

(To float ... or not to float ...)
How much FP64 performance
do we really need?

ModSim'18, University of Washington, Seattle, Aug. 16

Satoshi MATSUOKA Laboratory, GSIC, Tokyo Institute of Technology

Jens Domke, Dr.



WRN: Results presented hereafter are work-in-progress!



Figure: by s3aphotography (CC BY-NC-ND 2.0)

The work was made possible by the dedicated efforts of these students

- Kazuaki Matsumura and Yohei Tsuji
- Haoyu Zhang and Hamid R. Zohouri and Ryan Barton
- Keita Yashima and Toshiki Tsuchikawa

of the Matsuoka Lab, Department of Mathematical and Computer Science, Tokyo Tech and their **supervisors Prof. Satoshi Matsuoka**, **Artur Podobas**, and **Mohamed Wahib** (and myself).

Jens Domke

Outline

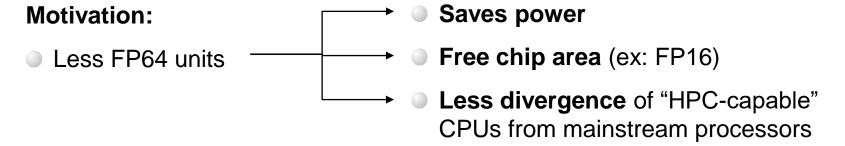


- Motivation and Initial Question
- Methodology
 - CPU Architectures
 - Benchmarks and Execution Environment
 - Information Extraction via Performance Tools
- Results
 - Breakdown FP32 vs. FP64 vs. Integer
 - Compare Time-to-Solution in Solver
 - Memory-Bound vs Compute-Bound
- Outlook into Post-K Simulations
- Summary and Conclusions
 Lessons-learned & Suggestions for Vendors and HPC Community

Motivation and Initial Question (To float ... or not to float ...?)



Thanks to the (curse of) the TOP500 list, the HPC community (and vendors) are chasing higher FP64 performance, thru frequency, SIMD, more FP units, ...



Resulting Research Questions:

- Q1: How much do HPC workloads actually depend on FP64 instructions?
- Q2: How well do our HPC workloads utilize the FP64 units?
- Q3: Are our architectures well- or ill-balanced: more FP63, or FP32, Integer, Memory?

... and ...

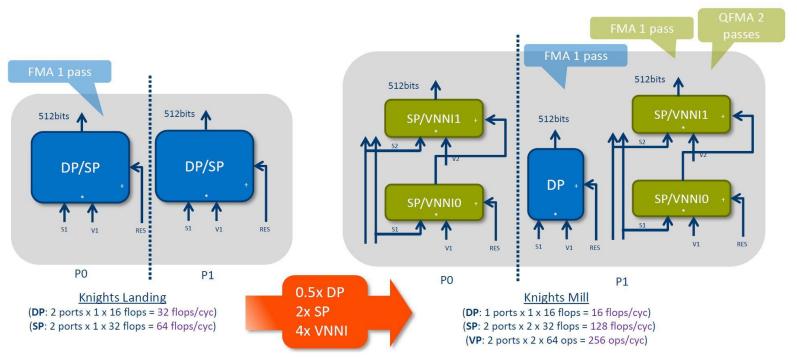
Q4: How can we actually verify our hypothesis, that we need less FP64 and should invest \$ and chip area in more/faster FP32 units and/or memory)?

Methodology – CPU Architectures



- Two very similar CPUs w/ large difference in FP64 units
- Intel dropped 1 DP unit for 2x SP and 4x VNNI (similar to Nvidia's TensorCore)
- Vector Neural Network Instruction (VNNI) supports SP floating point and mixed precision integers (16-bit input/32-bit output) ops
- → KNM: 2.6x higher SP peak performance and 35% lower DP peak perf.

KNL vs KNM: Port comparisons



Methodology – CPU Architectures



- Results may be subject to adjustments to reflect minor differences (red)
- Use dual-socket Broadwell-EP as reference system

Feature	KNL	KNM	Broadwell-EP Xeon
Model	Intel Xeon Phi CPU 7210F @ 1.30GHz	Intel Xeon Phi CPU 7295 @ 1.50GHz	Intel(R) Xeon CPU E5-2650 v4 @ 2.2GHz
# of Cores	64 (4x HT)	<mark>72</mark> (4x HT)	12 (2x HT)
CPU Base Frequency	1.3 GHz	1.5 GHz	2.2 GHz
Max Turbo Frequency	1.5 GHz (1 or 2 cores) 1.4 GHz (all cores)	1.6 GHz	2.9 GHz
CPU Mode	Quadrant mode	Quadrant mode	N/A
TDP	230 W	320 W	205 W
Memory Size	384 GB	384 GB	1,536 GB
Memory Theor. Peak BW	102 GB/s	115.2 GB/s	76.8 GB/s
MCDRAM Size	16 GB	16 GB	N/A
MCDRAM Theor. Peak BW	410 GB/s	410 GB/s	N/A
MCDRAM Mode	Cache mode (caches DDR)	Cache mode	N/A
LLC Size	32 MB	36 MB	30 MB
Instruction Set Extension	AVX-512	AVX-512	AVX2 (256 bits)
Theor. Peak Perf. (SP)	5,324 GFLOPS	13,824 GFLOPS	691.2 GFLOPS
Theor. Peak Perf. (DP)	2,662 GFLOPS	1,728 GFLOPS 345.6 GFLOPS	

Methodology – Benchmarks and Execution Environment



23 mini-apps used in procurement process of next-gen machines

ECP	Workload	Post-K	Workload
AMG	Algebraic multigrid solver for unstructured grids	CCS QCD	Linear equation solver (sparse matrix) for lattice quantum chromodynamics (QCD) problem
CANDLE	DL predict drug response based on molecular features of tumor cells	FFVC	Solves the 3D unsteady thermal flow of the incompressible fluid
CoMD	Generate atomic transition pathways between any two structures of a protein	NICAM	Benchmark of atmospheric general circulation model reproducing the unsteady baroclinic oscillation
Laghos	Solves the Euler equation of compressible gas dynamics	mVMC	Variational Monte Carlo method applicable for a wide range of Hamiltonians for interacting fermion systems
MACSio	Scalable I/O Proxy Application	NGSA	Parses data generated by a next-generation genome sequencer and identifies genetic differences
miniAMR	Proxy app for structured adaptive mesh refinement (3D stencil) kernels used by many scientific codes	MODYLAS	Molecular dynamics framework adopting the fast multipole method (FMM) for electrostatic interactions
miniFE	Proxy for unstructured implicit finite element or finite volume applications	NTChem	Kernel for molecular electronic structure calculation of standard quantum chemistry approaches
miniTRI	Proxy for dense subgraph detection, characterizing graphs, and improving community detection	FFB	Unsteady incompressible Navier-Stokes solver by finite element method for thermal flow simulations
Nekbone	High order, incompressible Navier-Stokes solver based on spectral element method	Bench	Workload
SW4lite	Kernels for 3D seismic modeling in 4th order accuracy	HPL	Solves dense system of linear equations Ax = b
SWFFT	Fast Fourier transforms (FFT) used in by Hardware Accelerated Cosmology Code (HACC)	HPCG	Conjugate gradient method on sparse matrix
XSBench	Kernel of the Monte Carlo neutronics app: OpenMC	Stream	Throughput measurements of memory subsystem

Methodology – Benchmarks and Execution Environment



- OS: clean install of centos 7
- Kernel: 3.10.0-862.9.1.el7.x86_64 (w/ enabled meltdown & spectre patches)
- Identical SSD for all 3 nodes
- Similar DDR4 (with 2400 Mhz; different vendors)
- No parallel FS (lustre/NFS/...) → low OS noise
- Boot with `intel_pstate=off` for better CPU frequency control
- Fixed CPU core/[uncore] freq. to max: 2.2/[2.7] BDW, 1.3 KNL, 1.5 KNM
- Compiler: Intel Parallel Studio XE (2018; update 3) with default flags for each benchmark plus additional: `-ipo -xHost` (exceptions: AMG w/ xCORE-AVX2 and NGSA bwa with gcc) and Intel's Tensorflow with MKL-DNN (for CANDLE)

Methodology – Info. Extraction via Performance Tools



- Step 1: Check benchmark settings for strong-scaling runs (none for MiniAMR)
- Step 2: Identify kernel/solver section of the code > wrap with additional instructions for timing, SDE, PCM, Vtune, etc
- Step 3: Find "optimal" #MPI + #OMP configuration for each benchmark (try under-/over-subscr.; each 3x runs; "best" based on time or Gflop/s)
- Step 4: Run 10x "best" configuration w/o additional tool
- Step 5: Exec proxyapp once with each performance tool

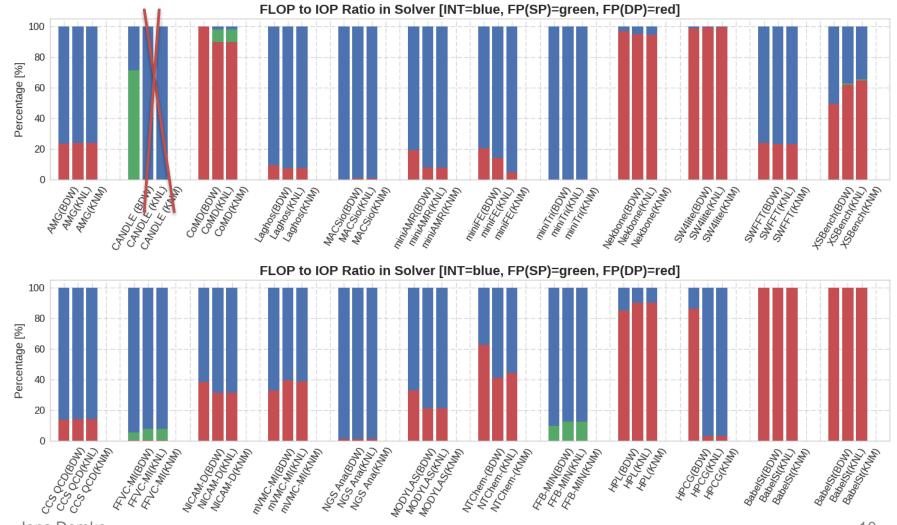
Performance analysis tools we used **(on the solver part)**:

- GNU perf (perf. counters, cache accesses, ...)
- Intel SDE (wraps Intel PIN; simulator to count each executed instruction)
- Intel PCM (measure memory [GB/s], power, cache misses, ...)
- Intel Vtune (HPC/memory mode: FPU, ALU util, memory boundedness, ...)
- Valgrind, heaptrack (memory utilization)
- (tried many more tools/approaches with less success (3)

Results – Breakdown FP32 vs. FP64 vs. Integer

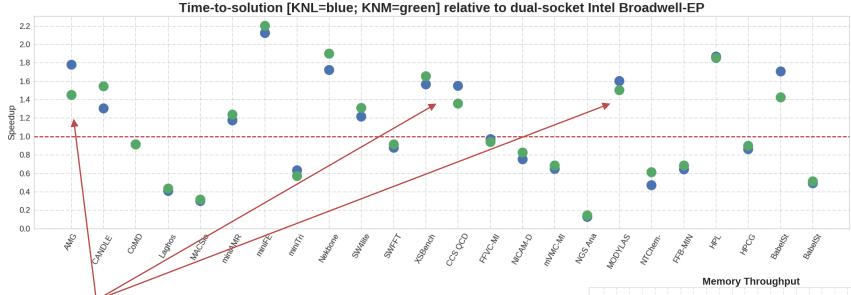


- Following: few ex. of >25 metrics (many more in raw data)
- Integer (+DP) heavy, only 4 w/ FP32, only 1 mixed precision



Results – Compare Time-to-Solution in Solver



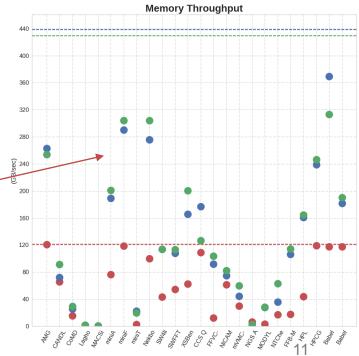


- Only 3 apps suffer from missing DP
- VNNI may help with CANDLE perf. on KNM
- 12 proxy-apps faster on 2-socket Broadwell
- Memory throughput on Phi (in cache mode) doesn't reach peak of flat mode

MiniAMR not strong-scaling → limited comparability

BabelStream ignored in above numbers

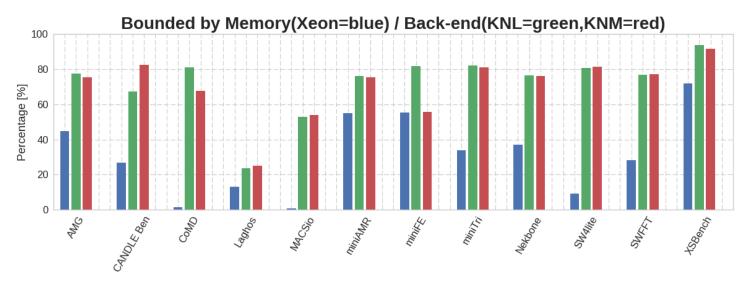
Jens Domke

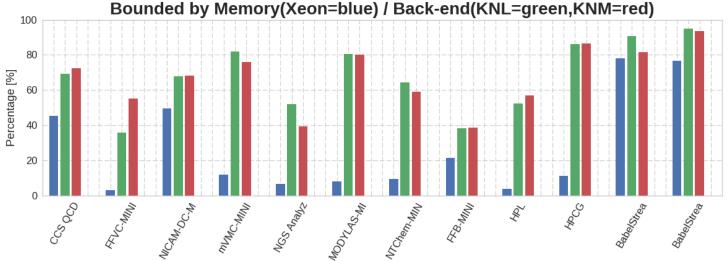


Results – Memory-/Backend-bound (VTune)



Surprisingly high (~80% for Phi) → unclear how VTune calculates these % (Memory-bound != backend-bound → no direct comparison BDW vs Phi)

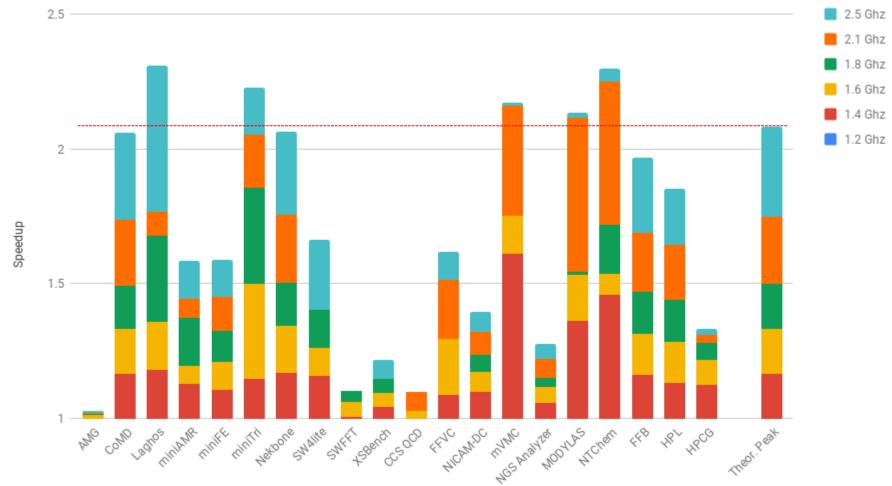




Results – Frequency Scaling for Memory-Boundedness



- Alternative idea: higher freq. > faster compute? > compute-bound?
- 15 of 20 of apps below ideal scaling → not compute-bound → memory-bound?
 Performance Improvement through Frequency Scaling



Next Steps & Outlook into Post-K Simulations



Wrap-up current work

- (If any:) Test tool/metric suggestions of ModSim attendees
 - Any feedback is welcome! → Talk to me later (ex: @poster session)
- Re-run frequency scaling experiments in clean env
- Finish in-depth analysis and write TR/paper
- Release code and data in github

Post-K simulations

- Fujitsu developed CPU simulator for Post-K chip
- Try to run all 23 benchmarks in this environment
- Extrapolate Post-K performance for these HPC workloads
- Develop recommendations for CPU/node design for Post-Post-K
- Current state: getting red tape / NDA done

Summary & Lessons-learned & Suggestions



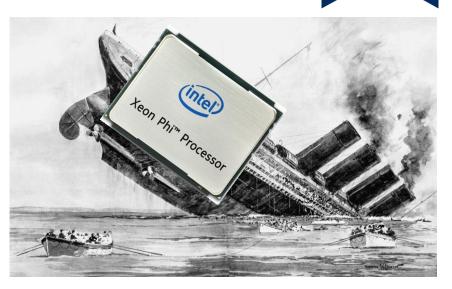
Summary (currently inconclusive):

- Procurement (proxy)apps highly FP64 dependent, but often memory-bound?
 Lessons-learned:
- IOP counting method may be misleading (→ instructions instead of ops?)
- Fixing uncore frequency is important
- Defining/measuring memory boundedness is hard \(\operatorname{\operatorname{O}} \)
- Intel MPI good on all intel chips (i.e., default settings, rank/thread mapping)
- Intel performance tools need some improvements (others: A LOT)
 - SDE: CANDLE; VTune+sample driver: nodes crash; Heaptrack: NGSA, ...

Suggestions:

- Improved proxy-apps and better documentation (and more diversity?)
 - Avoid bugs, e.g. MACSio+icc, NGSA+icc, and AMG+avx512
 - Easy choice of inputs for adapting runtime and strong-/weak-scaling
- Community effort into one repo of BMs (similar to SPEC)?





Thank You!

http://domke.gitlab.io/

domke.j.aa@m.titech.ac.jp

