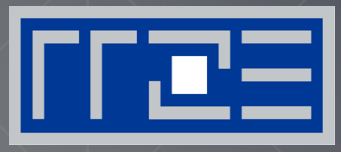
ERLANGEN REGIONAL COMPUTING CENTER

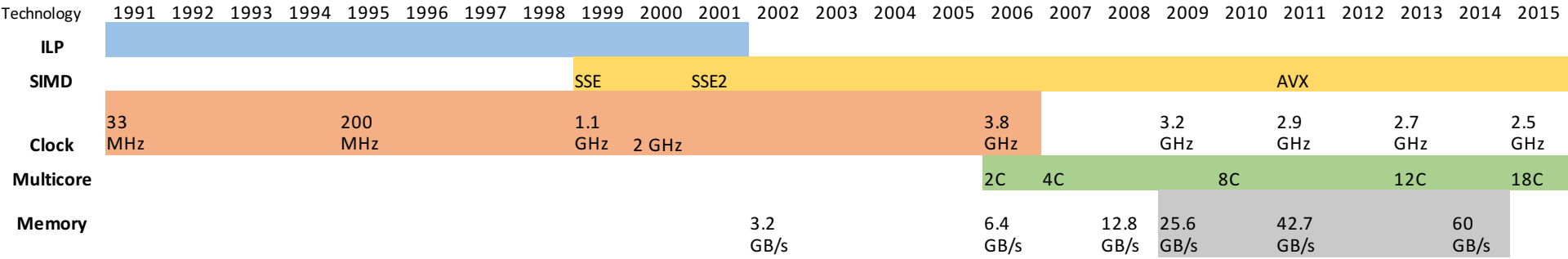


Evaluation of Intel Xeon Phi "Knights Corner": Opportunities and Shortcomings

J. Eitzinger

29.6.2016

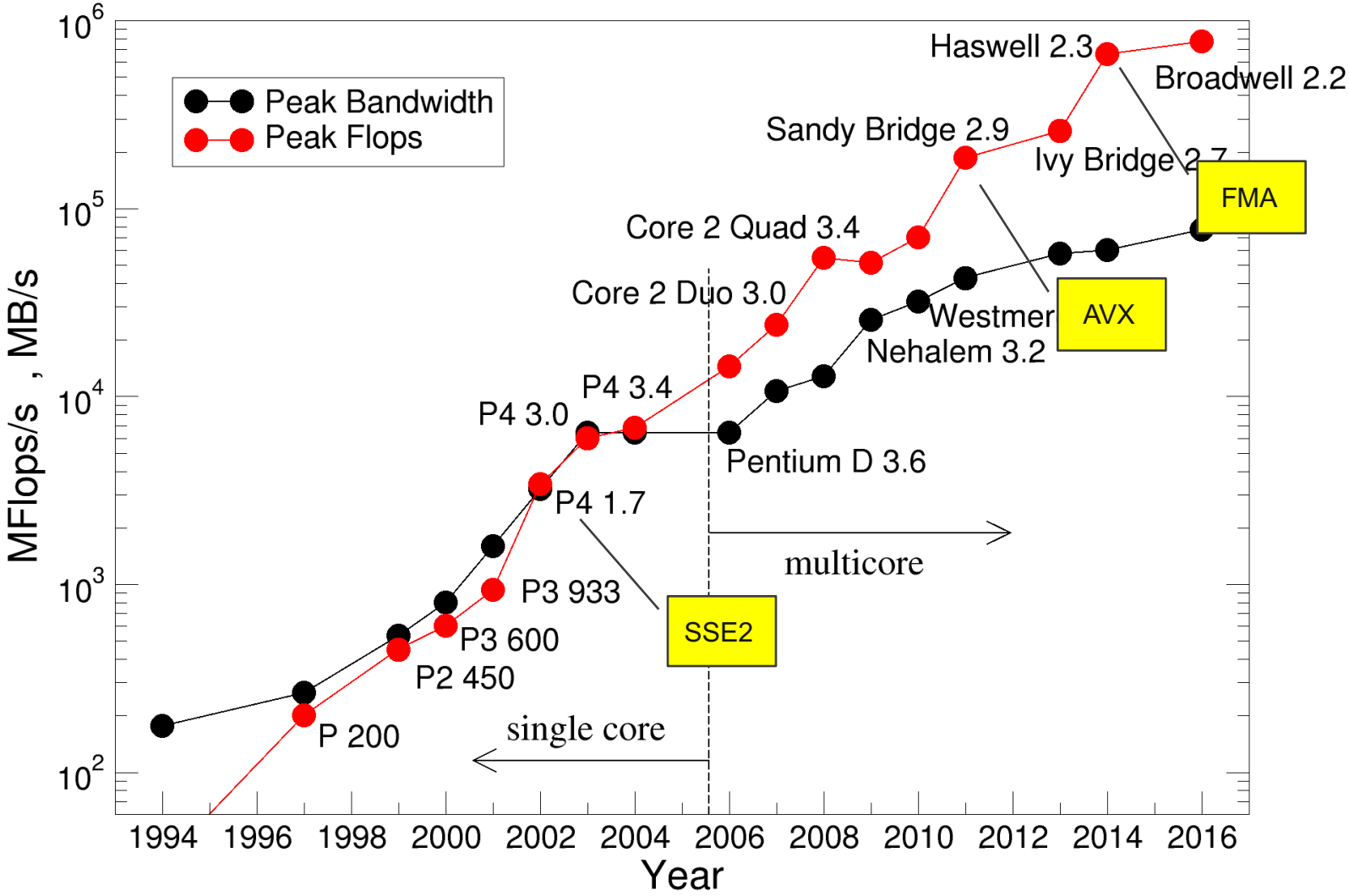
Technologies Driving Performance



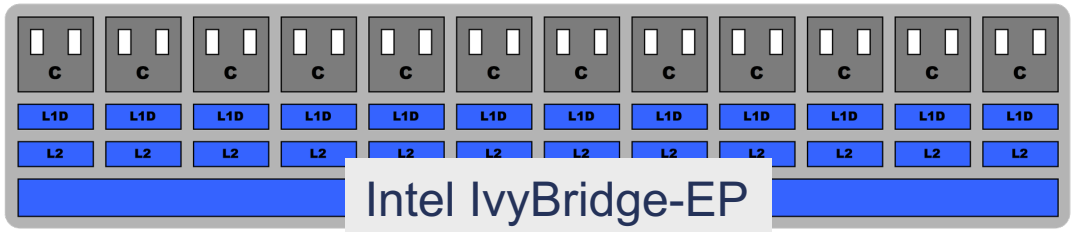
- **Before Multicore Era performance gains in the hardware transparent to software**
- **Since 2006 performance gains need to be explicitly utilized by software**

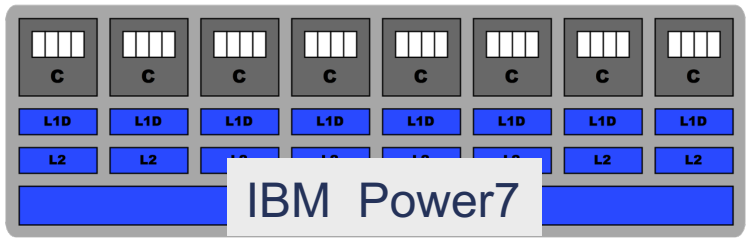
- ILP **Obstacle**: Not more parallelism available
- Clock **Obstacle**: Heat dissipation
- Multi- Manycore **Obstacle**: Getting data to/from cores
- Memory Bandwidth **Obstacle**: Wires, Cost

Performance vs. Bandwidth with new features



The driving forces behind performance





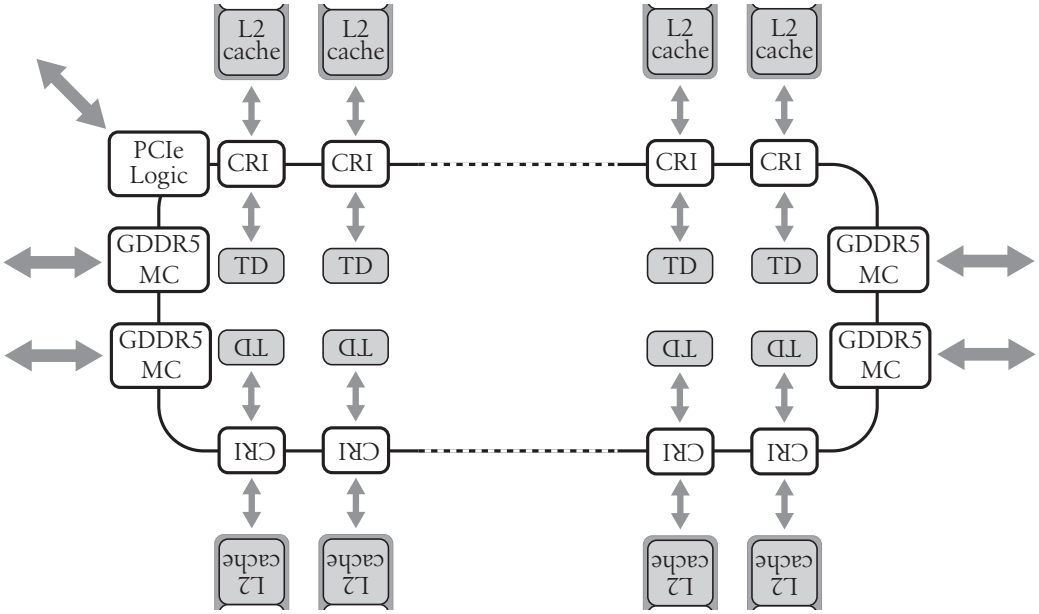
$$P = n_{core} * F * S * \nu$$

	Intel IvyBridge-EP	IBM Power7
Number of cores n_{core}	12	8
FP instructions per cycle F	2	2 (DP) / 1 (SP)
FP ops per instructions S	4 (DP) / 8 (SP)	2 (DP) / 4 (SP) - FMA
Clock speed [GHz] ν	2.7	3.7
Performance [GF/s] P	**259 (DP) / 518 (SP)**	**236 (DP/SP)**

TOP500 rank 1 (1996)

But: P=5.4 GF/s or 14.8 GF/s(dp) for serial, non-SIMD code

What is different on MIC (KNC)?



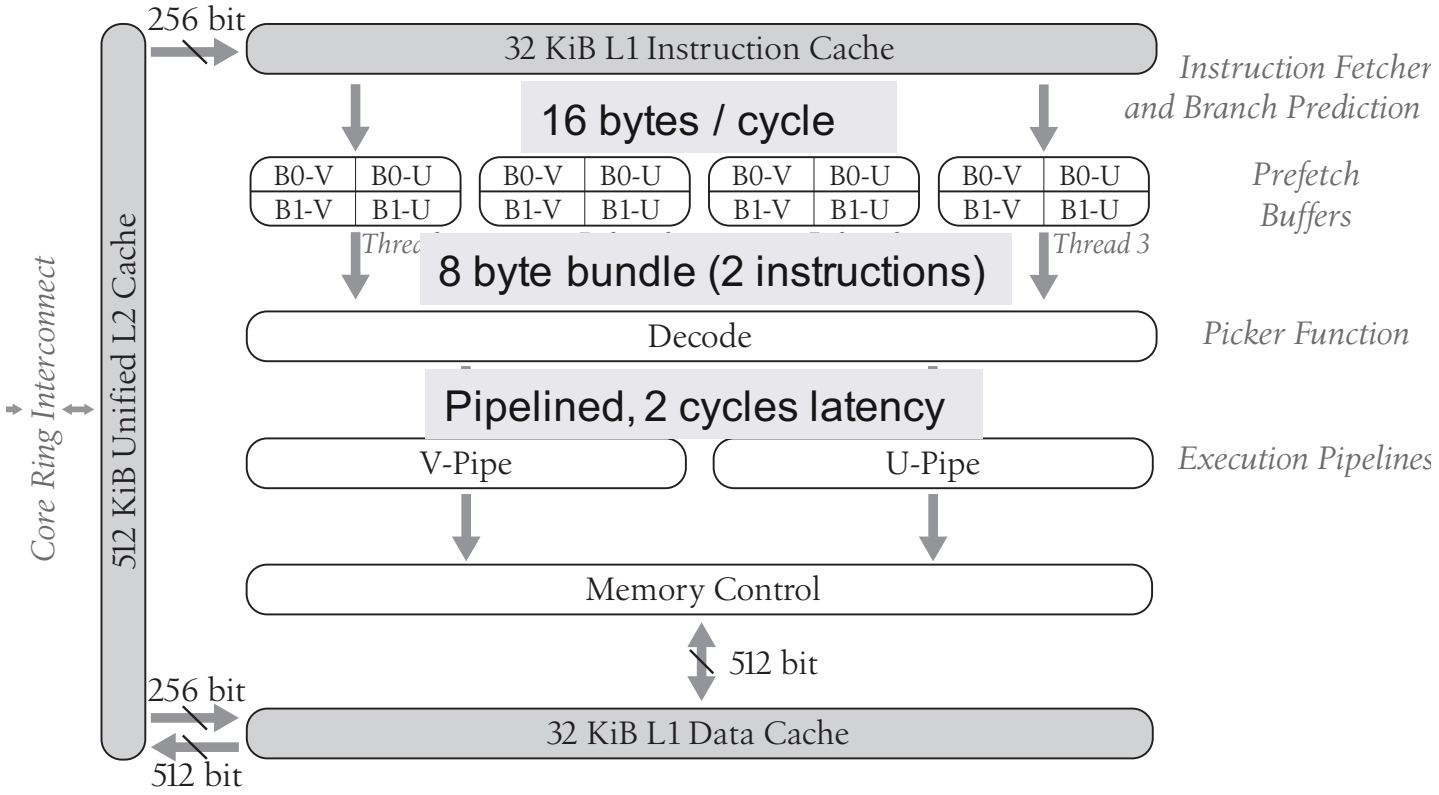
5x number of cores

2x SIMD lanes

Slower clock

	Intel Xeon Phi (KNC)	Intel Xeon Phi (KNL)
Number of cores n_{core}	60	72
FP instructions per cycle F	1	2
FP ops per instructions S	2 (FMA) x 8 (DP) / 16 (SP)	2 (FMA) x 8 (DP) / 16 (SP)
Clock speed [GHz] ν	1.05	1.5
Performance [GF/s] P	**1008 (DP) / 2016 (SP)**	**3456 (DP) / 6912 (SP)**

Xeon Phi Core architecture



- In-order execution
- 2-way superscalar
- 4-way SMT
- Scalar V-Pipe
- 512bit Vector Unit U-Pipe
- 32 64byte Vector registers
- 8 16bit Vector Mask registers

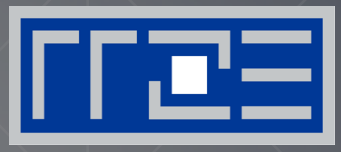
Comparison memory hierarchies

	Intel IvyBridge-EP	Intel Xeon Phi
L1 D-Cache	32 kB	32 kB
L2	256 kB	512 kB 30 MB total
L3	12 x 2.5 MB 30 MB total (shared)	-
Memory	8 channels DDR3-1866 (2S node)	6 channels GDDR5 5GHz
Peak Bandwidth	119.4 GB/s	320 GB/s
Update Bandwidth	98 GB/s (81%)	168 GB/s (53%)

Further differences:

- On Xeon Phi no hardware prefetchers between L1 and L2
- More sophisticated hardware prefetchers on IvyBridge-EP
- LLC on Xeon Phi is not shared. Cores can steal data from remote cache segments but not place data in them.

ERLANGEN REGIONAL COMPUTING CENTER



Comparing the performance of different x86 SIMD instruction sets for a medical imaging application

J. Hofmann, J. Treibig, G. Hager, G. Wellein

Motivation

- Builds upon fastrabbit paper
- Port RabbitCT benchmark to Intel Xeon Phi
- Evaluate influence of different instruction sets and their implementation in a microarchitecture on performance
- Targets:
  - SSE,AVX — IvyBridge-EP, Haswell
  - AVX2 — Haswell
  - IMCI — Knights Corner

J. Treibig, G. Hager, H. G. Hofmann, J. Hornegger, and G. Wellein: *Pushing the limits for medical image reconstruction on recent standard multicore processors*. International Journal of High Performance Computing Applications (2012), Preprint: arXiv:1104.5243

Computer Tomography reconstruction

General motivation:
Reconstruction of 3D data from 2D X-ray projection images.

Here:
X-ray projections acquired during an intervention.

Courtesy of
J. Hornegger, H. Hofmann

Reconstruction should be as fast as possible:

- Interventional
- Time resolved (4D) reconstruction

Method: 3D cone beam reconstruction of high resolution C-arm Computer Tomography dataset

RabbitCT – 3-D reconstruction: open competition

Open platform for performance-comparison of back projection implementations based on a high resolution C-arm CT dataset of a rabbit

Everyone is invited to submit results

Department of Neuroradiology and
Pattern Recognition Lab; Univ. Erlangen-Nuremberg

Partners

SIEMENS intel Leap ahead IBM BaCaTeC

Bavarian California Technology Center

References:

http://www.rabbitct.com/

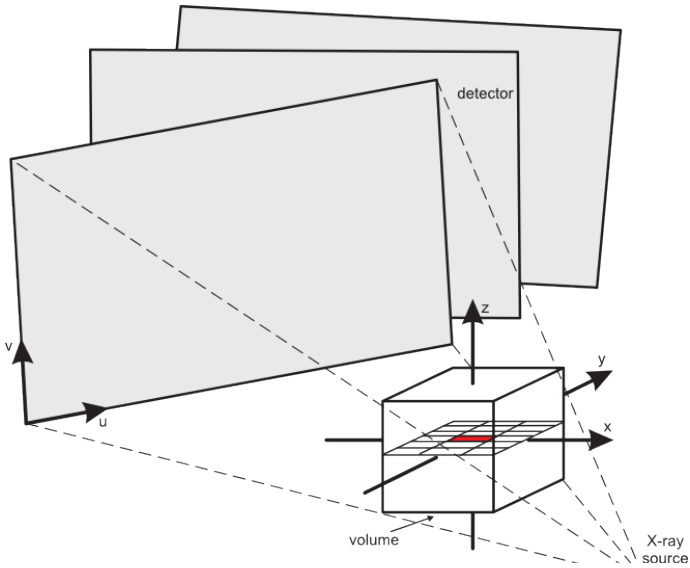
http://code.google.com/p/rabbitct

Why is RabbitCT interesting?

- Strong scaling problem
- Volume with 512^3 voxels (1024^3 also available)
- 496 projection images (1248x960px) (2.4GB)
- 13 additions, 5 subtractions, 17 multiplications, 3 divides (can be reduced to 1 reciprocal), single precision
- Data volume on volume 496GB, **streaming pattern**
- **Non-deterministic** data access **pattern** on projection images
- Non-trivial arithmetic (profits from FMA)
- On current architectures **instruction throughput limited**

- Demanding target for SIMD vectorization
- Popular optimization target for GPUs

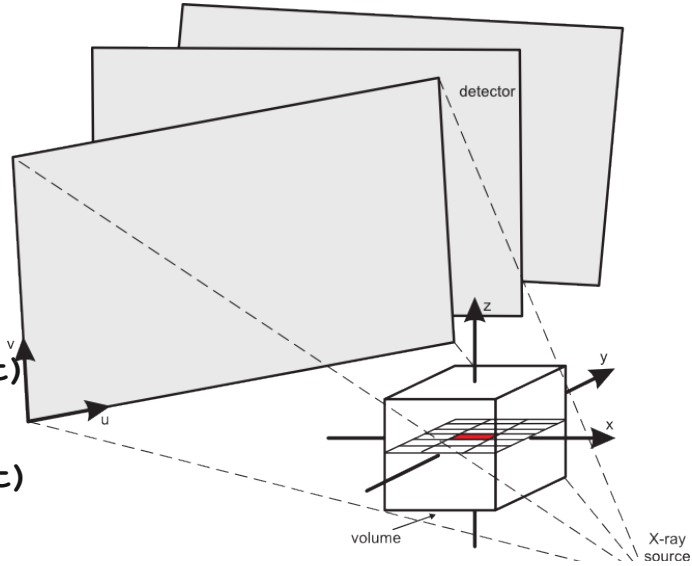
```
for(int z=0; z<L; z++) {
 for (int y=0; y<L; y++) {
   for (int x=0; x<L; x++) {
     // PART 1
     float wx = O + x * MM;
     float wy = O + y * MM;
     float wz = O + z * MM;
     // convert from WCS to ICS
     float u=wx*A[0]+wy*A[3]+wz*A[6]+A[9];
     float v=wx*A[1]+wy*A[4]+wz*A[7]+A[10];
     float w=wx*A[2]+wy*A[5]+wz*A[8]+A[11];
     // de-homogenize
     float ix = u / w;
     float iy = v / w;
     //integer indices to access projection image
     int iix = (int)ix;
     int iiy = (int)iy;
     // calculate interpolation weights
     float scalex = ix - iix;
     float scaley = iy - iiy;
```



Part 1

```
    // PART 2
    // load four values for bilinear interpolation
    float valbl = 0.0f; float valbr = 0.0f;
    float valtr = 0.0f; float valtl = 0.0f;
    if (iiy>=0 && iiy<width && iix>=0 && iix<height)
      valbl = I[iiy * width + iix];
    if (iiy>=0 && iiy<width && iix+1>=0 && iix+1<height)
      valbr = I[iiy * width + iix + 1];
    if (iiy+1>=0 && iiy+1<width && iix>=0 && iix<height)
      valtl = I[(iiy + 1) * width + iix];
    if (iiy+1>=0 && iiy+1<width && iix+1>=0 && iix+1<height)
      valtr = I[(iiy + 1)* width + iix + 1];

    // PART 3
    // perform bilinear interpolation
    float valb = (1-scalex)*valbl + scalex*valbr;
    float valt = (1-scalex)*valtl + scalex*valtr;
    float val = (1-scaley)*valb + scaley*valt;
    // add weighted results to voxel
    VOL[z*L*L+y*L+x] += val / (w * w);
  } // x-loop
 } // y-loop
} // z-loop
```



Part 2 +

Part 3

Preliminary Work

- Use reciprocal
- Eliminate if condition with **padding**
- Work reduction with **clipping mask**
- Store back indices to memory for addressing
- Pairwise load of pixel data with shuffles in correct order

- Improvements in present work:
  - Improved clipping mask
  - Better register scheduling
  - More efficient parameter handling

Results in **x1.25** better performance.

Gather instruction interface on KNC and Haswell

KNC:

```
kxnor k2, k2
..L100:
    vgatherdps zmm13{k2}, [rdi + zmm17 * 4]
    jkzd k2, ..L101
    vgatherdps zmm13{k2}, [rdi + zmm17 * 4]
    jknzd k2, ..L100
..L101:
```

Haswell:

```
vpcmpeqw   ymm7, ymm7, ymm7
vgatherdps   ymm15, [rdi + ymm11 * 4], ymm7
```

Testbed

Microarchitecture	IvyBridge-EP	Haswell	Knights Corner
Model	Xeon E5-2660 v2	Xeon E3-1240 v3	Xeon Phi 5110P
Base Clock	2.2 GHz	3.4 GHz	1.053 GHz
Sockets/Cores/SMT	2/20/40	1/4/8	1/60/240
SIMD support	SSE(128bit) AVX(256bit)	AVX2(256bit) FMA3(256bit)	IMCI(512bit)
Vector registers	16	16	32
Memory system	8 ch. DDR3-1866	2 ch. DDR3-1600	16 ch. GDDR5
Node Peak BW	119.4 GB/s	25.6 GB/s	320 GB/s
Node Update BW	98 GB/s (81%)	23 GB/s (90%)	168 GB/s (53%)

Gather microbenchmarking results

	Knight Corner				Haswell	
	L1 Cache		L2 Cache		L1 Cache	L2 Cache
	Instruction	Loop	Instruction	Loop	Instruction	Instruction
16 per CL	9.0	9.0	13.6	13.6	-	-
8 per CL	4.2	8.4	9.4	18.8	10.0	10.0
4 per CL	3.7	14.8	9.1	36.4	11.0	11.2
2 per CL	2.9	23.2	8.6	68.8	10.0	12.0
1 per CL	2.3	36.8	8.1	129.6	11.2	11.2

Serialization for loading several items per CL

No working prefetching for gather on KNC

Haswell:

- Working prefetching
- Microcode solution

Instruction code analysis

		SSE	AVX	AVX2	IMCI
Part 1	Memory	4	5	0	0
	Shuffle	6	6	4	5
	Arithmetic	21	22	15	15
	Total	31	33	19	20
Part 2	Memory	18	32	4	16
	Shuffle	1	4	7	6
	Arithmetic	6	0	0	16
	Total	25	36	11	38
Part 3	Memory	2	2	2	3
	Shuffle	5	7	0	0
	Arithmetic	6	11	9	7
	Total	13	20	11	10

FMA

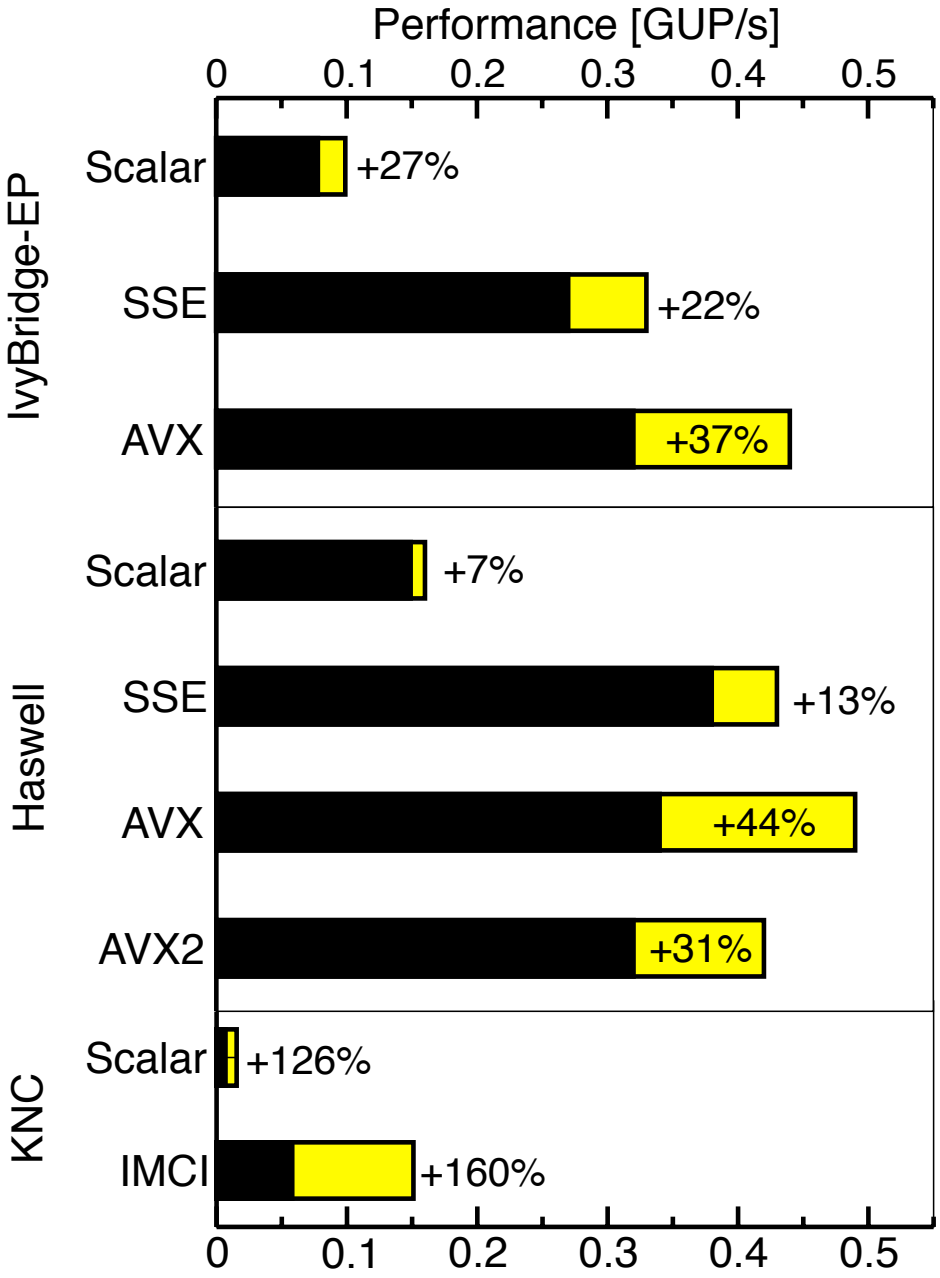
Static analysis summary

	IvyBridge-EP		Haswell	KNC
	SSE	**AVX**	**AVX2**	**IMCI**
Voxels/loop	4	8	8	16
Instr. SIMD	69	89	41	68
Instr. Scalar	57	46	41	-
Instr. Count Efficiency	83%	52%	100%	-
Runtime Speedup	3.30	4.06	2.61	-
Runtime Efficiency	83%	41%	33%	-

Direct correlation

Bad efficiency

Scalar code uses same instruction set capabilities!

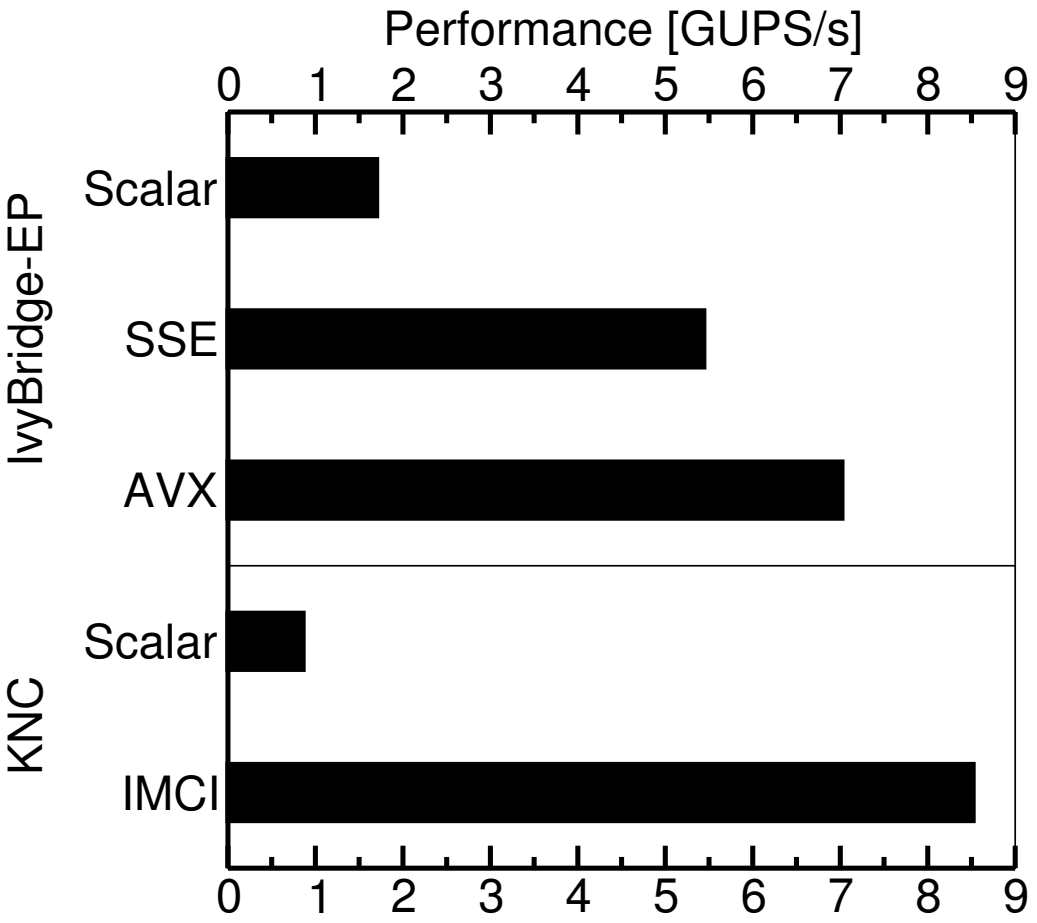


- All results with Turbo
- Great benefits from SMT
- AVX code suffers from critical path dependencies

- SSE is fastest w/o SMT
- AVX2 is the slowest

- SMT crucial
- SIMD difference huge

Full Device Results



- Code scales almost perfectly (+90% parallel efficiency on both architectures)
- KNC is faster but far away from leveraging available performance

Comparison with generated code

System	Version	Performance	
IvyBridge-EP	C Code	4.6	**AVX**
	OpenMP4	4.5	
	ISPC	5.4	**AVX**
	ASM AVX	6.2	
Haswell	C Code	1.3	**AVX2**
	OpenMP4	1.3	
	ISPC	1.7	**AVX**
	ASM SSE	1.6	
	ASM AVX	1.9	
	ASM AVX2	1.6	
KNC	C Code	6.2	
	OpenMP4	5.6	
	ISPC	6.2	
	ASM IMCI	8.5	

- ICC 13.1
- ISPC1.5.0

- OpenMP4 not faster than native code

- ISPC fastest with AVX on Haswell !!
- Handtuned vs. C (ISPC):
  - IvyBridge-EP +26% (**+13%**)
  - Haswell + 32% (**+11%**)
  - KNC +27% (+27%)

Comparison with GPU

- Thumper code (GTX680) **8 times faster** than KNC

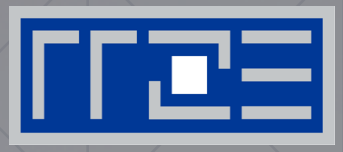
Reasons:

1. Bilinear interpolation (including the pixel data loading) is implemented completely in hardware (**texture units**). 90% of kernel time on KNC are spent on this. On GPU this is one instruction.
2. More robust **latency hiding** mechanisms. Gather instruction latencies and latencies of non contiguous data access are problematic on KNC.

Conclusion

- Employing wider SIMD units is **not efficient** for this algorithm with its non-contiguous data access
- The new **gather** instructions allow for a **simpler code** but do not improve on performance
- **Code generators** are more competitive on the gather enabled architectures
- GPUs are faster because they provide **application specific** hardware
- KNC has a lot of shortcomings

MICROBENCHMARKING FOR ARCHITECTURAL EXPLORATION

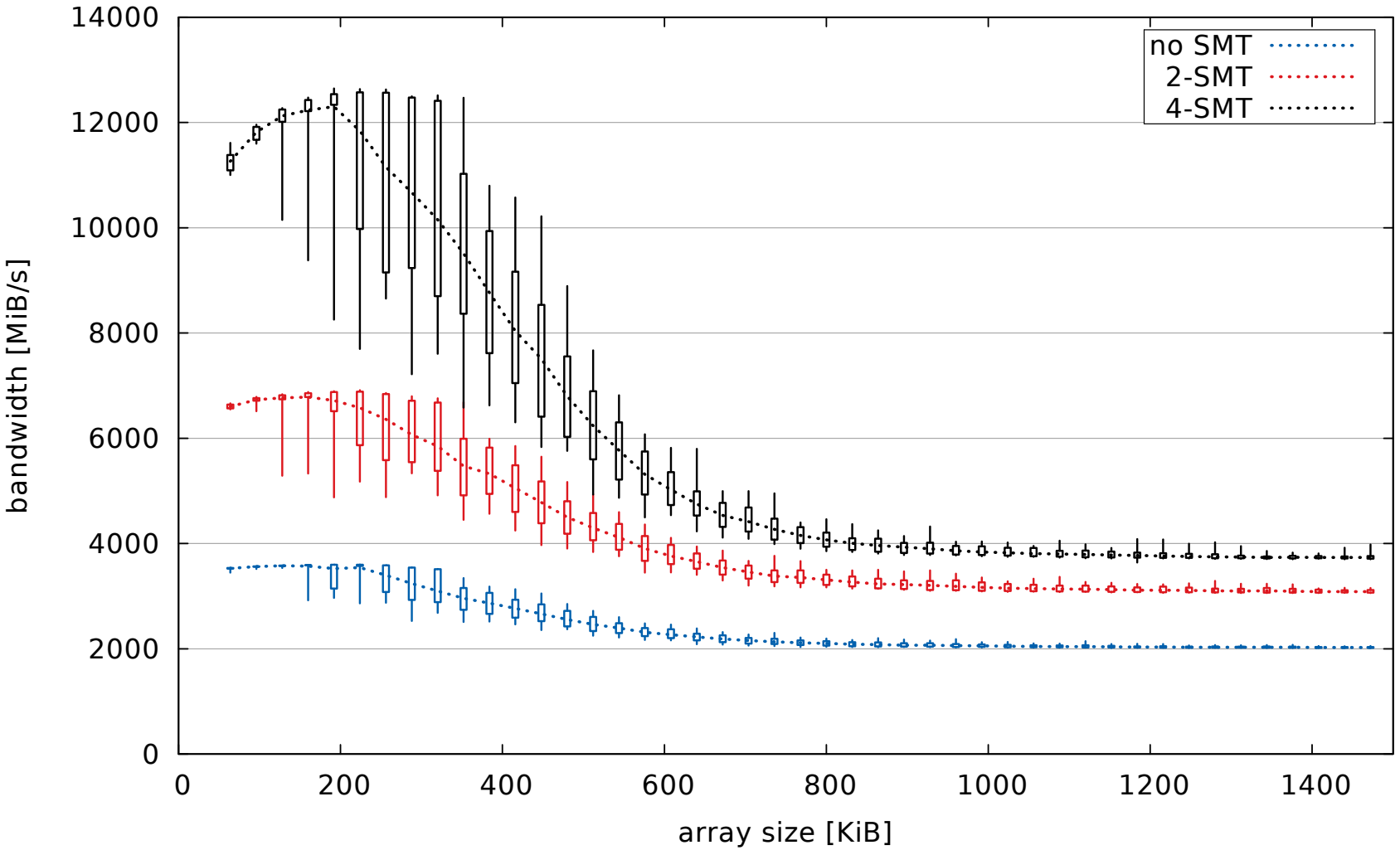


Probing of the memory hierarchy

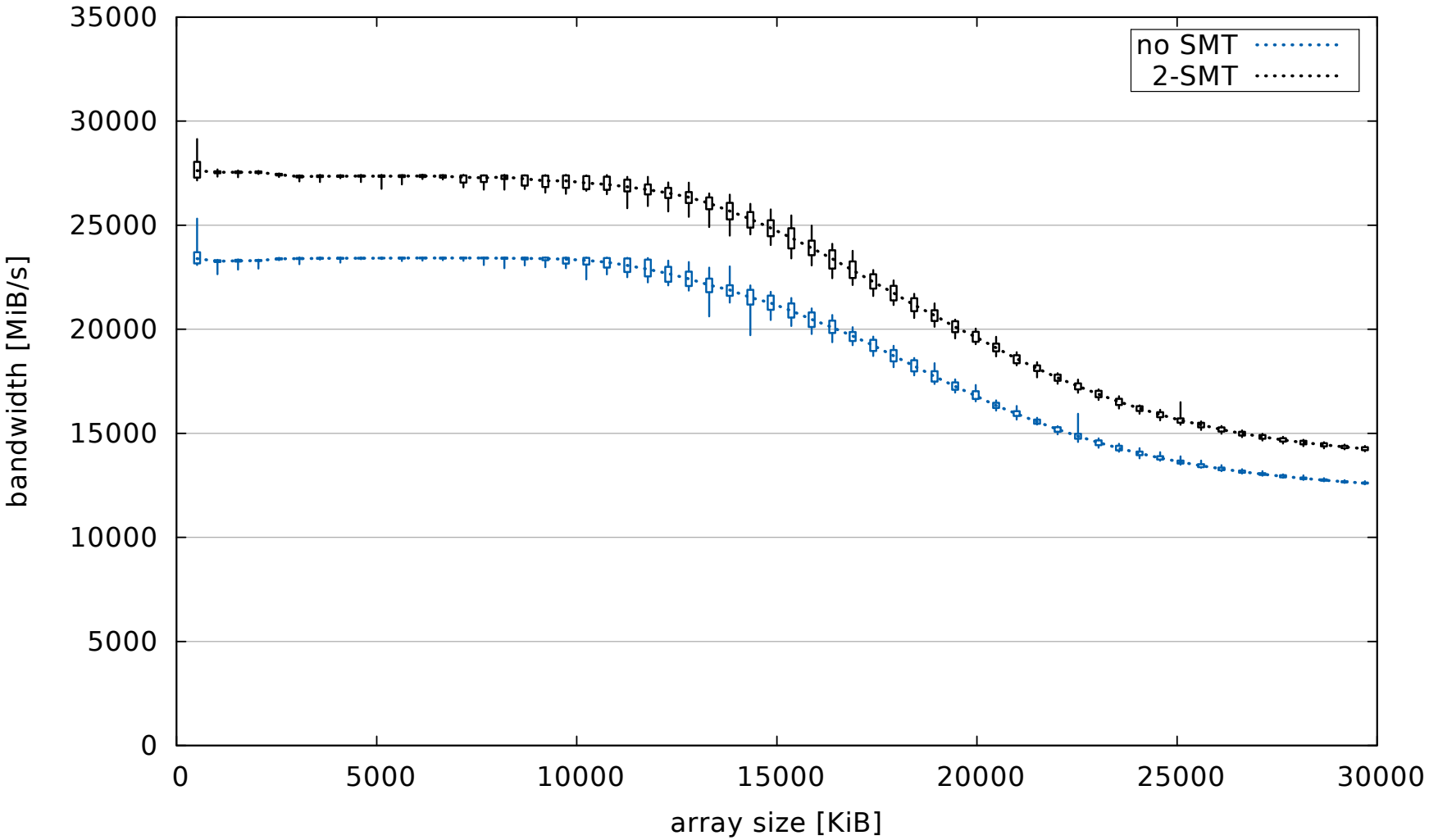
Saturation effects in cache and memory

Typical OpenMP overheads

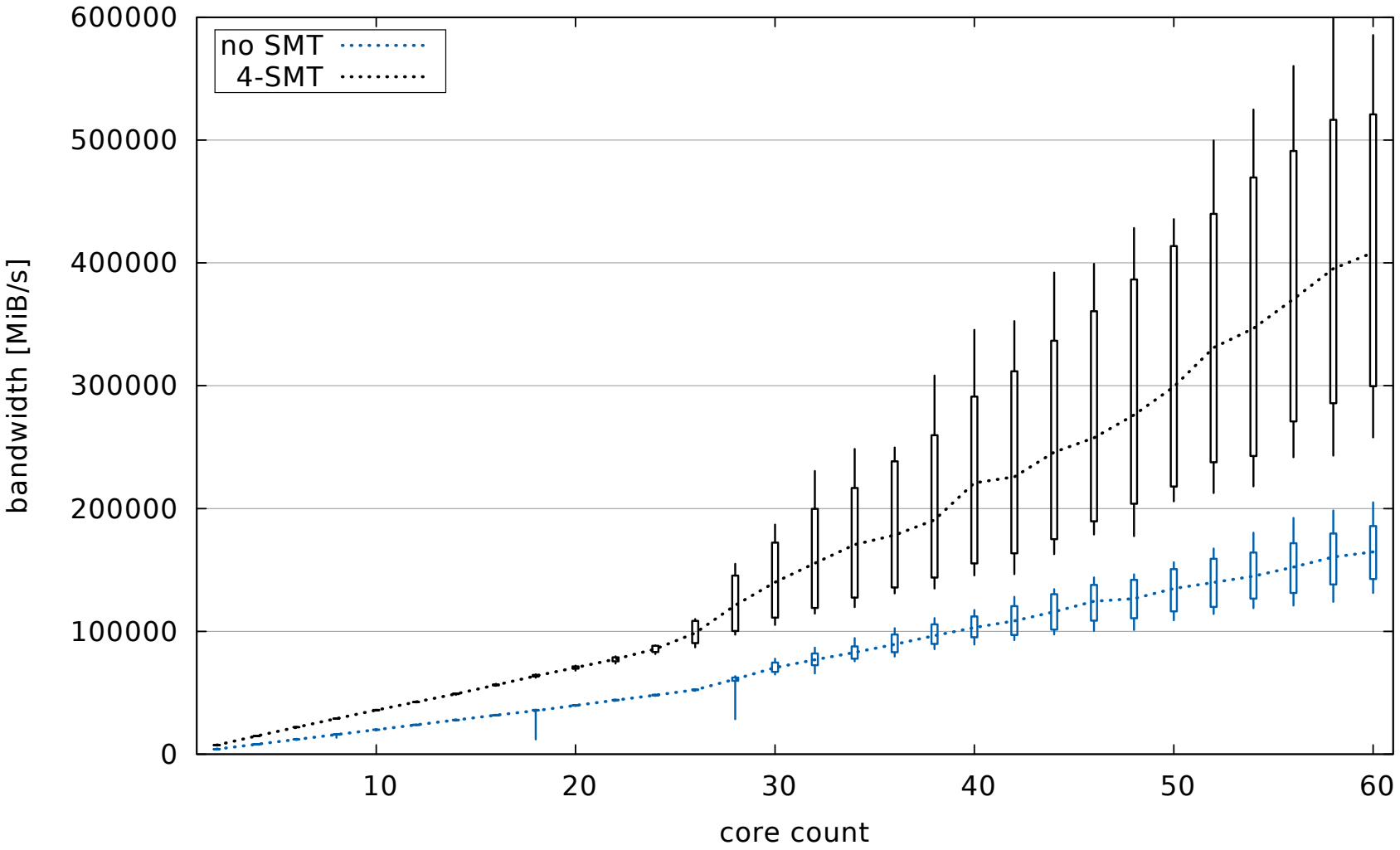
LLC performance on Xeon Phi (1 core)



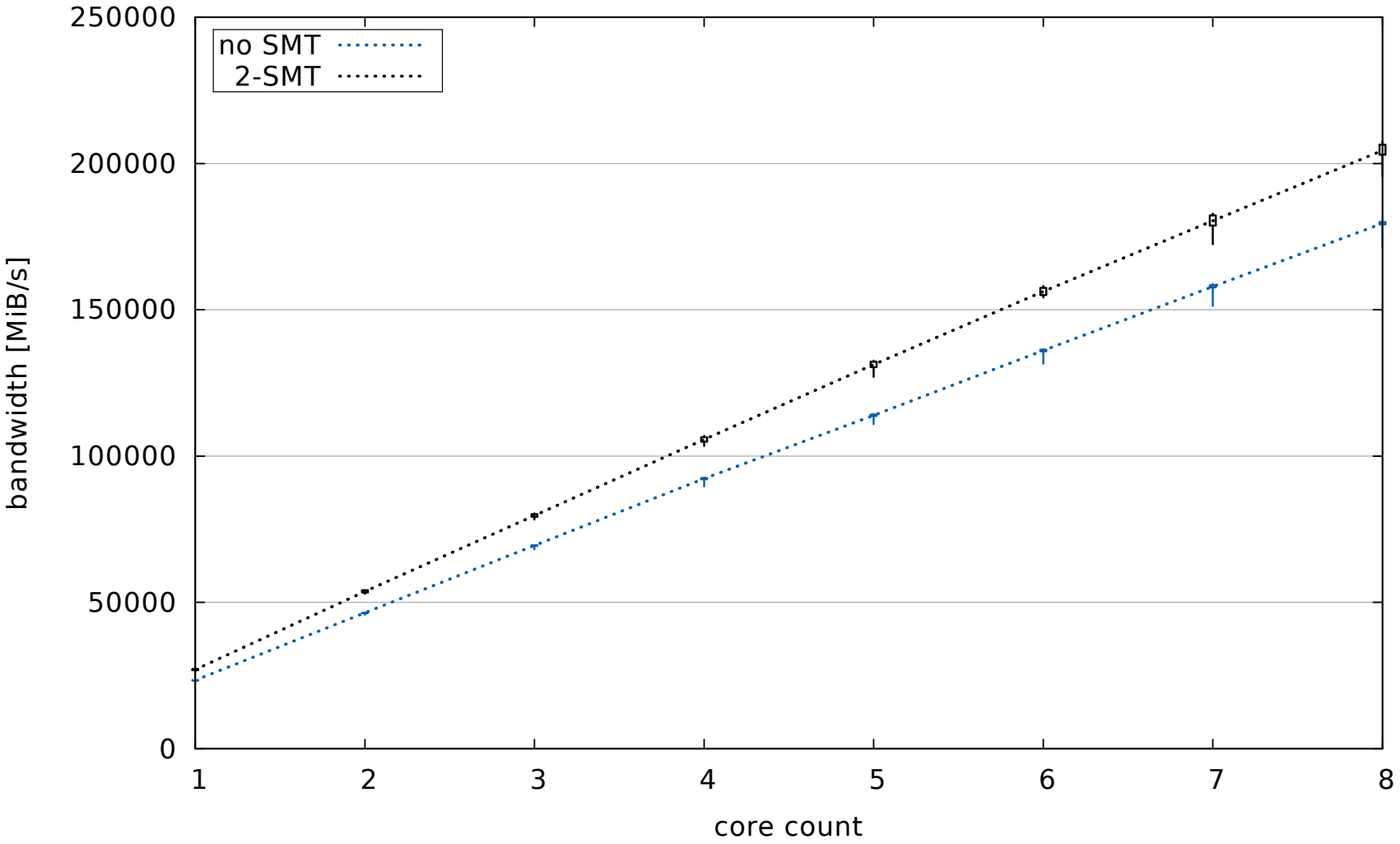
LLC performance on SandyBridge-EP (1 core)



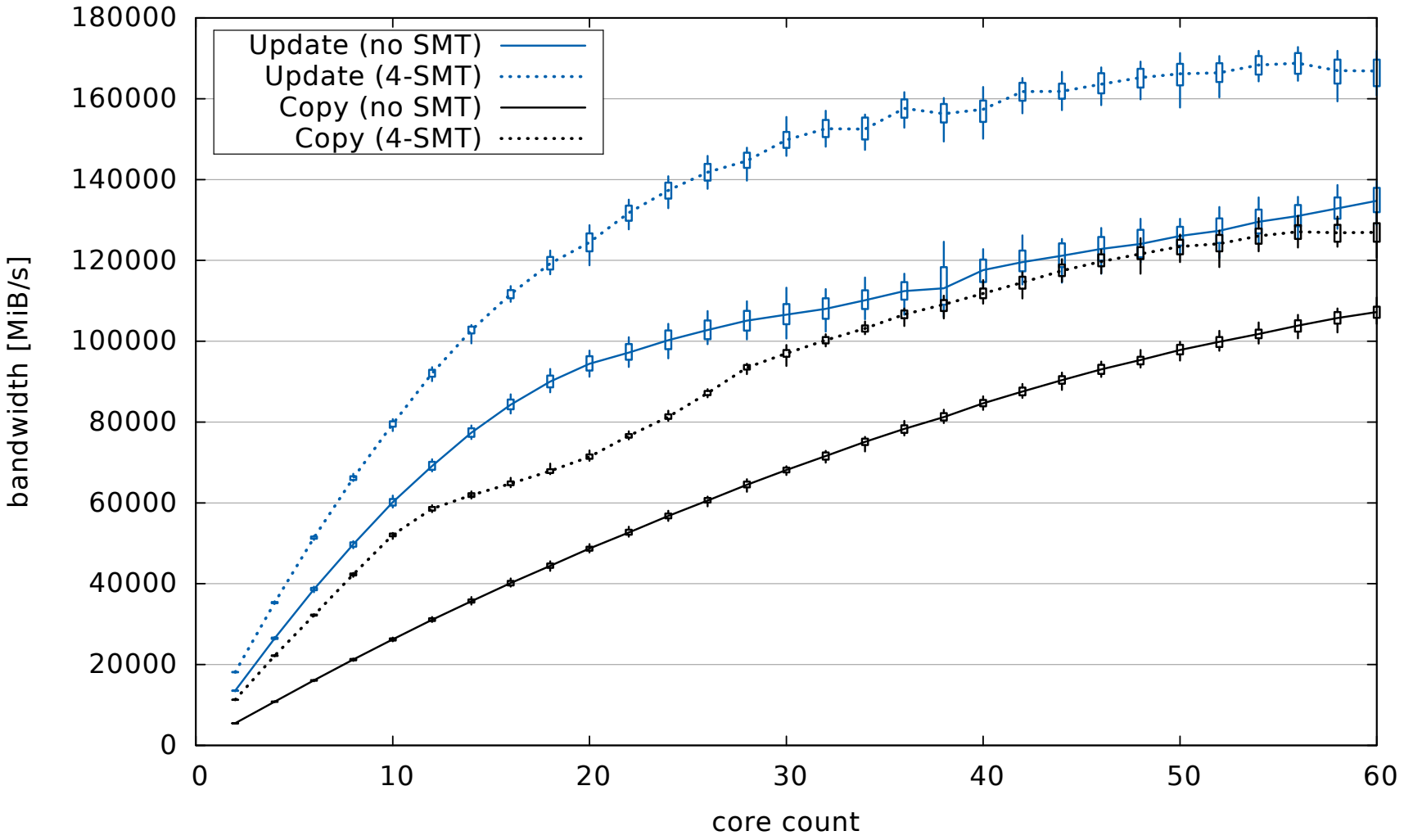
LLC bandwidth scaling Xeon Phi



LLC bandwidth scaling SandyBridge-EP



Memory bandwidth saturation on Xeon Phi



Memory bandwidth saturation on SandyBridge-EP