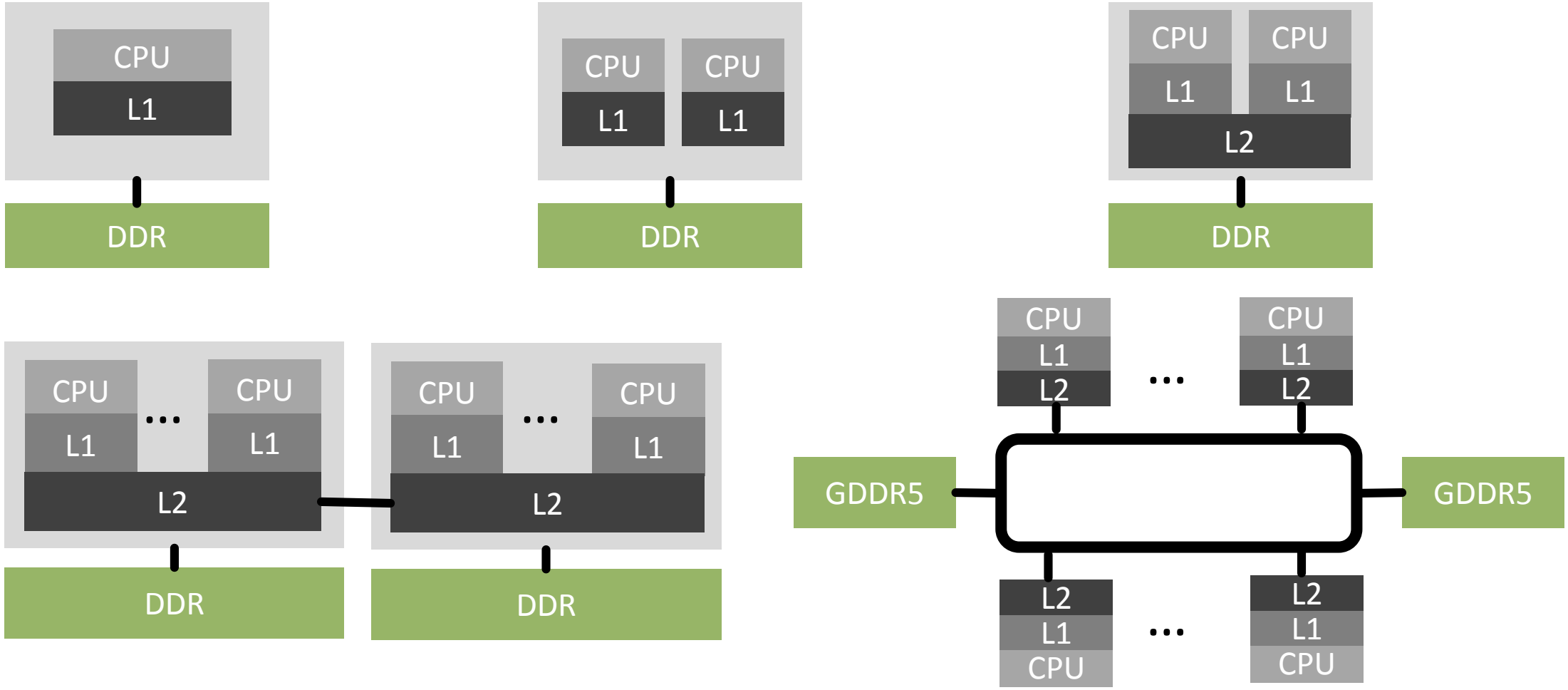
TORSTEN HOEFLER, SABELA RAMOS, TAL BEN-NUN, AND SPCL'S DAPP TEAM

HPC Performance Optimization Advances at Extreme Scale

Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL

Microarchitectures are becoming more and more complex



How to optimize codes for these complex architectures?

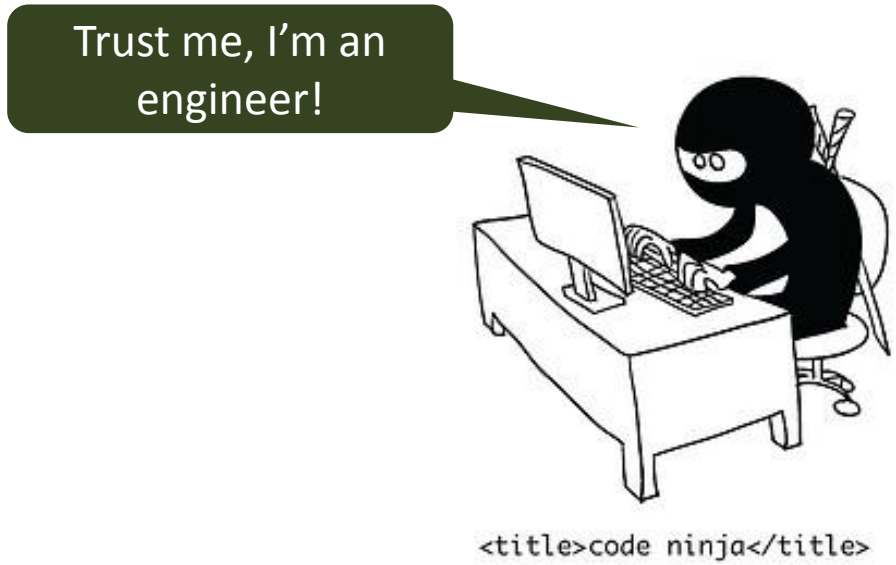
- **Performance engineering**: ***"encompasses the set of roles, skills, activities, practices, tools, and deliverables applied at every phase of the systems development life cycle which ensures that a solution will be designed, implemented, and operationally supported to meet the non-functional requirements 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.)

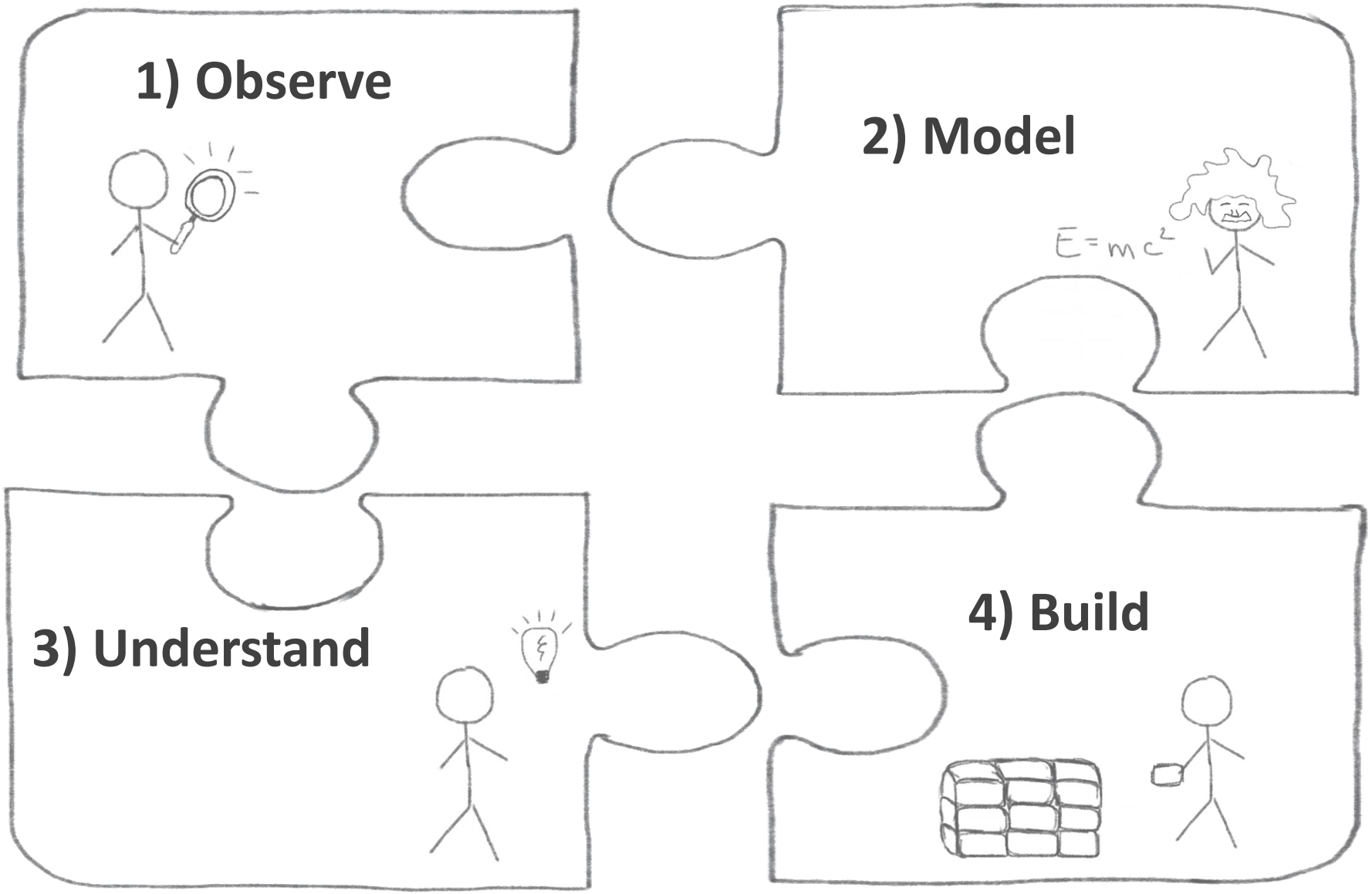
    Caches (associativity, sizes etc.)

    Microarchitecture (number of outstanding loads etc.)



An engineering example – Tacoma Narrows Bridge

Scientific Performance Engineering



Scientific Performance Engineering

Application Model

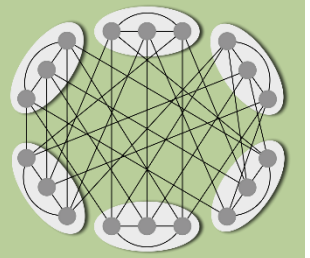
$$f(p) = \sum_{k=1}^{n} c_k \times p^{i_k} \times \log_2^{j_k}(p)$$

- Automatic application modeling for optimization and co-design

Calotoiu et al., SC'14

Network Model

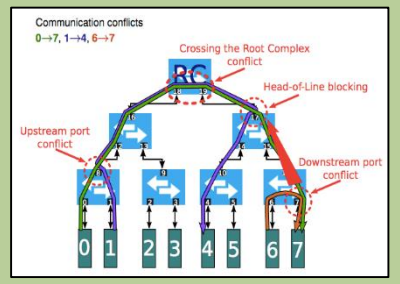
- Close-to-optimal network topology design



Besta et al., SC'14

Compute Node Model

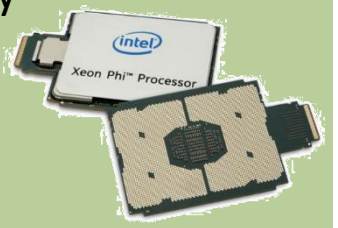
- PCIe model
- Used for multi-GPU system



Martinasso et al., SC'16

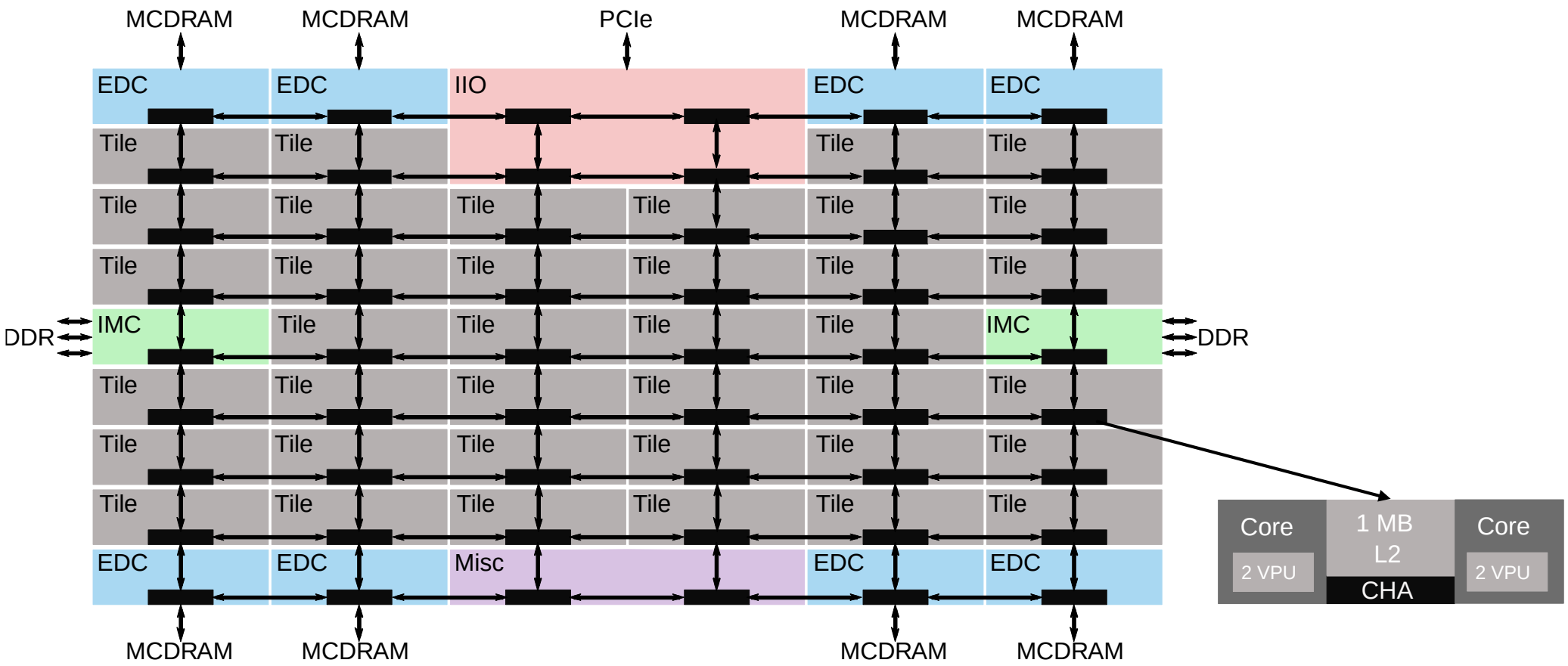
CPU Model

- Detailed memory architecture models for development

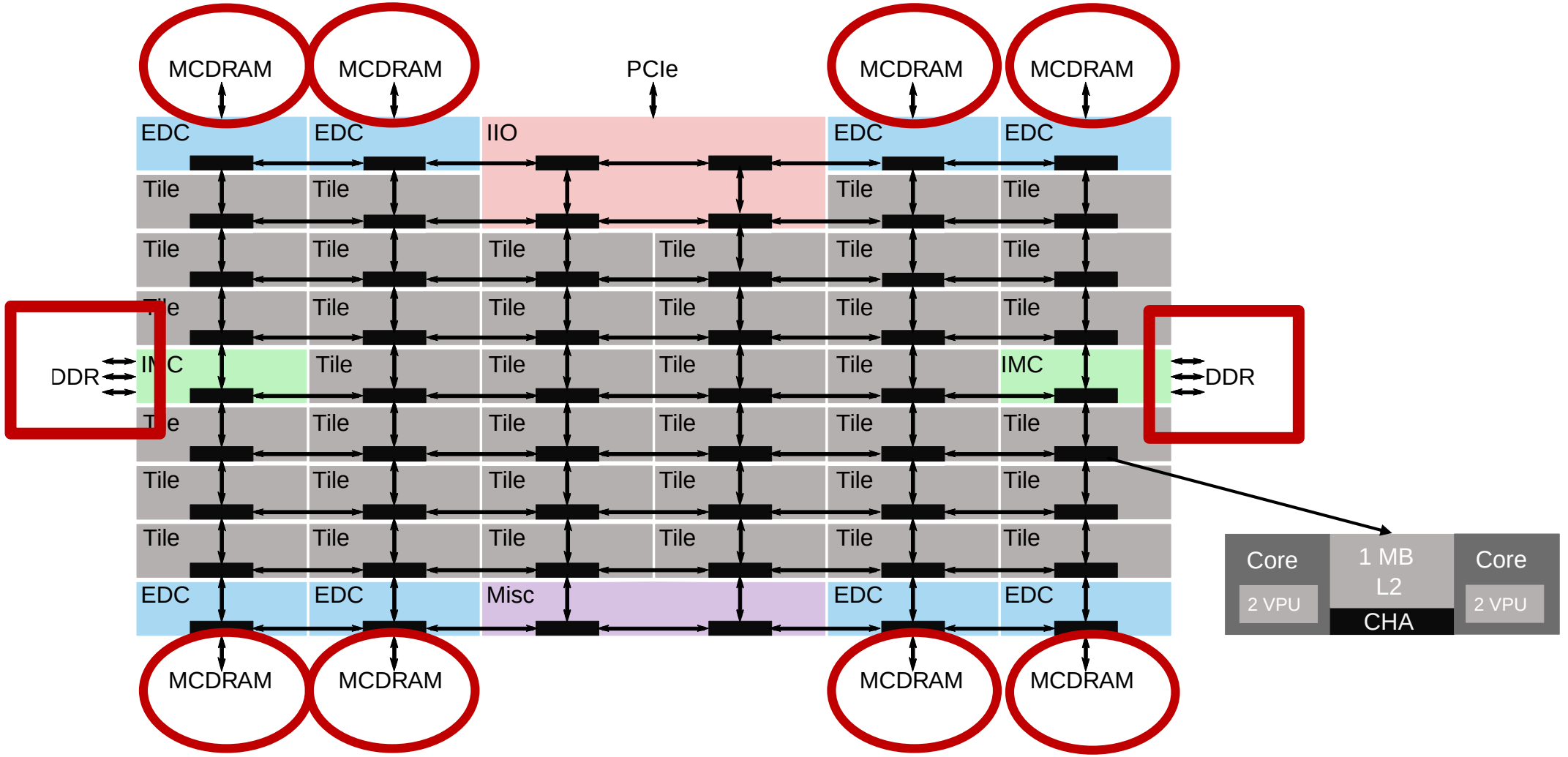


Ramos et al., IPDPS'17

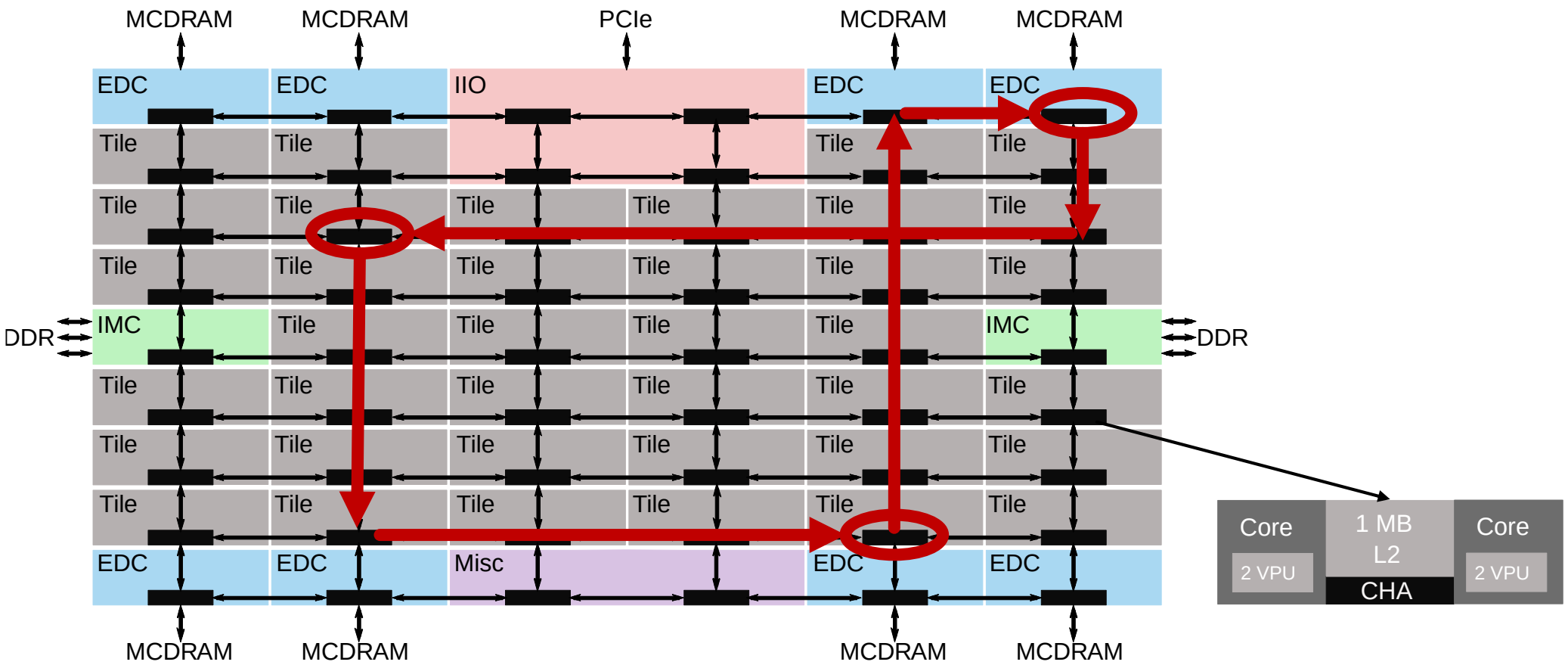
Modeling by example: KNL Architecture (mesh)



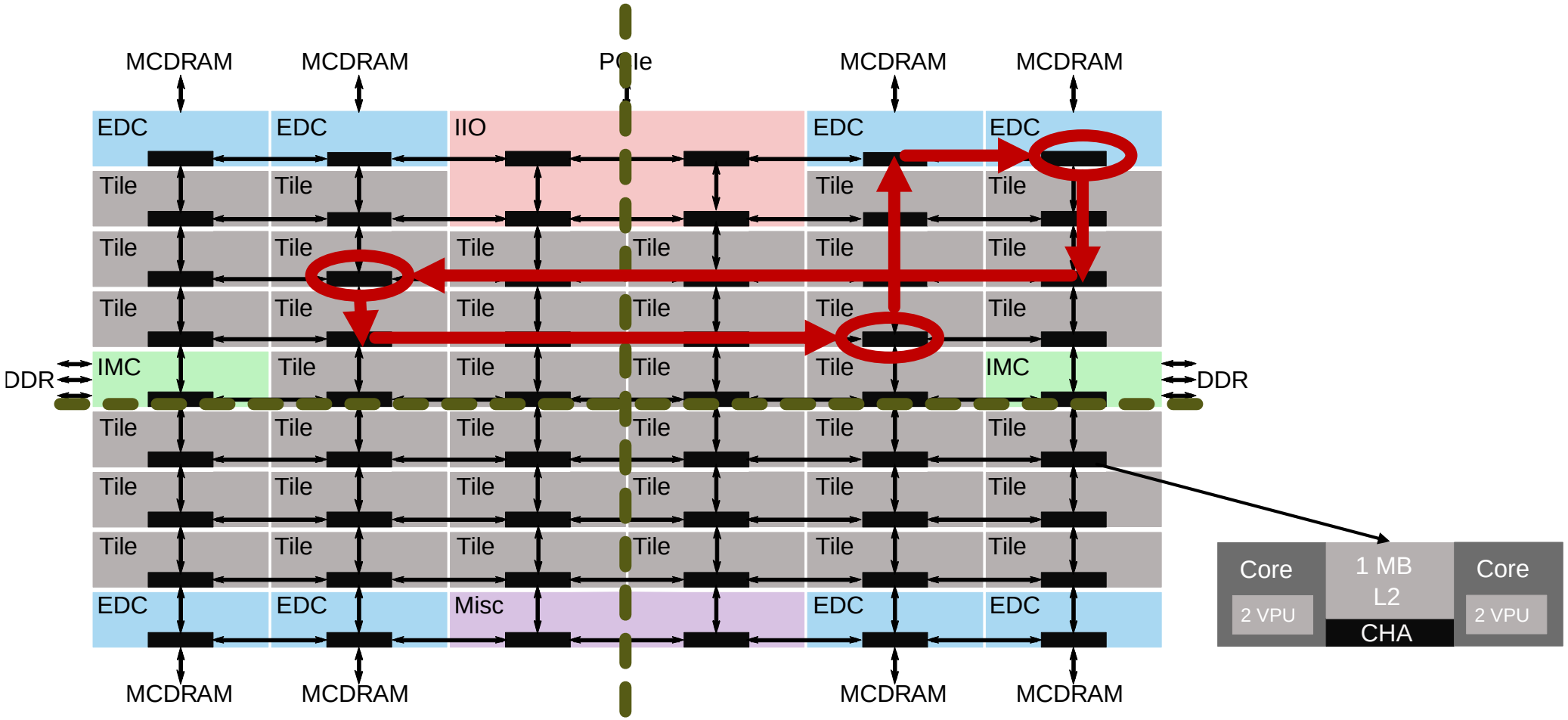
KNL Architecture (memory: Flat & Cache)



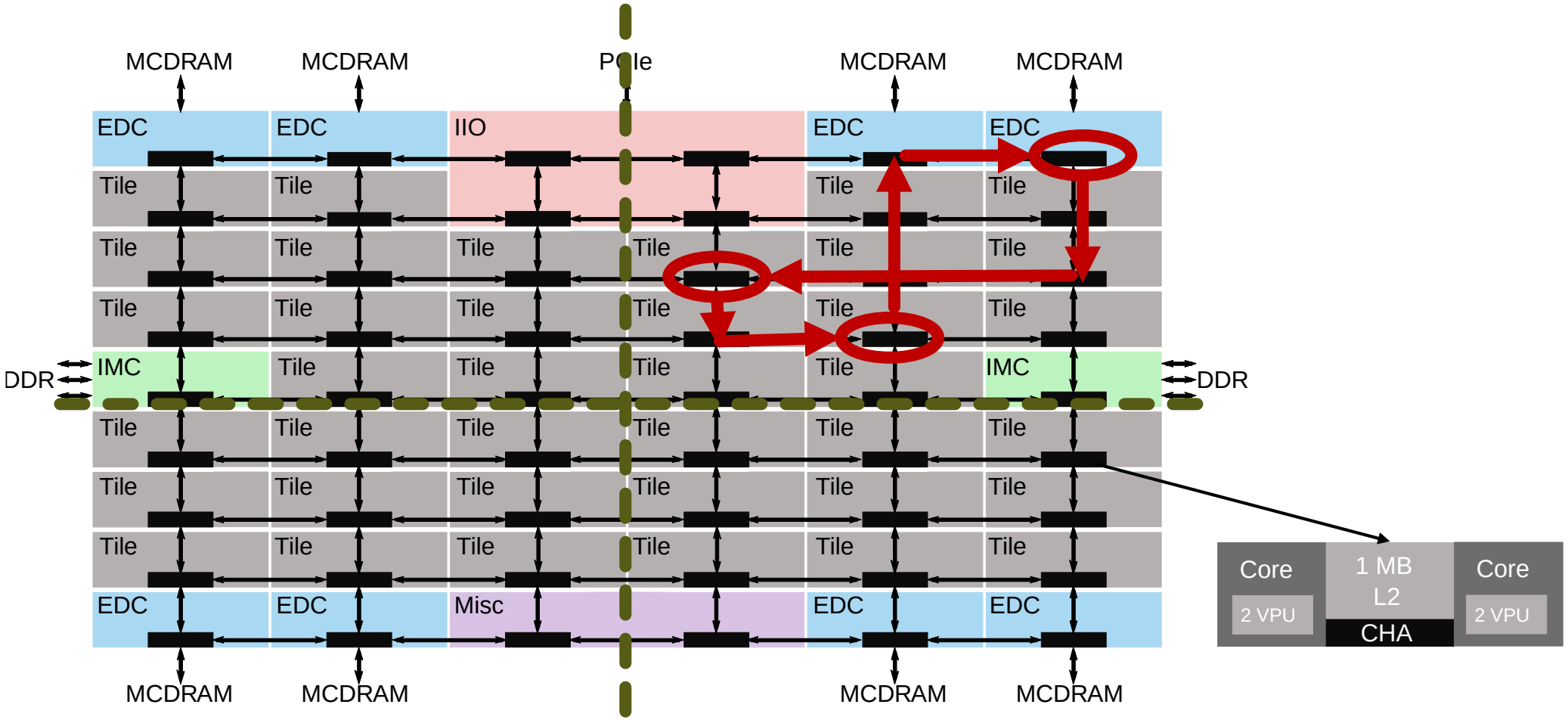
KNL Architecture (all to all mode)



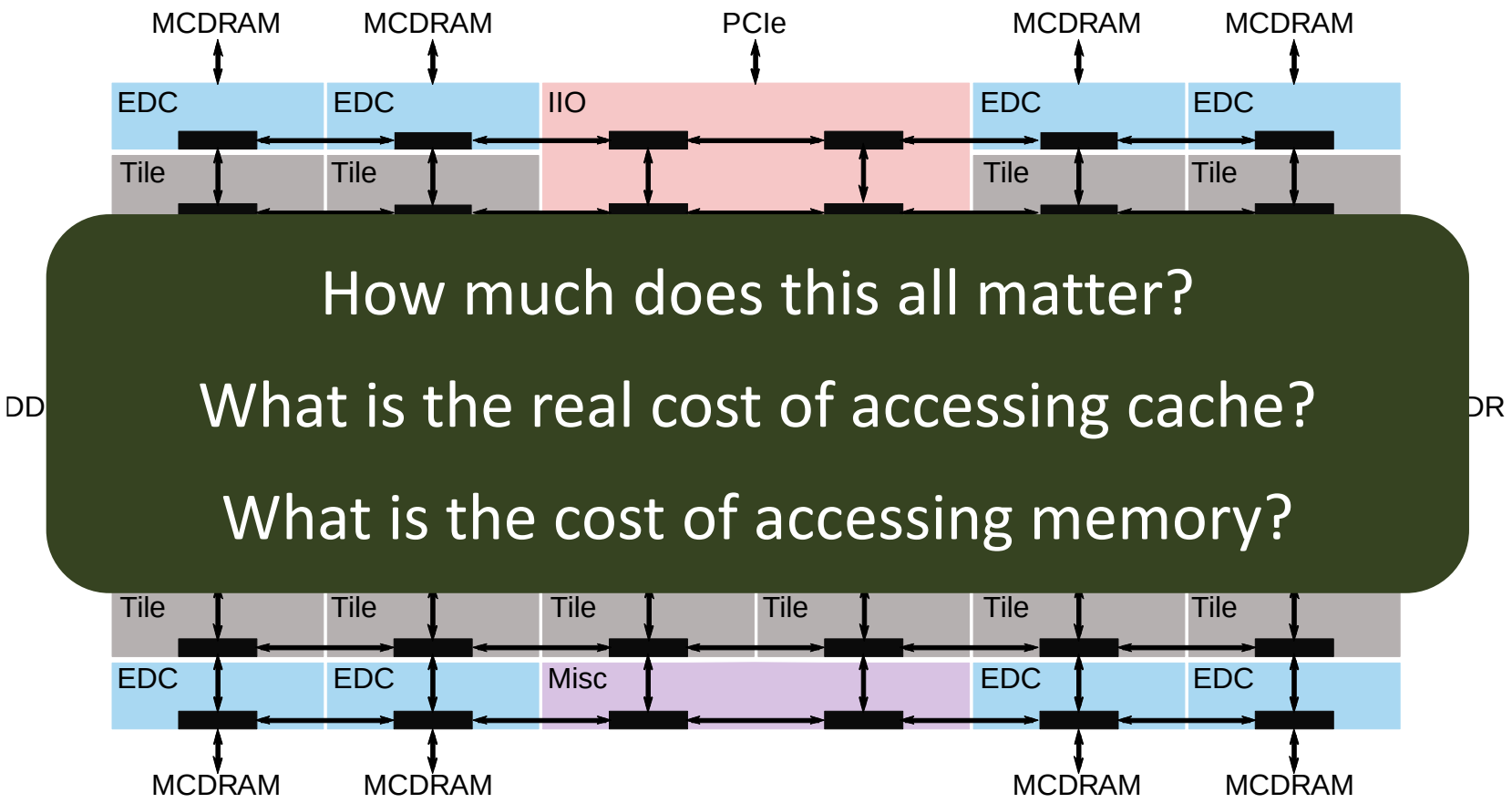
KNL Architecture (Quadrant or Hemisphere)



KNL Architecture (SNC-4 or SNC-2)



KNL Architecture



Step 1: Understand core-to-core transfers – MESIF cache coherence

Write back overhead

Location: only 5-15% difference

Contention effects?

That is curious!

		SNC4	SNC2	QUAD	HEM	A2A
Latency [ns] (Copy/BenchIT)	Local (L1)	3.8	3.8	3.8	3.8	3.8
	Tile (L2)	34 (M) 17 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)
	Remote	107-122 (M) 98-114 (E) 96-118 (S,F)	111-125 (M) 104-117 (E) 104-118 (S,F)	119 (M) 116 (E) 107-117 (S,F)	120 (M) 116 (E) 107-117 (S,F)	122 (M) 116 (E) 109-117 (S,F)
Bandwidth [GB/s] (Copy)	Tile	6.7 (M) 7.6 (E)	6.7 (M) 6.7 (E)	7.5 (M) 9.2 (E)	7.4 (M) 9.2 (E)	7.5 (M) 9.2 (E)
	Remote	7.7	6.7	7.5	7.5	7.5
Contention [ns] (1:N copy)	α	200	200	200	200	200
Linear, $\mathcal{T}_C(N) = \alpha + \beta \cdot N$	β	34	34	34	34	34

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

		Software NUMA		Software UMA		
		SNC4	SNC2	QUAD	HEM	A2A
Latency [ns] (BenchIT)	DRAM	130-140	134-146	140	140	139
	MCDRAM	160-175	160-170	167	167	168
Bandwidth [GB/s] (Copy NT / STREAM Copy)	DRAM	69 / 77	69 / 77	70 / 77	71 / 77	71 / 77
	MCDRAM	342 / 418	333 / 388	333 / 415	315 / 372	306 / 359
Bandwidth [GB/s] (Read)	DRAM	71	71	77	77	77
	MCDRAM	243	288	314	314	314
Bandwidth [GB/s] (Write)	DRAM	33	34	36	36	36
	MCDRAM	147	163	171	165	161
Bandwidth [GB/s] (Triad NT / STREAM Triad)	DRAM	71 / 82	71 / 82	74 / 82	73 / 82	73 / 82
	MCDRAM	371 / 448	347 / 441	340 / 441	332 / 434	325 / 427
Latency [ns] (BenchIT)		158-178	161-171	166	168	172
Bandwidth [GB/s] (Copy NT / STREAM Copy)		150 / 252	130 / 252	175 / 255	134 / 237	132 / 233
Bandwidth [GB/s] (Read)		87	95	124	128	118
Bandwidth [GB/s] (Write)		56	56	72	72	68
Bandwidth [GB/s] (Triad NT / STREAM Triad)		296 / 292	246 / 294	296 / 309	273 / 274	264 / 269

MCDRAM 20% slower!

MCDRAM 4-6x faster!

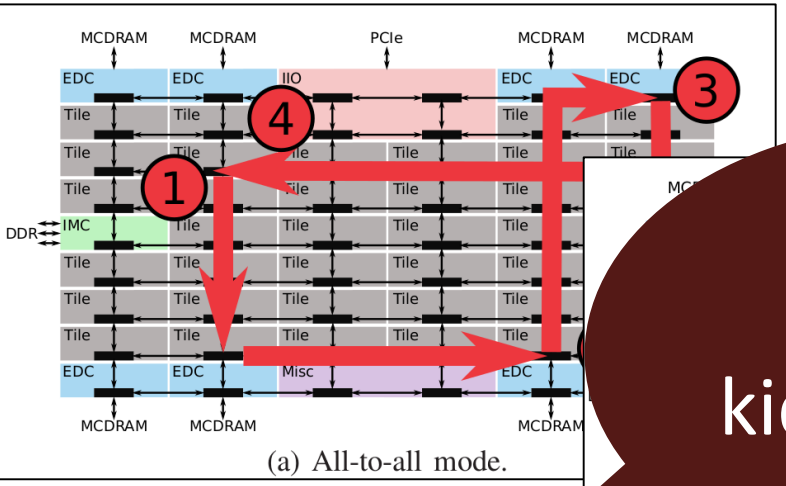
Need to read **and** write for full bandwidth

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

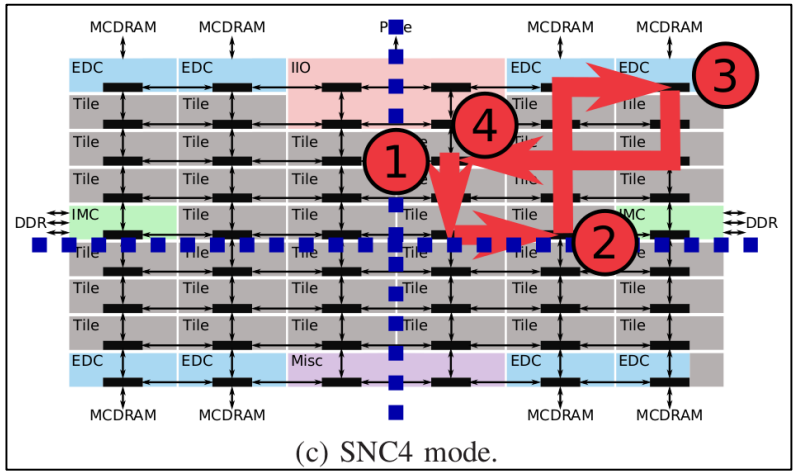
Performance engineers optimize your code!



(a) All-to-all mode.

(b) Quadrant mode.

Are you kidding me?



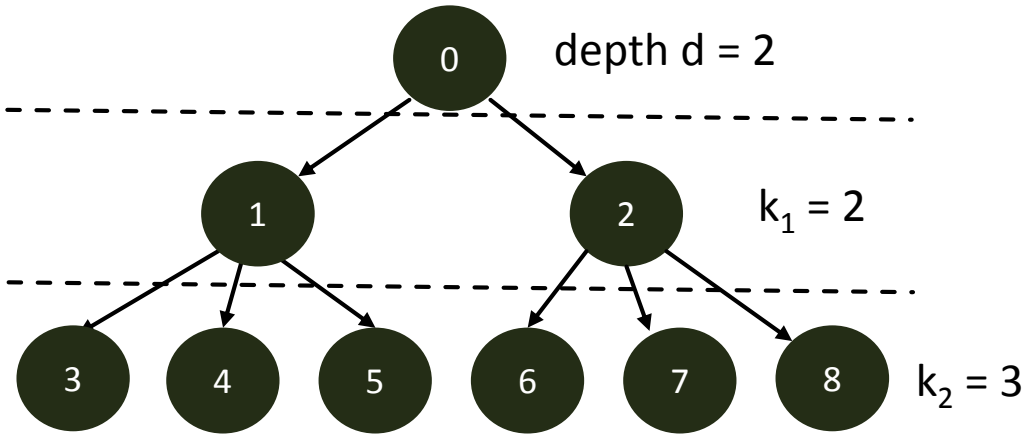
(c) SNC4 mode.

		Software NUMA		Software UMA		
		SNC4	SNC2	QUAD	HEM	A2A
Latency [ns] (Copy/BenchIT)	Local (L1)	3.8	3.8	3.8	3.8	3.8
	Tile (L2)	34 (M) 17 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)
	Remote	107-122 (M) 98-114 (E) 96-118 (S,F)	111-125 (M) 104-117 (E) 104-118 (S,F)	119 (M) 116 (E) 107-117 (S,F)	120 (M) 116 (E) 107-117 (S,F)	122 (M) 116 (E) 109-117 (S,F)
Bandwidth [GB/s] (Read)		2.5	2.5	2.5	2.5	2.5
Bandwidth [GB/s] (Copy)	Tile	6.7 (M) 7.6 (E)	6.7 (M) 6.7 (E)	7.5 (M) 9.2 (E)	7.4 (M) 9.2 (E)	7.5 (M) 9.2 (E)
	Remote	7.7	6.7	7.5	7.5	7.5
Congestion (P2P pairs)		None				
Contention [ns] (1:N copy) Linear, $T_C(N) = \alpha + \beta \cdot N$	α β	200 34	200 34	200 34	200 34	200 34

			Software NUMA		Software UMA		
			SNC4	SNC2	QUAD	HEM	A2A
Flat Mode	Latency [ns] (BenchIT)	DRAM MCDRAM	130-140 160-175	134-146 160-170	140 167	140 167	139 168
	Bandwidth [GB/s] (Copy NT / STREAM Copy)	DRAM MCDRAM	69 / 77 342 / 418	69 / 77 333 / 388	70 / 77 333 / 415	71 / 77 315 / 372	71 / 77 306 / 359
	Bandwidth [GB/s] (Read)	DRAM MCDRAM	71 243	71 288	77 314	77 314	77 314
	Bandwidth [GB/s] (Write)	DRAM MCDRAM	33 147	34 163	36 171	36 165	36 161
	Bandwidth [GB/s] (Triad NT / STREAM Triad)	DRAM MCDRAM	71 / 82 371 / 448	71 / 82 347 / 441	74 / 82 340 / 441	73 / 82 332 / 434	73 / 82 325 / 427
Cache Mode	Latency [ns] (BenchIT)		158-178	161-171	166	168	172
	Bandwidth [GB/s] (Copy NT / STREAM Copy)		150 / 252	130 / 252	175 / 255	134 / 237	132 / 233
	Bandwidth [GB/s] (Read)		87	95	124	128	118
	Bandwidth [GB/s] (Write)		56	56	72	72	68
	Bandwidth [GB/s] (Triad NT / STREAM Triad)		296 / 292	246 / 294	296 / 309	273 / 274	264 / 269

A principled approach to designing cache-to-cache broadcast algorithms

Multi-ary tree example



Tree depth

Level size

$$\mathcal{T}_{tree} = \sum_{i=1}^{d} \mathcal{T}_C(k_i) = \sum_{i=1}^{d} (c \cdot k_i + b)$$

Tree cost

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

$$\mathcal{T}_{sbcast} = \min_{d,k_i} \left(\mathcal{T}_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} \mathcal{T}_{nb}(k_i + 1) \right)$$

Reached threads

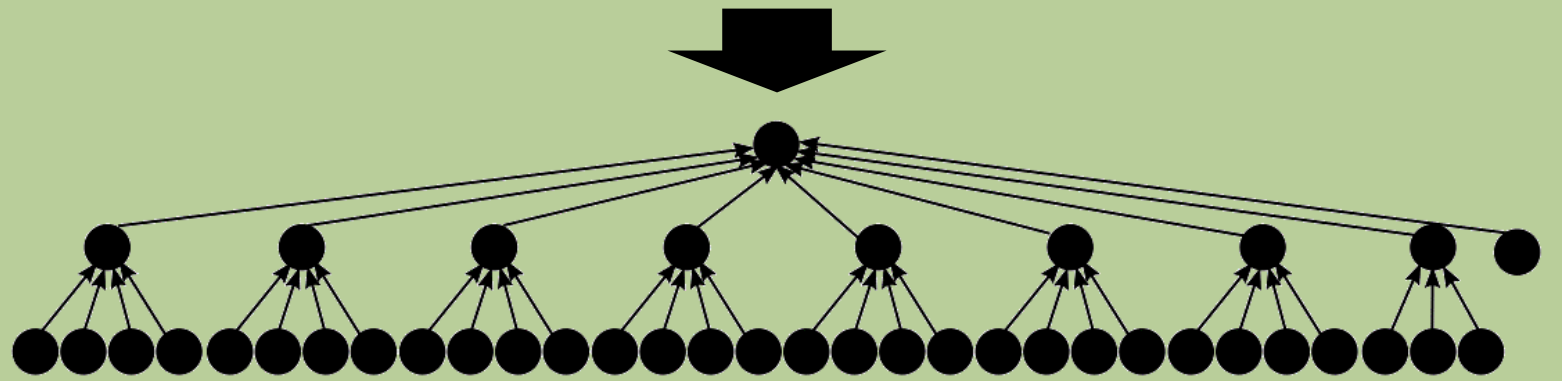
$$N \leq 1 + \sum_{i=1}^{d} \prod_{j=1}^{i} k_j, \quad \forall i < j, k_i \leq k_j$$

S. Ramos, TH: *"Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", ACM HPDC'13*
S. Ramos, TH: *"Cache line aware optimizations for ccNUMA systems (IEEE TPDS'17).*

Model-driven performance engineering for broadcast

$$\mathcal{T}_{sbcast} = \min_{d,k_i} \left(\mathcal{T}_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} \mathcal{T}_{nb}(k_i + 1) \right)$$

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

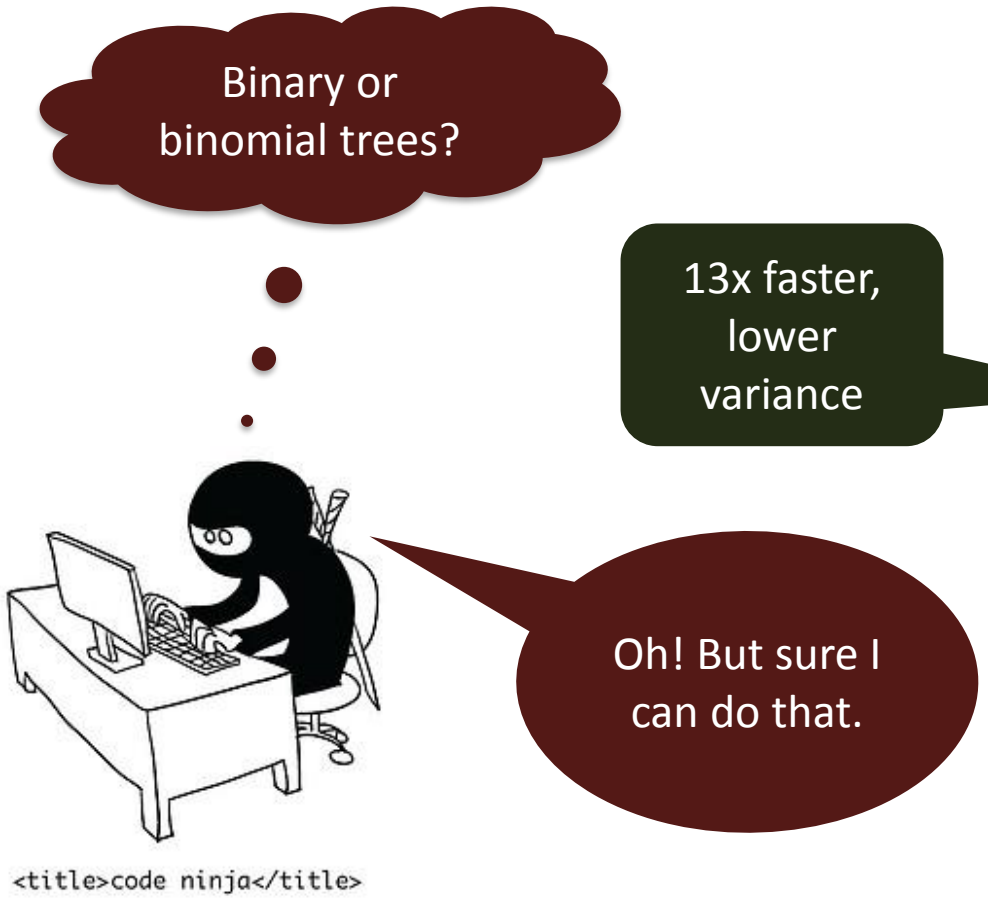


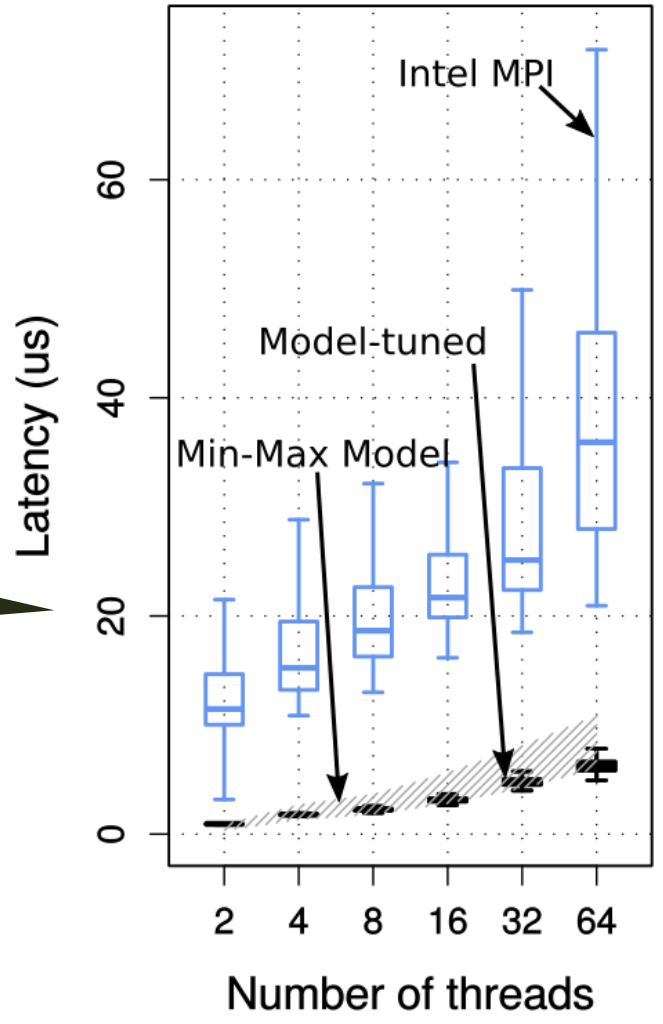
Binary or binomial trees?

Oh! But sure I can do that.

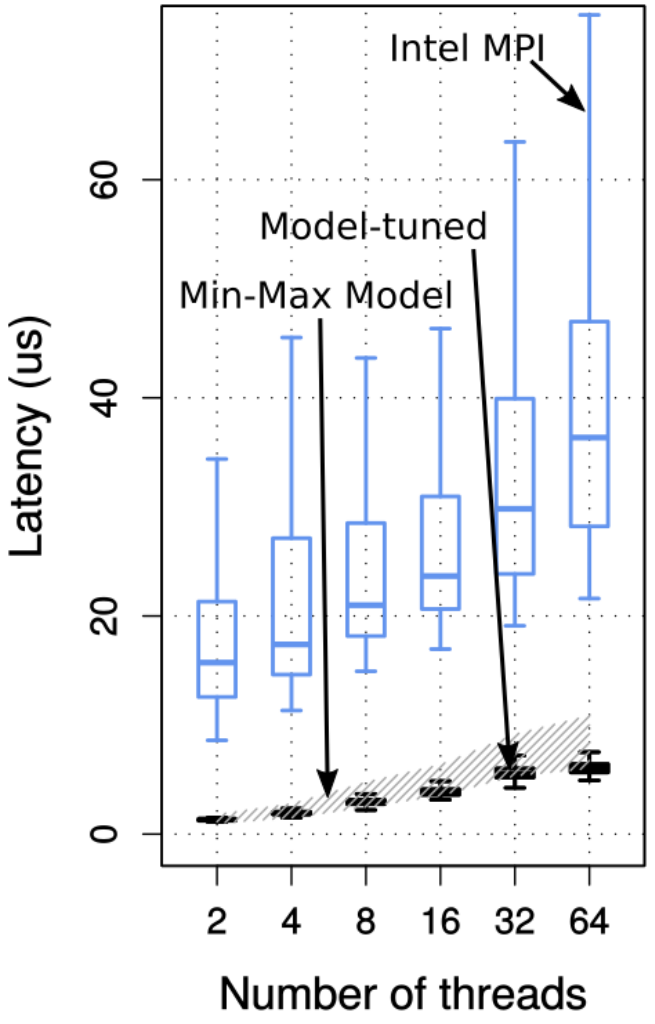
<title>code ninja</title>

Model-driven performance engineering for broadcast



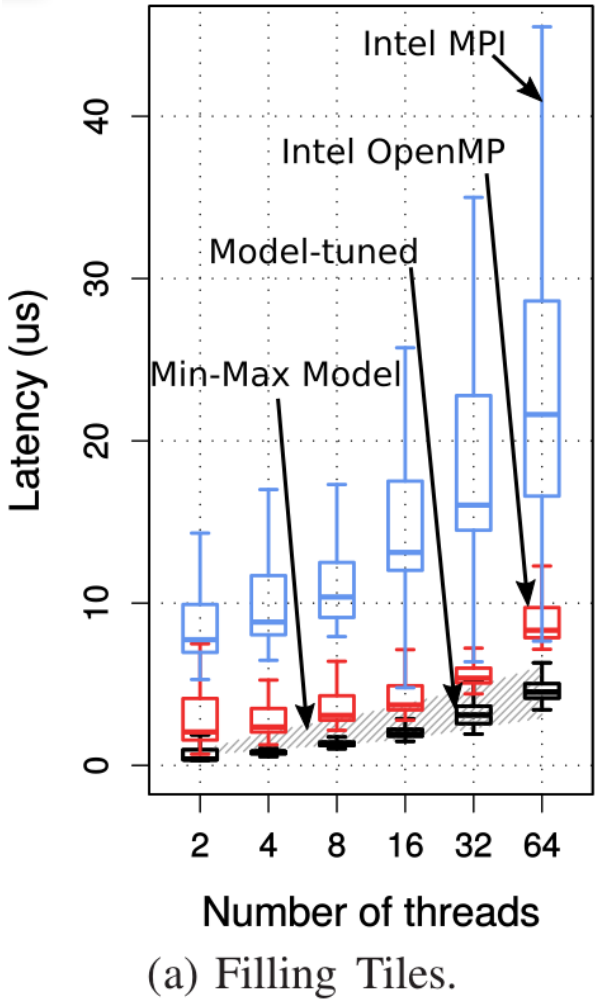


(a) Filling Tiles.

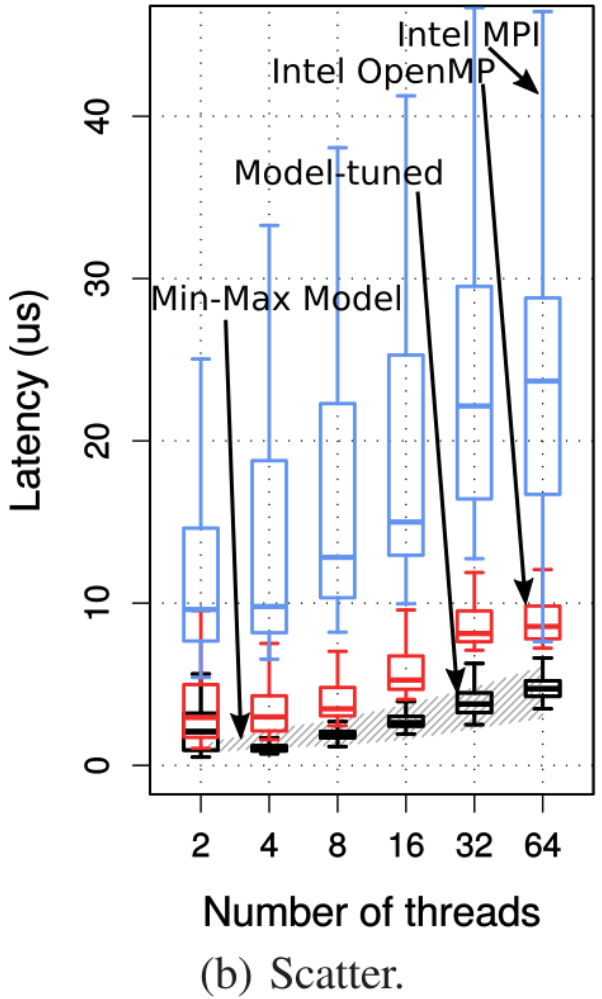


(b) Scatter.

Easy to generalize to similar algorithms

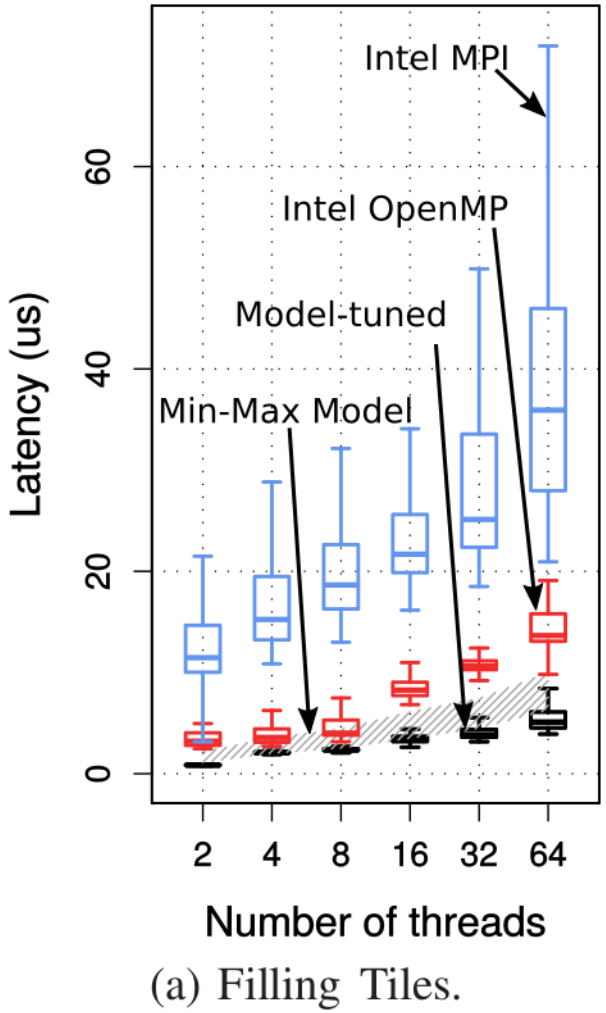


(a) Filling Tiles.

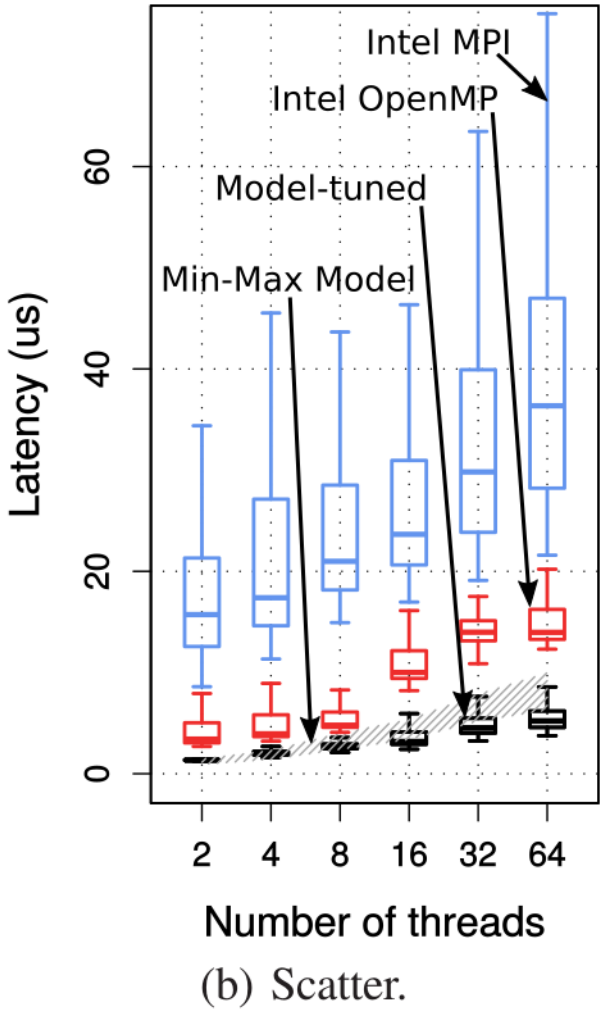


(b) Scatter.

Barrier (7x faster than OpenMP)



(a) Filling Tiles.

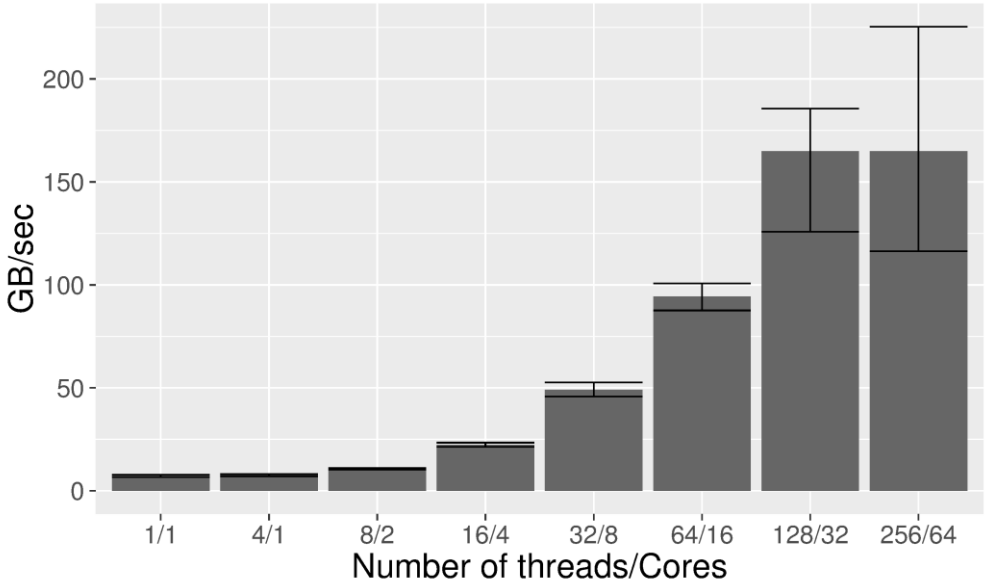


(b) Scatter.

Reduce (5x faster then OpenMP)

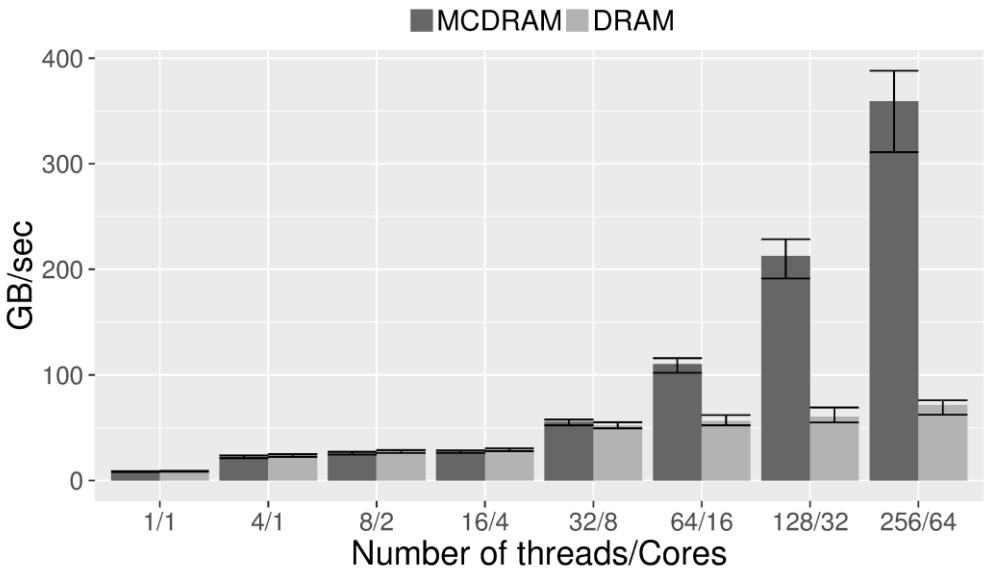
Sorting in complex memories

Triad
Cache Mode
SNC4



Check! Let's do something "Big Data"!

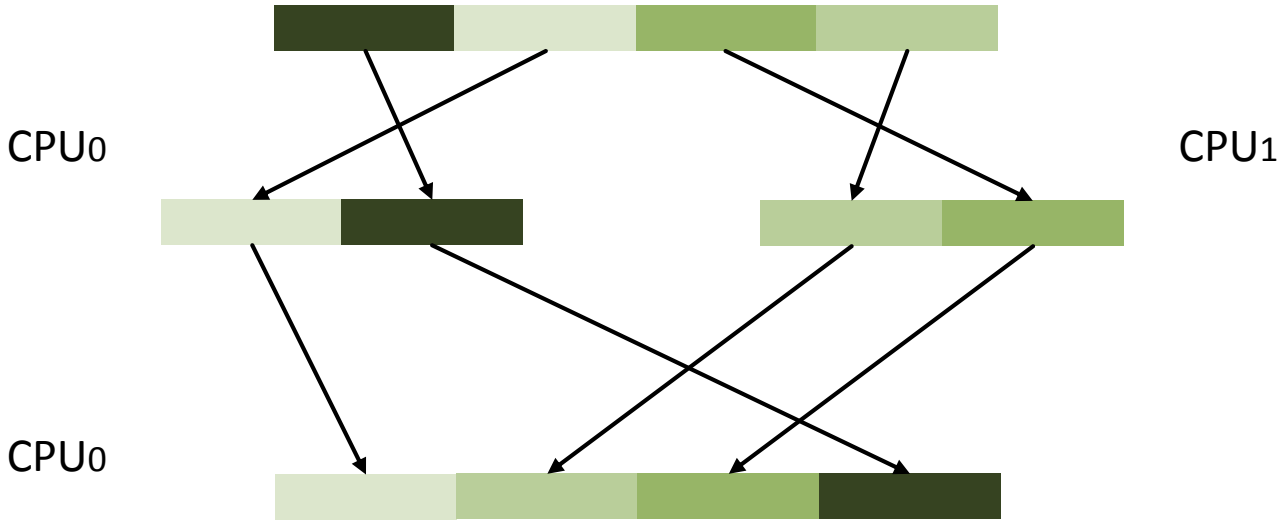
Triad
Flat Mode
SNC4



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

<title>code ninja</title>

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

L1 access cost

memory access cost

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

elements in L1

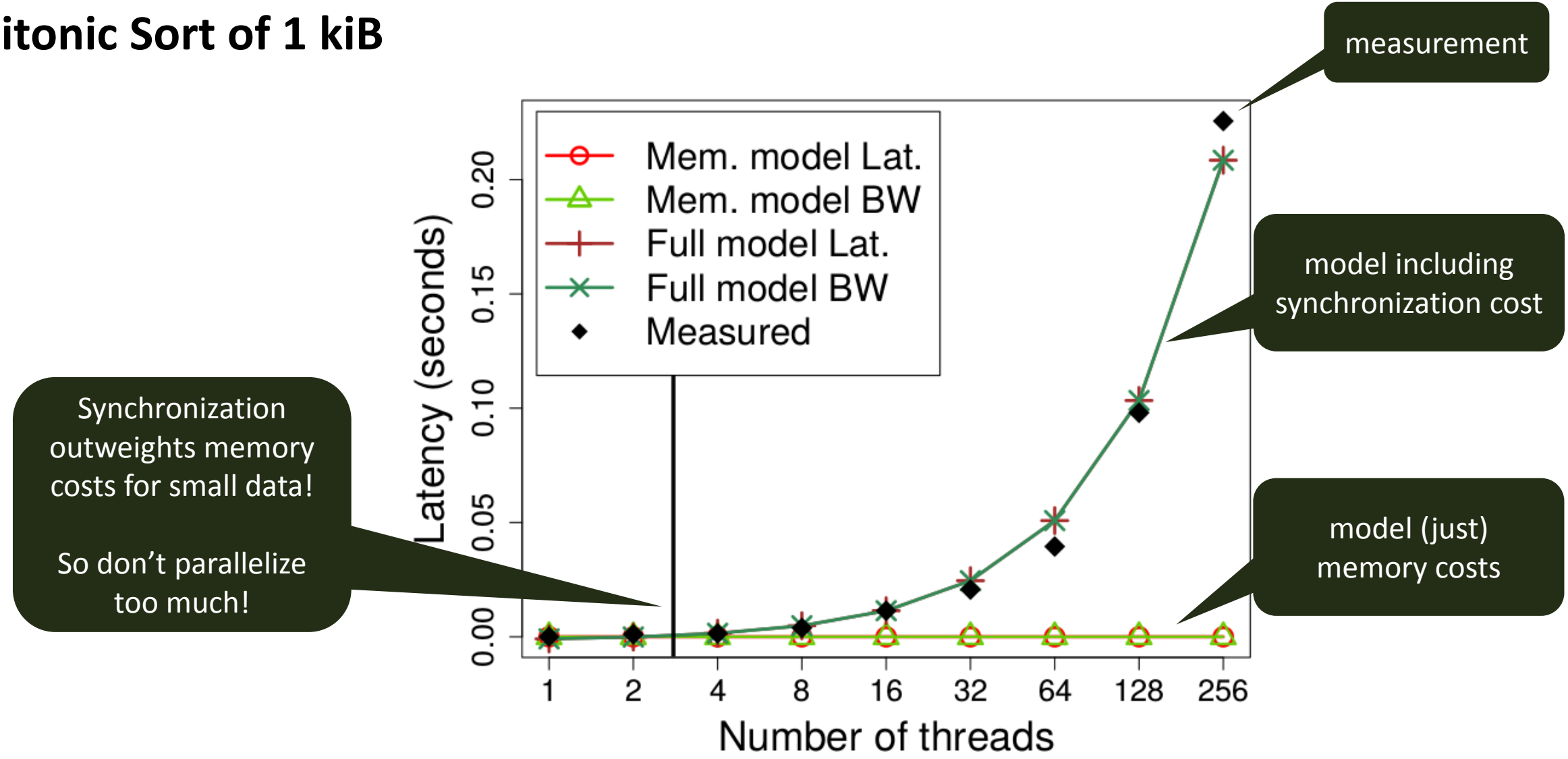
L2 access cost

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

elements in L2

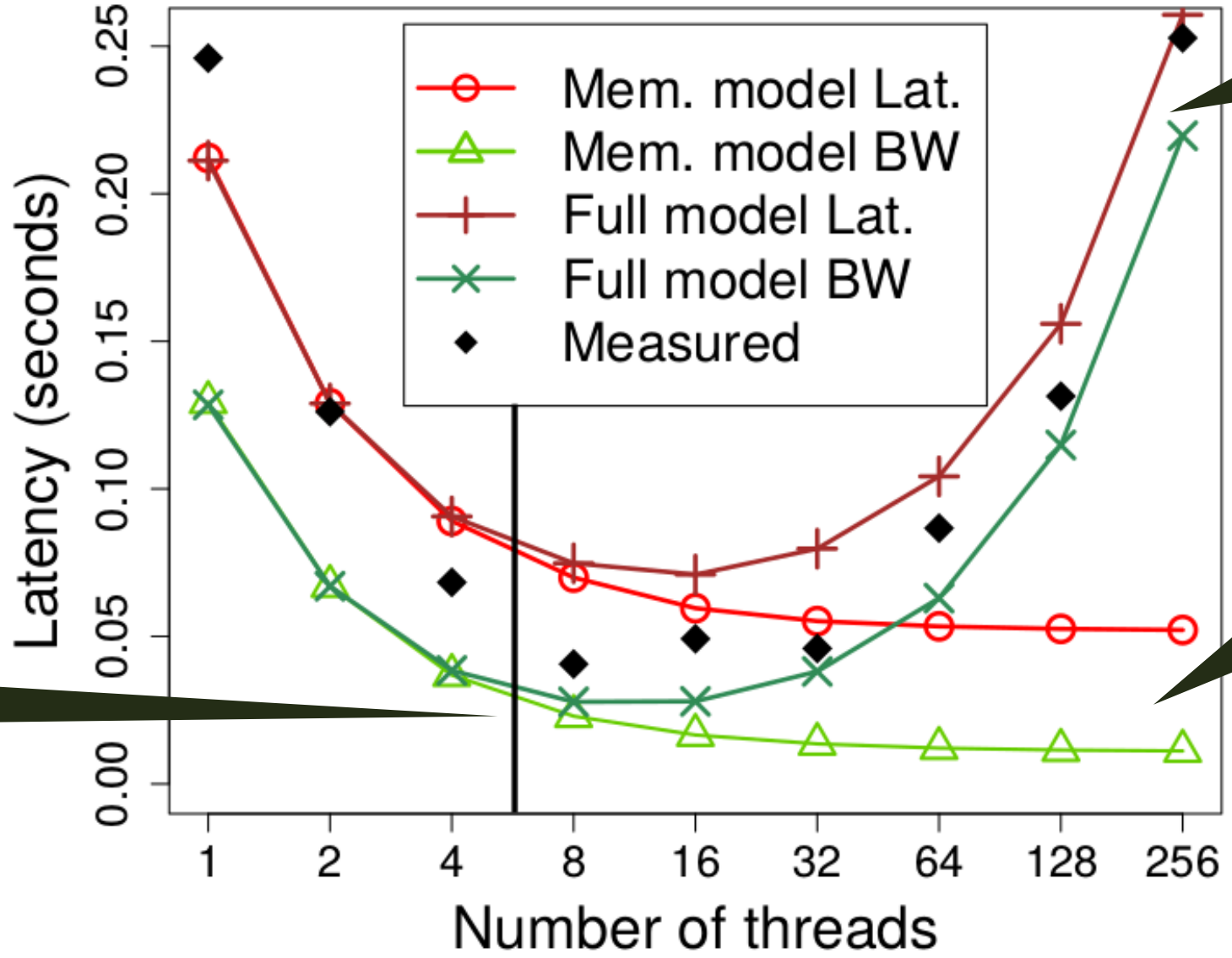
$$C_{mem}(n) = \frac{n}{n_{L2}} C_{L2}(n_{L2}) + [\log_2(n) - \log_2(n_{L2})]2n \cdot \text{cost}_{mem}$$

Bitonic Sort of 1 kiB



(a) Sorting 1 KB of integers.

Bitonic Sort of 4 MiB

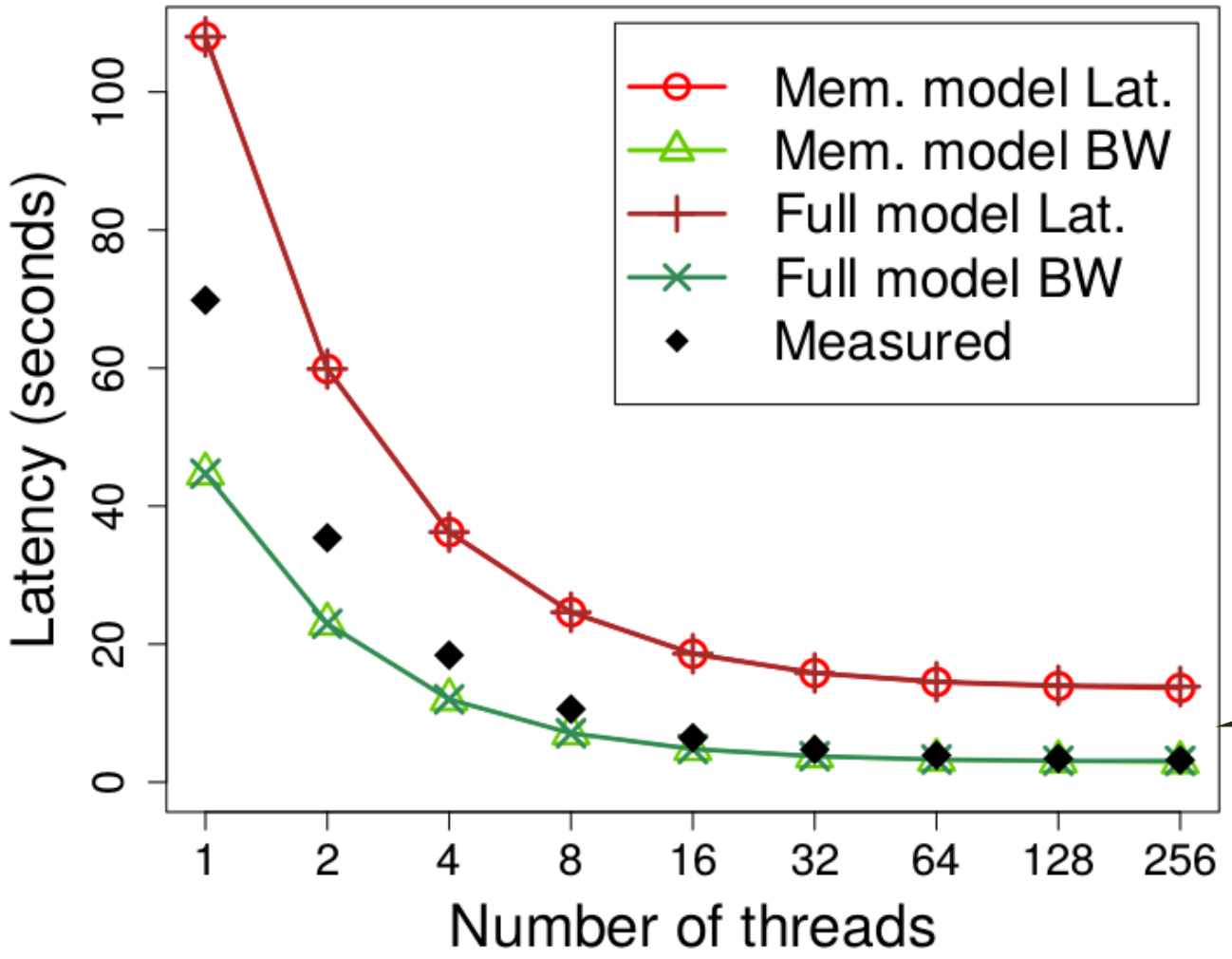


model including synchronization cost

model (just) memory costs

"parallelization boundary"

Bitonic Sort of 1 GiB



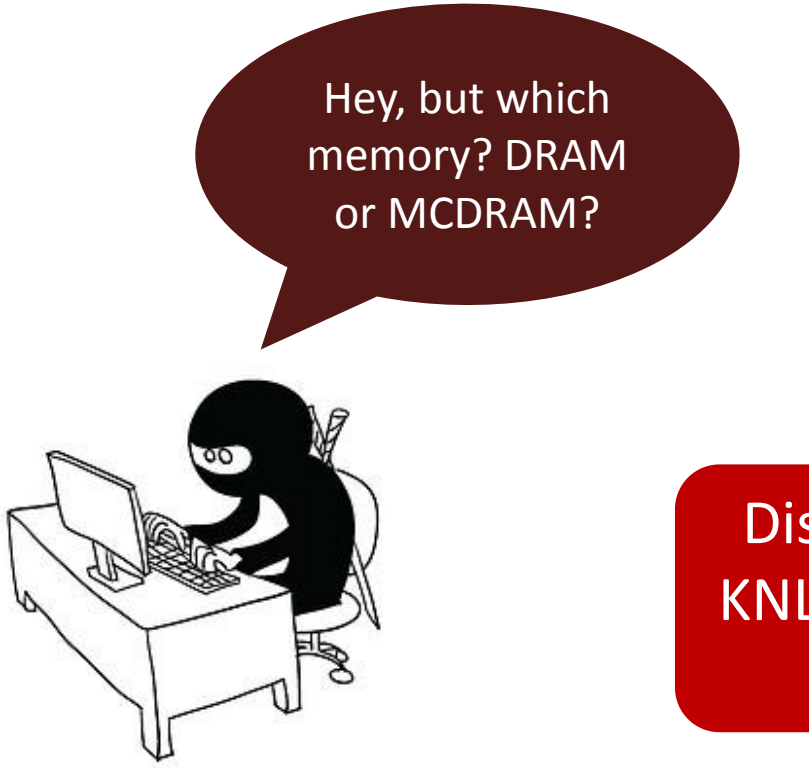
Always use all cores!

synchronization negligible

The most surprising result last …

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.

But what now? Is that it?

Not really, we need to enable large-scale applications!

DAPPy – Data-centric Parallel Programming for Python

- **Memory access decoupled from computation**
- **Programs are composed of Tasklets and Memlets**
  - Tasklets wrapped by simple primitives: Map, Iterate, Reduce
  - Hide communication, caching and data-movement
- **Easy-to-integrate Python programming interface**
- **Graph-based compilation pipeline**

```python
@dapp.program
def gemm(A, B, C):
    # local definitions

    @dapp.map(_[0:M, 0:K, 0:N])
    def multiplication(i, j, k):
        in_A << A[i,k]
        in_B << B[k,j]
        out  >> tmp[i,j,k]

        out = in_A * in_B

    @dapp.reduce(tmp, C, axis=2)
    def sum(a,b):
        return a+b
```

DAPPy Compilation Infrastructure

Code

```
@dapp.program
def program(A, B):

  @dapp.map(_[0:N,0:M])
  def transpose(i, j):
    a << A[i,j]
    b >> B[j,i]

  ...
```

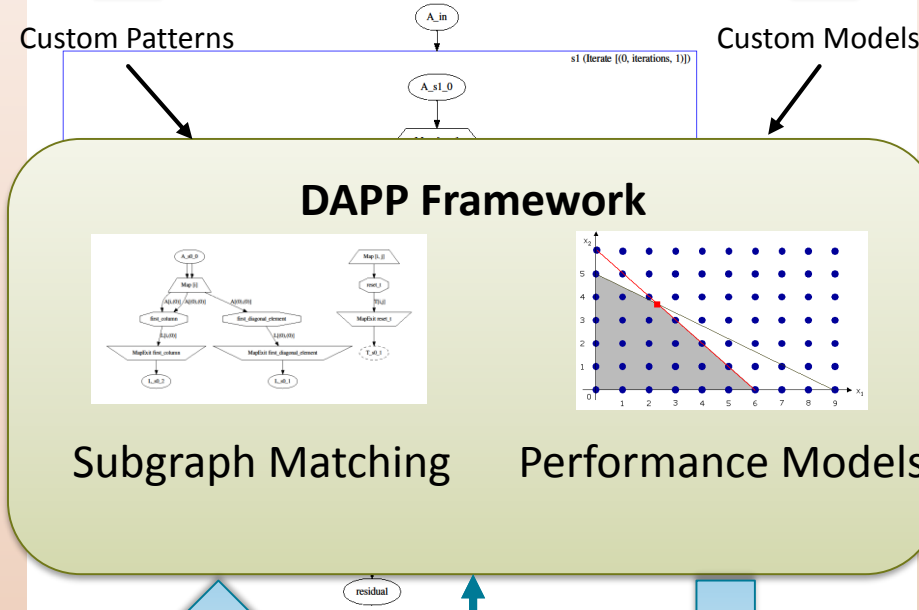
dappy Program

AST Analysis

Specialization

Custom Patterns

Custom Models



DAPP Framework

Subgraph Matching

Performance Models

S eful Dataflow Graph (SD)

Domain Programmer

Performance Engineer

Partitioning, Scheduling

Runtime

CPU Library

GPU Library

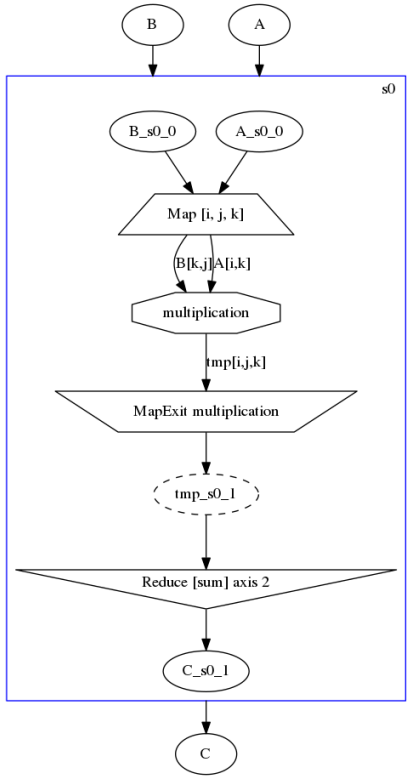
FPGA Modules

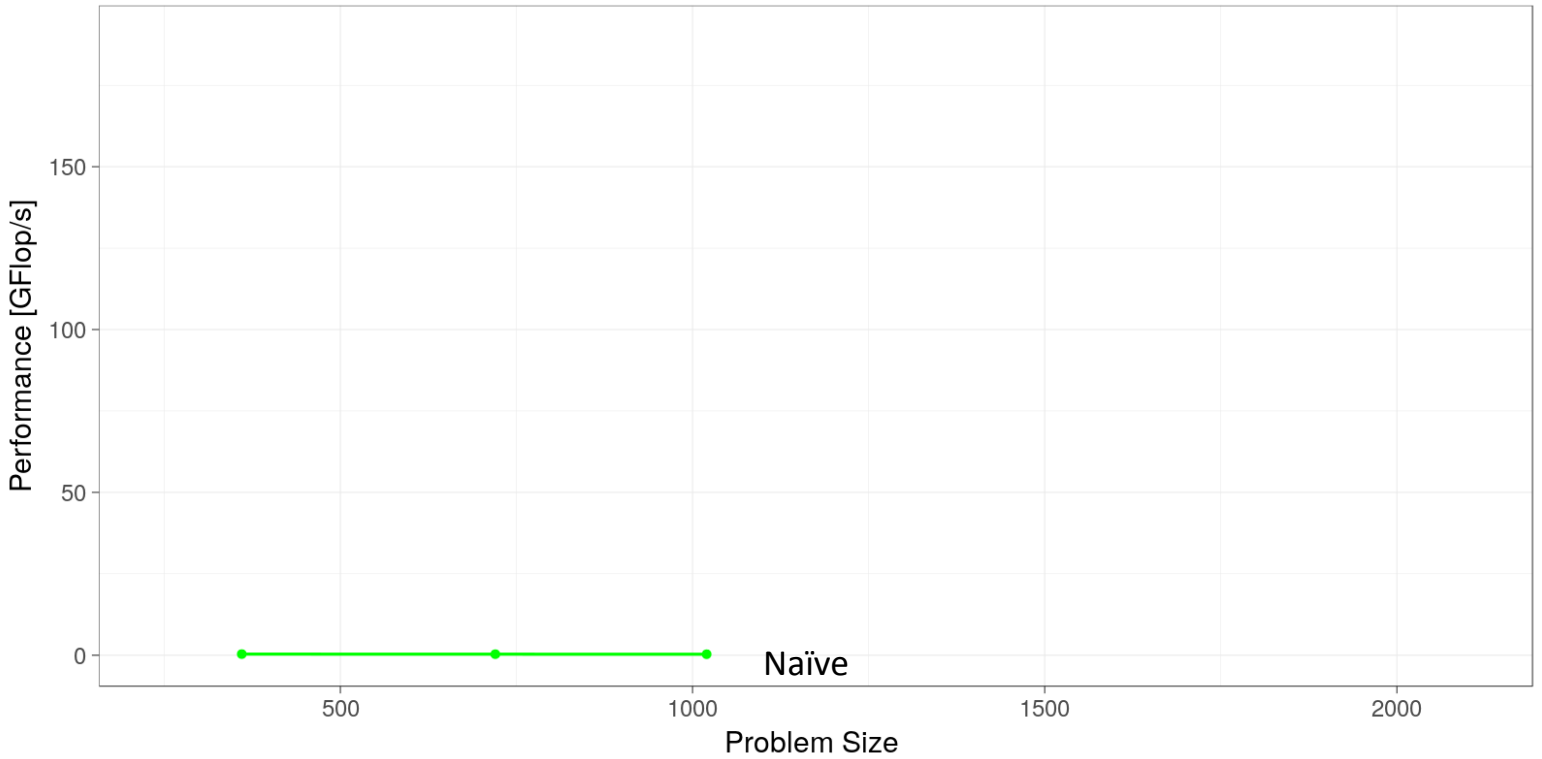
System Probe

Microbenchmarks

Architecture Expert

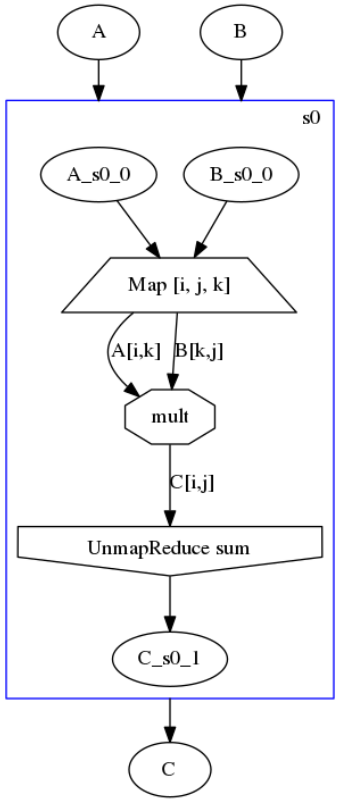
Performance



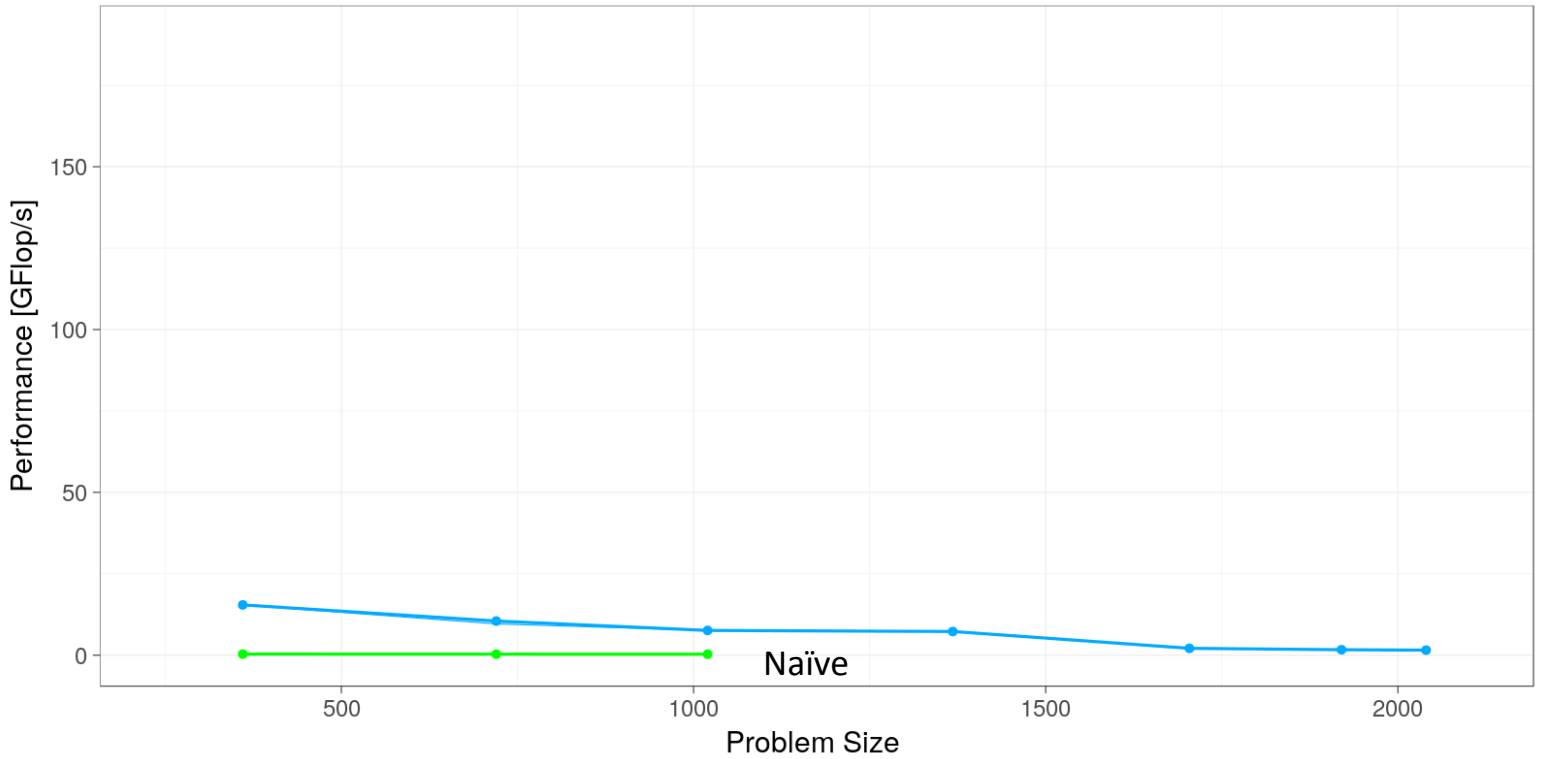


SDFG

Performance

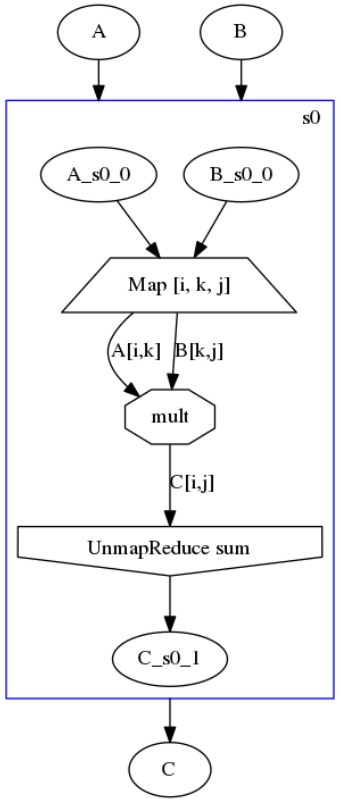


SDFG

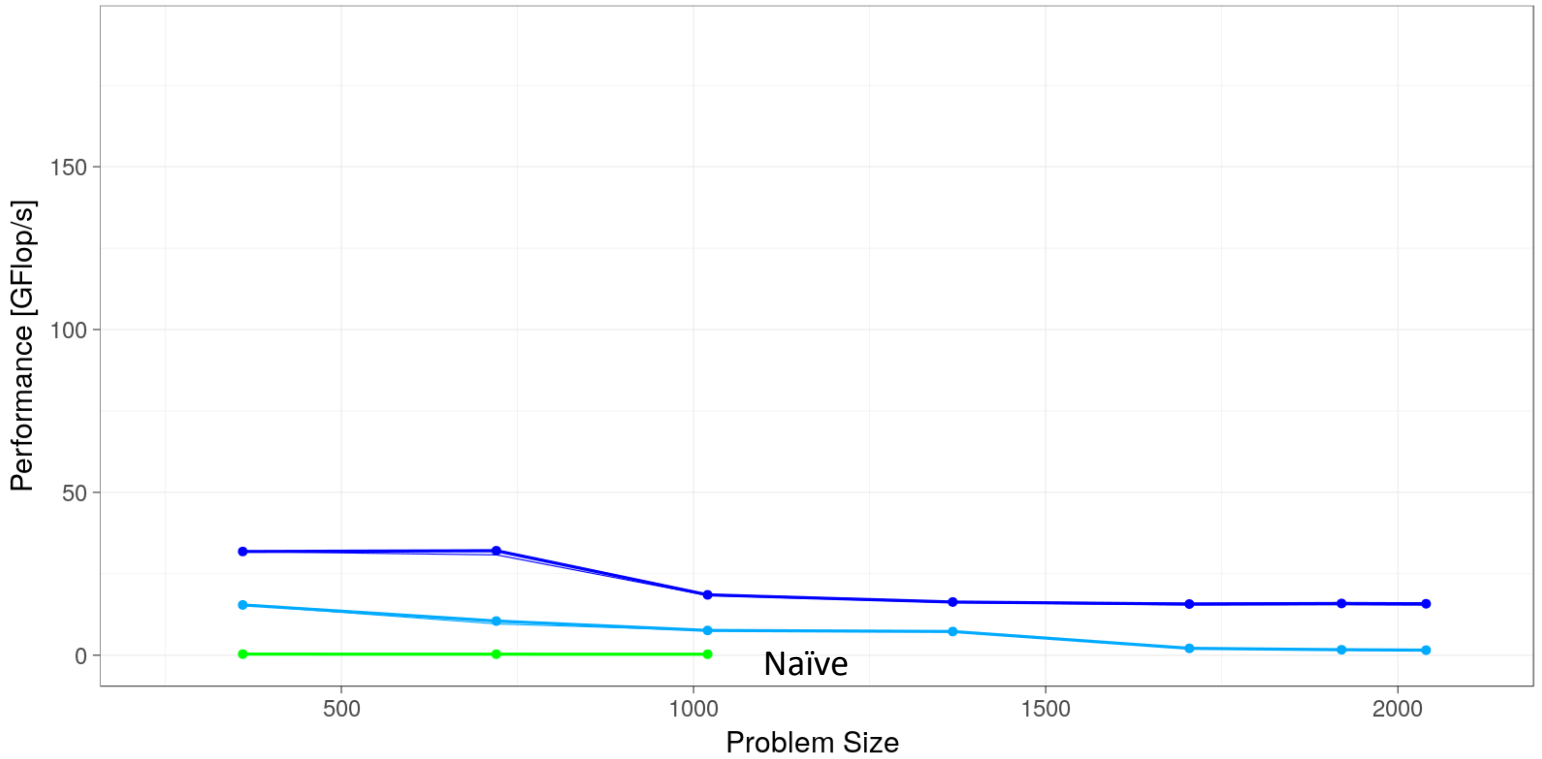


MapReduceFusion

Performance

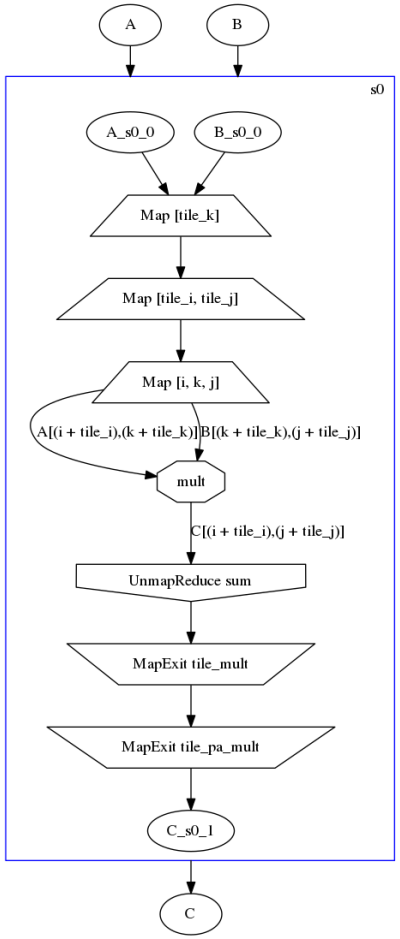


SDFG

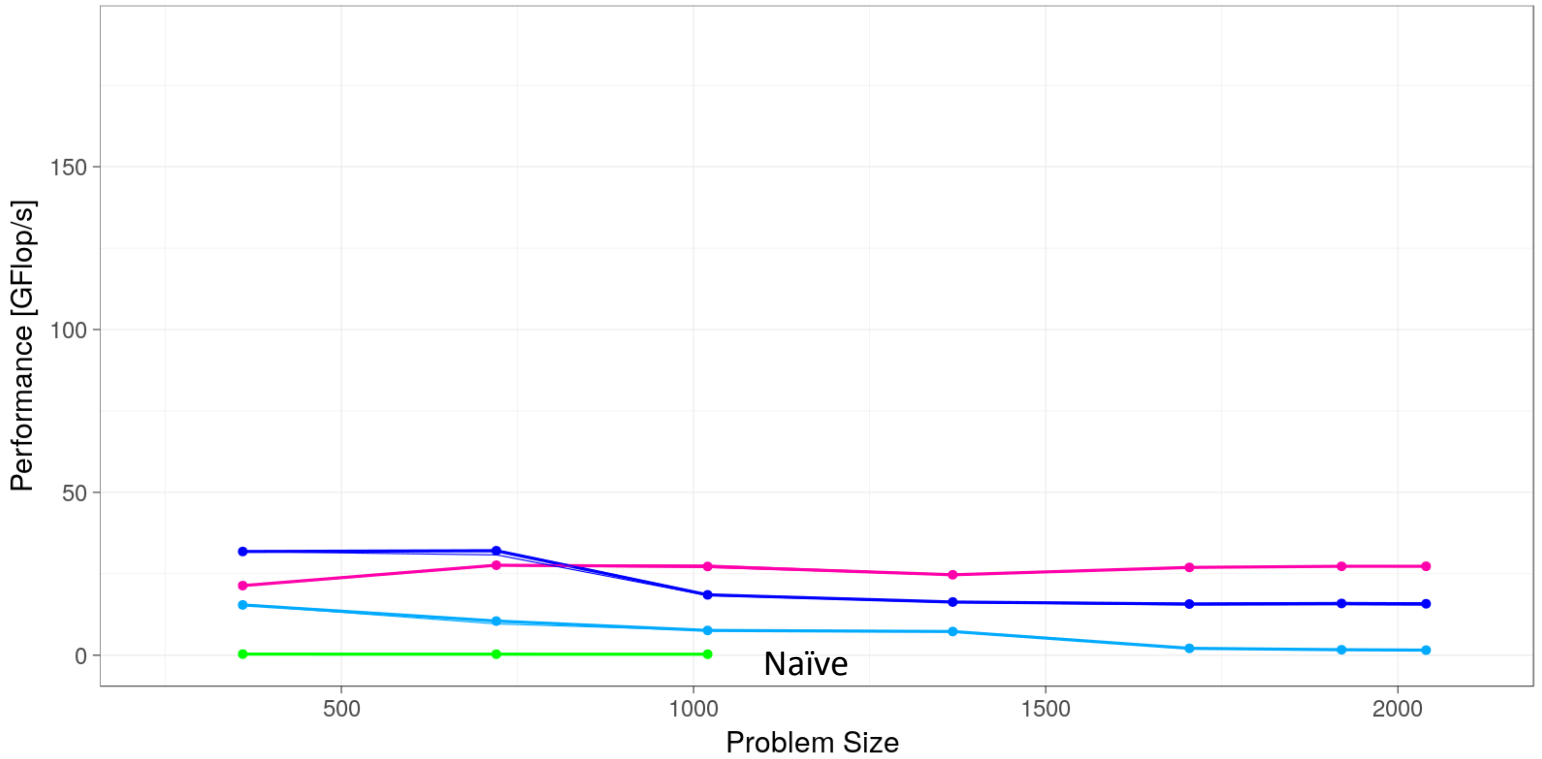


LoopReorder
MapReduceFusion

Performance

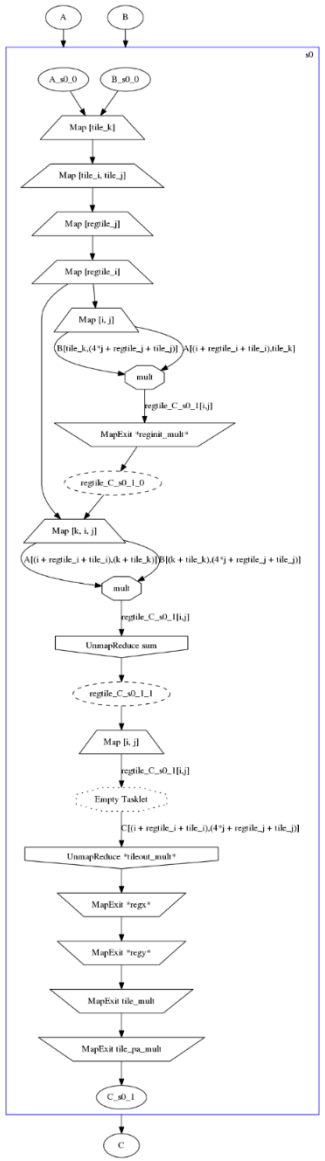


SDFG

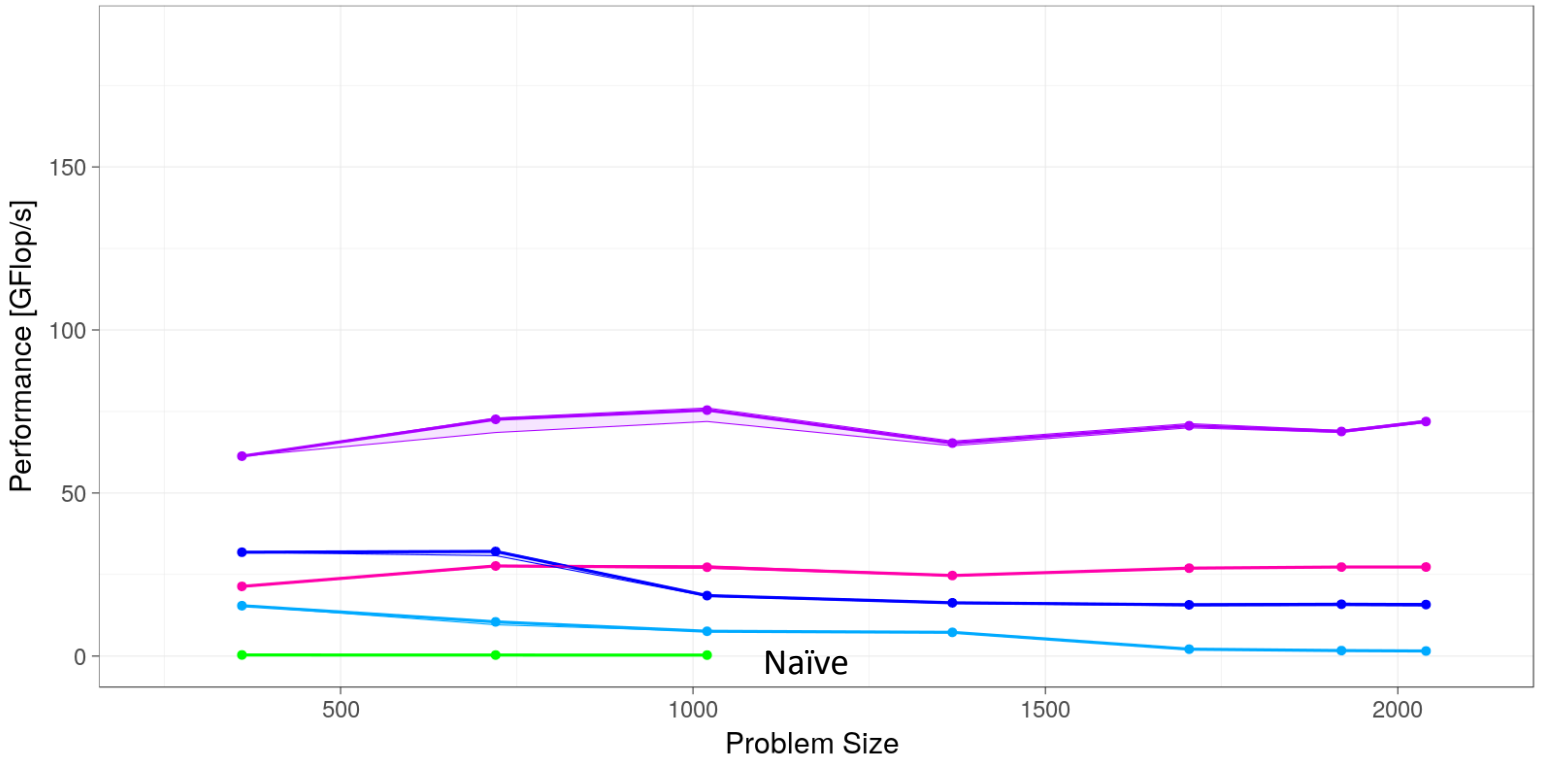


BlockTiling
LoopReorder
MapReduceFusion

Performance



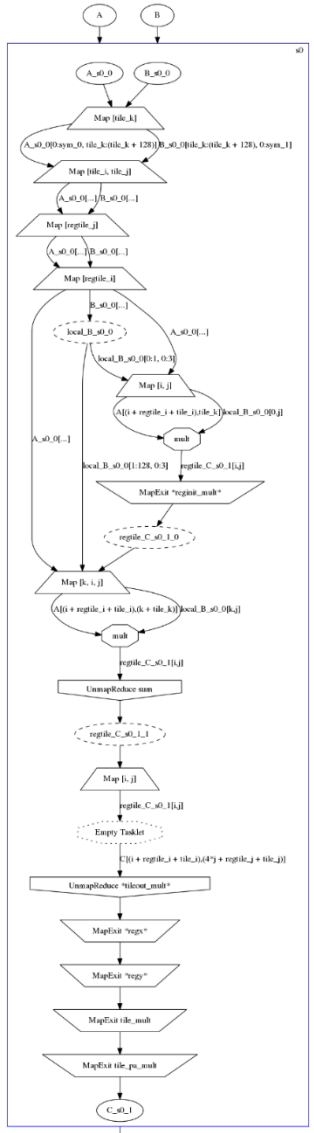
SDFG



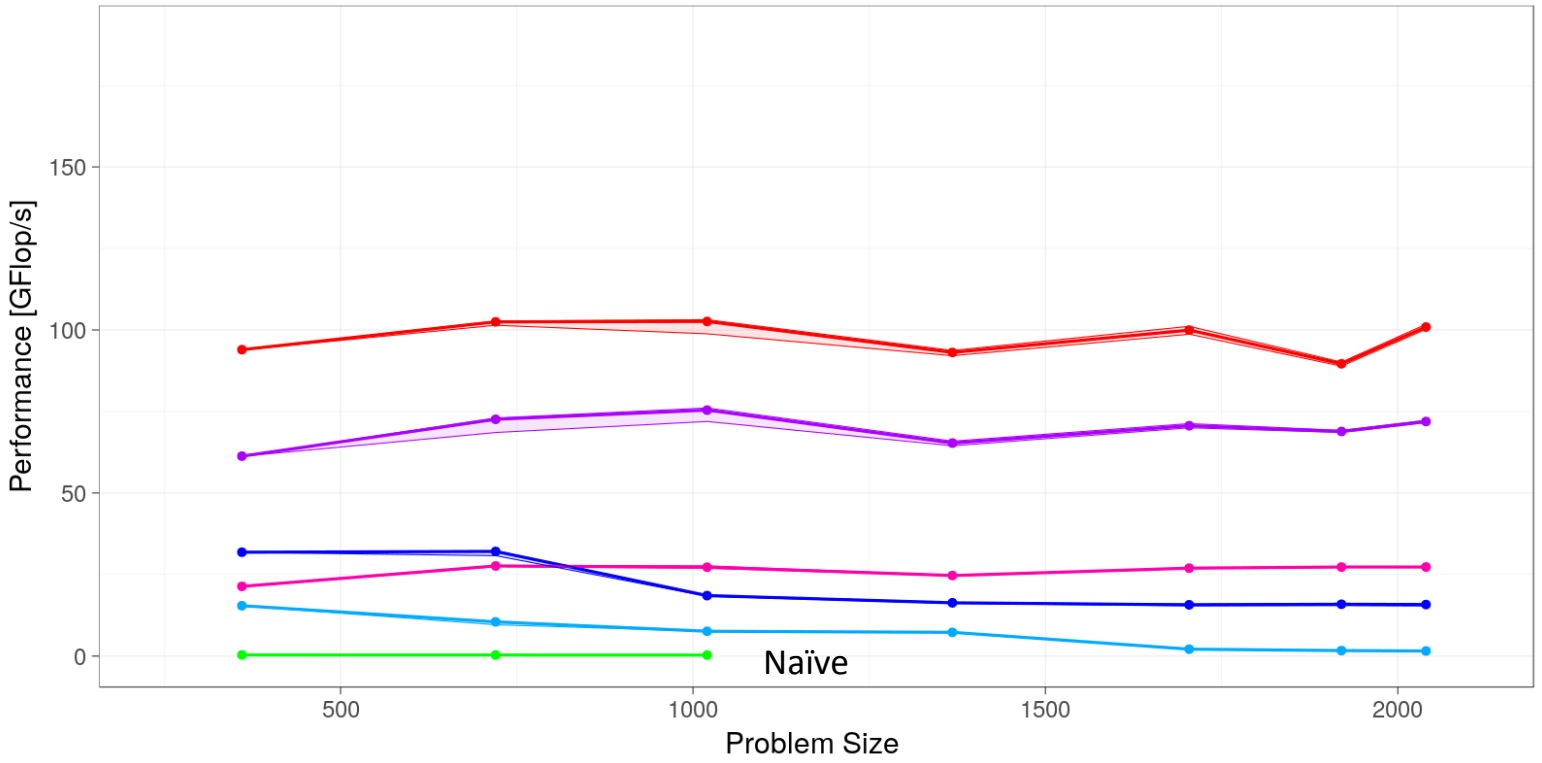
RegisterTiling

BlockTiling
LoopReorder
MapReduceFusion

Performance



SDFG

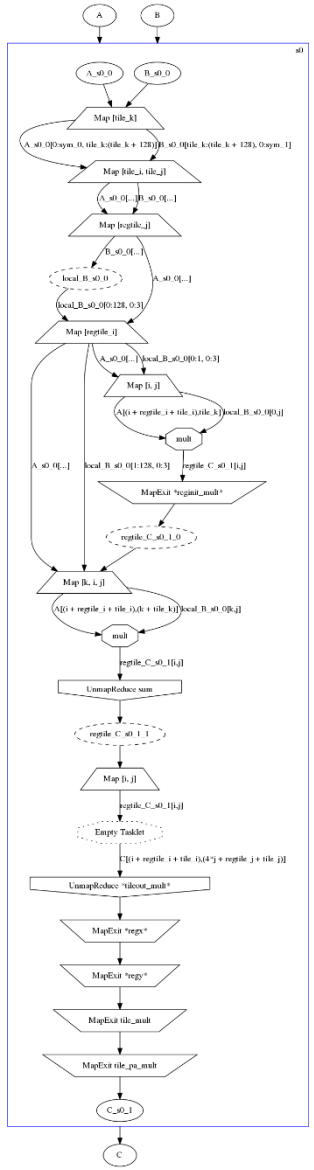


LocalStorage

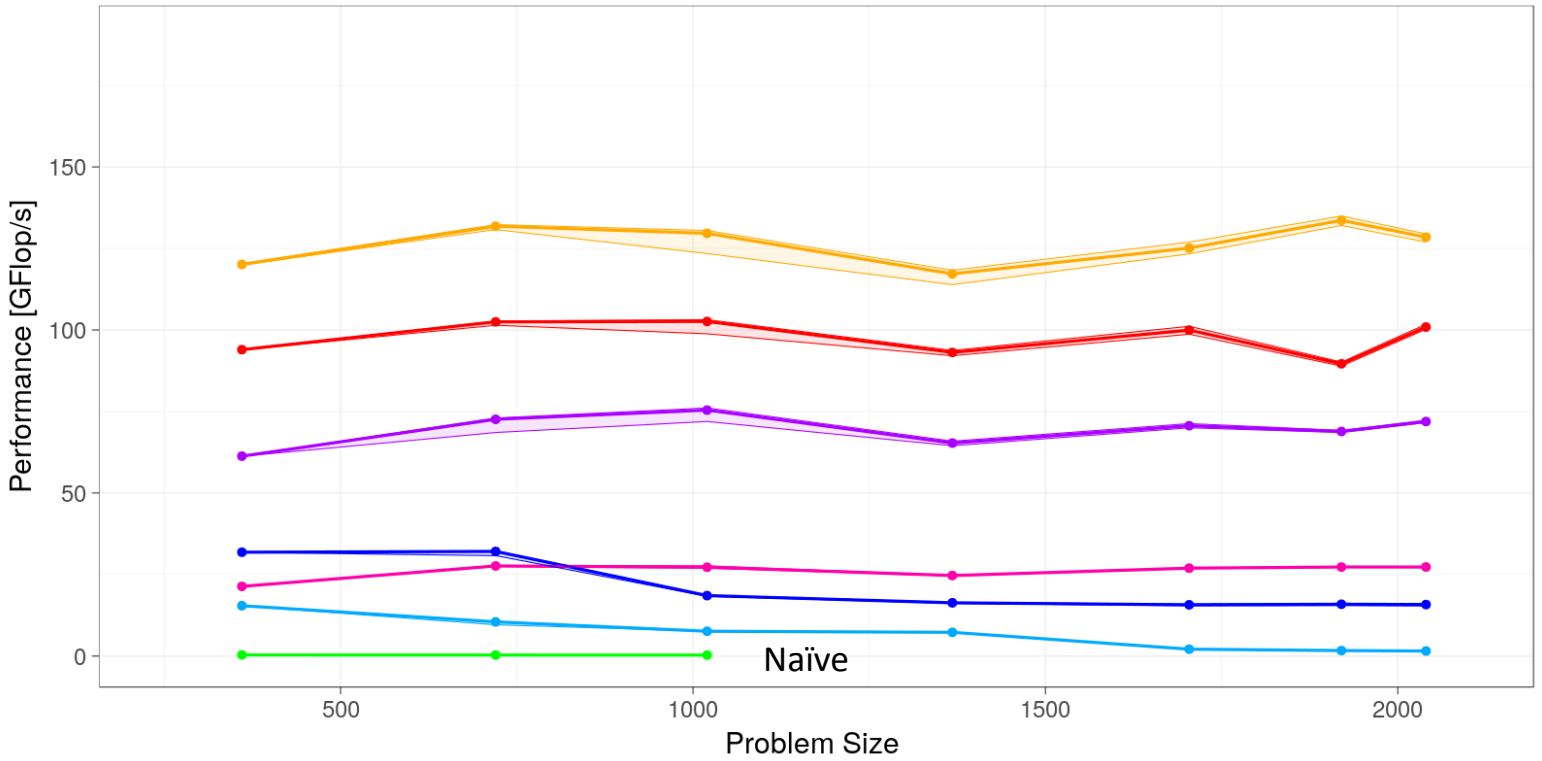
RegisterTiling

BlockTiling
LoopReorder
MapReduceFusion

Performance



SDFG



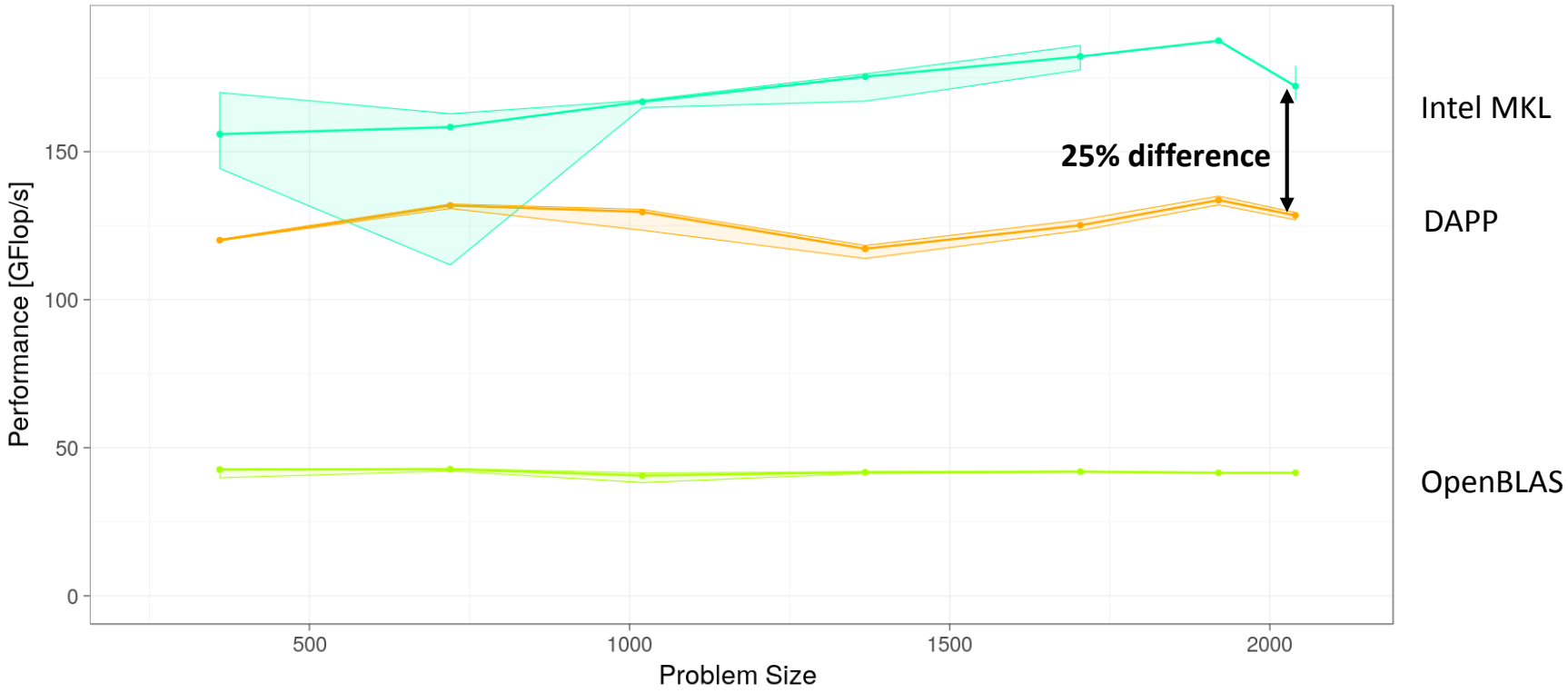
PromoteTransient

LocalStorage

RegisterTiling

BlockTiling
LoopReorder
MapReduceFusion

Performance



Generated DAPP/C++ Code (Excerpt)

```
void _program_gemm(int sym_0, int sym_1, int sym_2, double * __restrict__ A, double * __restrict__ B, double * __restrict__ C) {
    // State s0
    for (int tile_k = 0; tile_k < sym_2; tile_k += 128) {
        #pragma omp parallel for
        for (int tile_i = 0; tile_i < sym_0; tile_i += 64) {
            for (int tile_j = 0; tile_j < sym_1; tile_j += 240) {
                for (int regtile_j = 0; regtile_j < (min(240, sym_1 - tile_j)); regtile_j += 12) {

                    vec<double, 4> local_B_s0_0[128 * 3];
                    Global2Stack_2D_FixedWidth<double, 4, 3>(&B[tile_k*sym_1 + (regtile_j + tile_j)], sym_1,
                                                             local_B_s0_0, min(sym_2 - tile_k, 128));

                    for (int regtile_i = 0; regtile_i < (min(64, sym_0 - tile_i)); regtile_i += 4) {
                        vec<double, 4> regtile_C_s0_1[4 * 3];
                        for (int i = 0; i < 4; i += 1) {
                            for (int j = 0; j < 3; j += 1) {
                                double in_A = A[(i + regtile_i + tile_i)*sym_2 + tile_k];
                                vec<double, 4> in_B = local_B_s0_0[0*3 + j];
                                // Tasklet code (mult)
                                auto out = (in_A * in_B);
                                regtile_C_s0_1[i*3 + j] = out;
                            }
                        }
                        for (int k = 1; k < (min(128, sym_2 - tile_k)); k += 1) {
                        // ...
```

Scientific performance engineering for complex memory systems

Step 1: Understand core-to-core transfers – MESIF cache coherence

			Software NUMA		Software UMA		
			SNC4	SNC2	QUAD	HEM	A2A
Latency [ns] (Copy/BenchIT)	Local (L1)		3.8	3.8	3.8	3.8	3.8
	Tile (L2)		34 (M) 17 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)	34 (M) 18 (E) 14 (S,F)
	Remote		107-122 (M) 98-114 (E) 96-118 (S,F)	111-125 (M) 104-117 (E) 104-118 (S,F)	119 (M) 116 (E) 107-117 (S,F)	120 (M) 116 (E) 107-117 (S,F)	122 (M) 116 (E) 109-117 (S,F)
Bandwidth [GB/s] (Read)			2.5	2.5	2.5	2.5	2.5
Bandwidth [GB/s] (Copy)	Tile		6.7 (M) 7.6 (E)				
	Remote		7.7				
Congestion (P2P pairs)							
Contention [ns] (1:N copy) Linear, $\mathcal{T}_C(N) = \alpha + \beta \cdot N$		α β	200 34				

Write back overhead

Only 5-15% difference

Contention effects?

All values are medians within 10% of the 95% nonparametric CI, cf. TH, RB: "Scie

Step 2: Understand core-to-memory transfers – DRAM and MCDRAM

			Software NUMA		Software UMA		
			SNC4	SNC2	QUAD	HEM	A2A
Flat Mode	Latency [ns] (BenchIT)	DRAM	130-140	134-146	140	140	139
		MCDRAM	160-175	160-170	167	167	168
	Bandwidth [GB/s] (Copy NT / STREAM Copy)	DRAM	69 / 77	69 / 77	70 / 77	71 / 77	71 / 77
		MCDRAM	342 / 418	333 / 388	333 / 415	315 / 372	306 / 359
	Bandwidth [GB/s] (Read)	DRAM	71	71	77	77	77
		MCDRAM	243	288	314	314	314
	Bandwidth [GB/s] (Write)	DRAM	33	34	36	36	36
		MCDRAM	147	163	171	165	161
					71 / 82	74 / 82	73 / 82	73 / 82
					347 / 441	340 / 441	332 / 434	325 / 427
					161-171	166	168	172
					130 / 252	175 / 255	134 / 237	132 / 233
					95	124	128	118
					56	72	72	68
					246 / 294	296 / 309	273 / 274	264 / 269

MCDRAM 20% slower!

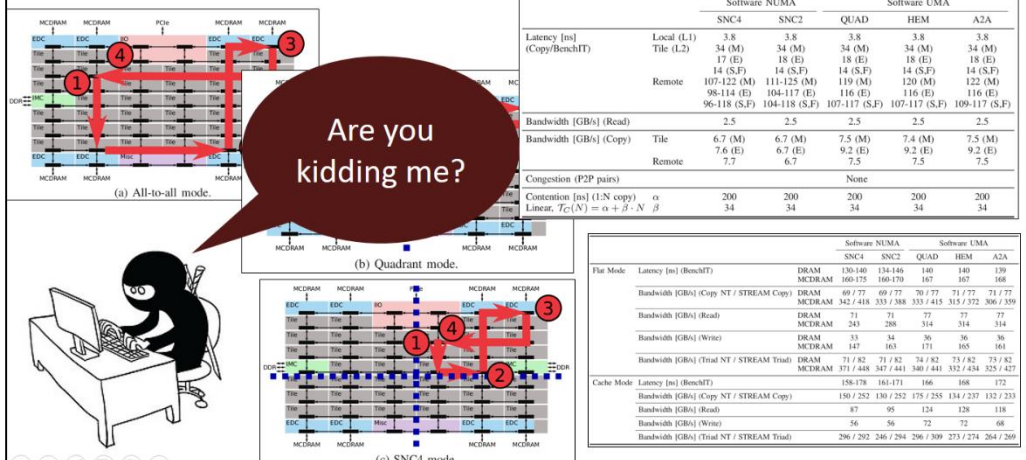
MCDRAM 4-6x faster!

Need to read and write for full bandwidth

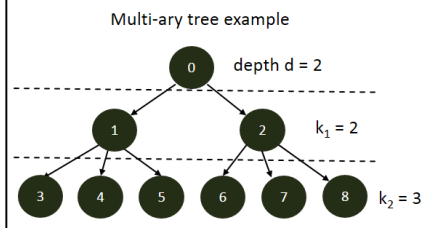
Tel Computing Systems, SC16

Questions/Discussions?

Performance engineers optimize your code!



A principled approach to designing cache-to-cache broadcast algorithms



Tree depth

Level size

$$\mathcal{T}_{tree} = \sum_{i=1}^{d} \mathcal{T}_C(k_i) = \sum_{i=1}^{d} (c \cdot k_i + b)$$

Tree cost

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

$$\mathcal{T}_{sbcast} = \min_{d,k_i} \left(\mathcal{T}_{fw} + \sum_{i=1}^{d} (c \cdot k_i + b) + \sum_{i=1}^{d} \mathcal{T}_{nb}(k_i + 1) \right)$$

Reached threads

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

S. Ramos, TH: "Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi ", ACM HPDC'13

S. Ramos, TH: "Cache line aware optimizations for ccNUMA systems (IEEE TPDS'17).