (\nabla^2 u)$

2 inputs
1 output
4 stencils

3.7x improvement

Work performed at the Intel Parallel Computing Center at ETH Zurich
Vertical Advection Stencil Program tuned to Xeon Phi KNL

Vertical stencil-like operations, derived from Iterations of tridiagonal solvers (Thomas algorithm)

\[
\begin{bmatrix}
  b_1 & c_1 & 0 \\
  a_2 & b_2 & c_2 & 0 \\
  & a_3 & b_3 & \ddots & \ddots \\
  & & & \ddots & a_{n-1} & c_{n-1} & 0 \\
  & & & & a_n & b_n \\
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_{n-1} \\
  x_n \\
\end{bmatrix}
= 
\begin{bmatrix}
  d_1 \\
  d_2 \\
  \vdots \\
  d_{n-1} \\
  d_n \\
\end{bmatrix}
\]

4.2x improvement

Work performed at the Intel Parallel Computing Center at ETH Zurich
Scientific performance engineering for complex memory systems

Questions/Discussions?
Backup
Sorting in complex memories

Check! Let’s do something “Big Data”!

We’ll need a parallel sort on all cores, right?

Triad Cache Mode SNC4

Triad Flat Mode SNC4
Memory model: Bitonic Mergesort

- Slices of 16 elements go through a bitonic network.
- Communication: CPU₀ accesses data from local and remote caches.
- Synchronization: CPU₀ waits for CPU₁.
- Memory accesses: latency vs. bandwidth.
Modeling Bitonic Sorting

\[ C_{L1}(n) = [\log_2(n) - 1]2n \cdot \text{cost}_{L1} + 2n \cdot \text{cost}_{mem} \]

\[ C_{L2}(n) = \frac{n}{n_{L1}} C_{L1}(n_{L1}) + \left[ \log_2(n) - \log_2(n_{L1}) \right]2n \cdot \text{cost}_{L2} \]

\[ C_{mem}(n) = \frac{n}{n_{L2}} C_{L2}(n_{L2}) + \left[ \log_2(n) - \log_2(n_{L2}) \right]2n \cdot \text{cost}_{mem} \]
Bitonic Sort of 1 kiB

Synchronization outweights memory costs for small data!
So don’t parallelize too much!

(a) Sorting 1 KB of integers.
Bitonic Sort of 4 MiB

Latency (seconds)

Number of threads

- Mem. model Lat.
- Mem. model BW
- Full model Lat.
- Full model BW
- Measured

"parallelization boundary"

model including synchronization cost

model (just) memory costs
Bitonic Sort of 1 GiB

Always use all cores!

synchronization negligible
The most surprising result last ...

Hey, but which memory? DRAM or MCDRAM?

The model (and practical measurements) indicate that it does not matter.

Thesis: the higher bandwidth of MCDRAM did not help due to the higher latency ($\log^2 n$ depth).

Disclaimer: this is NOT the best sorting algorithm for Xeon Phi KNL. It is the best we found with limited effort. We suspect that a combination of algorithms will perform best.