



Tokyo Tech

Double-precision FPUs in High-Performance Computing: An Embarrassment of Riches?

Workshop on Large-scale Parallel Numerical Computing
Technology (LSPANC 2020 January), RIKEN R-CCS, Japan

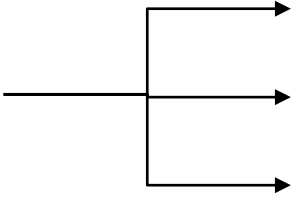
Satoshi MATSUOKA Laboratory
Dept. of Math. and Compute Sci.
Tokyo Institute of Technology

Jens Domke, Dr.

- **Motivation and Initial Question**
- **Methodology**
 - **CPU Architectures**
 - **Benchmarks and Execution Environment**
 - **Information Extraction via Performance Tools**
- **Results**
 - **Breakdown FP32 vs. FP64 vs. Integer**
 - **Gflop/s, ...**
 - **Memory-Bound vs Compute-Bound**
- **Discussion & Summary & Lessons-learned**
Suggestions for Vendors and HPC Community

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

Motivation:

- Less FP64 units
- 
- **Saves power**
 - **Free chip area** (for e.g.: FP16)
 - **Less divergence** of “HPC-capable” CPUs from mainstream processors

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 FP64, 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)?

Idea/Methodology

- Compare two similar chips; different balance in FPU's → Which?
- Use 'real' applications running on current/next-gen. machines → Which?

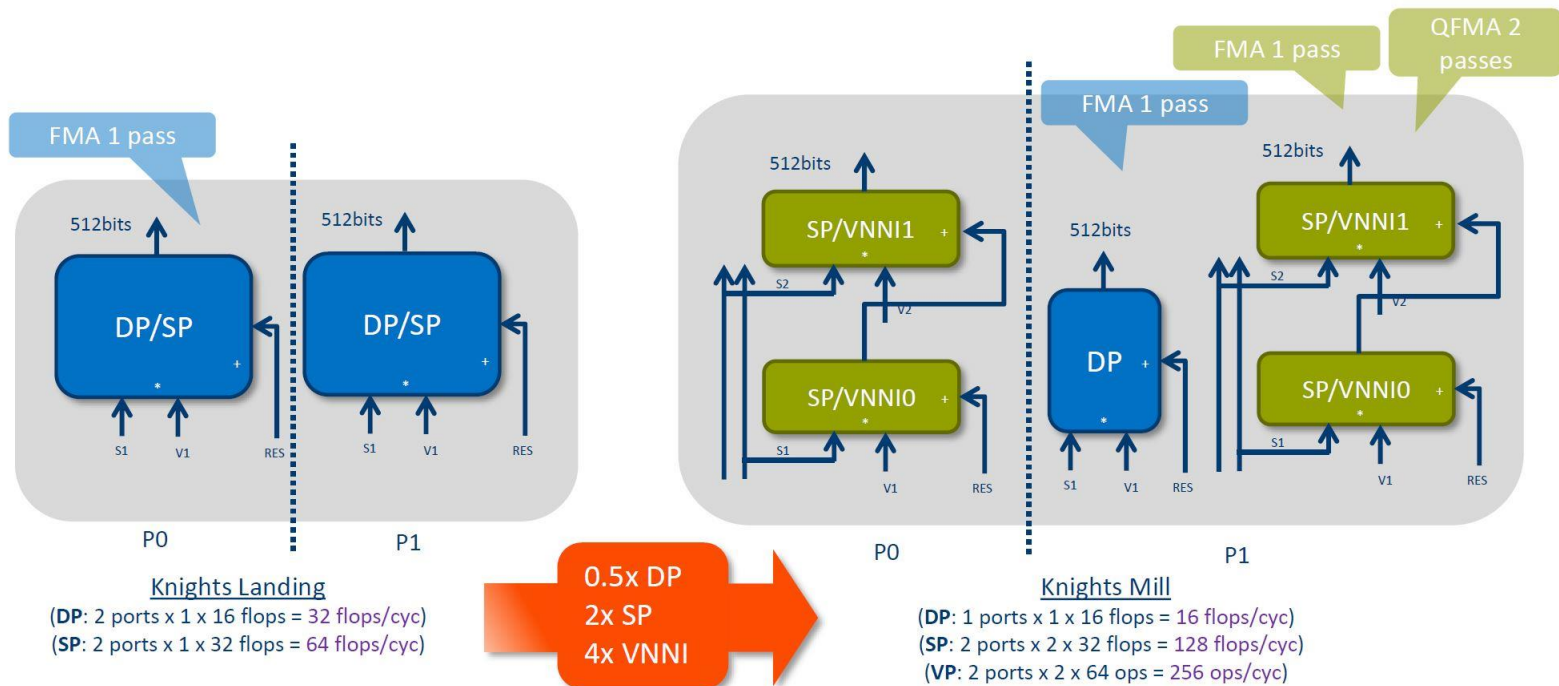
Assumptions

- Our HPC (mini-)apps are well-optimized
 - Appropriate compiler settings
 - Used in procurement of next gen. machines (e.g. Summit, Post-K, ...)
 - Mini-apps: Legit representative of the priority applications ¹
- We can find two chips which are similar
 - No major differences (besides FP64 units)
 - Aside from minor differences we know of (...more on next slide)
- The measurement tools/methods are reliable
 - Make sanity checks (e.g.: use HPL and HPCG as reference)

¹ Aaziz et al, "A Methodology for Characterizing the Correspondence Between Real and Proxy Applications", in IEEE Cluster 2018

- Two very **similar CPUs** with 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



- Results may be subject to adjustments to reflect minor differences (red)
- Use dual-socket Intel Broadwell-EP as reference system (to avoid any “*bad apples -to- bad apples*” comparison); values per node:

Feature	Knights Landing	Knights Mill	2x Broadwell-EP Xeon
Model	Intel Xeon Phi CPU 7210F	Intel Xeon Phi CPU 7295	Xeon E5-2650 v4
# of Cores	64 (4x HT)	72 (4x HT)	24 (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	210 W
Memory Size	96 GiB	96 GiB	256 GiB
➔ Triad Stream BW	71 GB/s	88 GB/s	122 GB/s
MCDRAM Size	16 GB	16 GB	N/A
➔ Triad BW (flat mode)	439 GB/s	430 GB/s	N/A
➔ MCDRAM Mode	Cache mode (caches DDR)	Cache mode	N/A
LLC Size	32 MB	36 MB	60 MB
Instruction Set Extension	AVX-512	AVX-512	AVX2 (256 bits)
Theor. Peak Perf. (SP)	5,324 Gflop/s	13,824 Gflop/s	1,382 Gfflop/s
Theor. Peak Perf. (DP)	2,662 Gflop/s	1,728 Gflop/s	691 Gflop/s

- **Exascale Computing Project (ECP)** proxy applications (12 apps)
 - Used in procuring CORAL machine
 - They mirror the priority applications for DOE/DOD (US)

- **RIKEN R-CCS' Fiber mini-apps** (8 apps)
 - Used in procuring Post-K computer
 - They mirror the priority applications for RIKEN (Japan)

- **Intel's HPL and HPCG** (and BabelStream) (3 apps)
 - Used for sanity checks

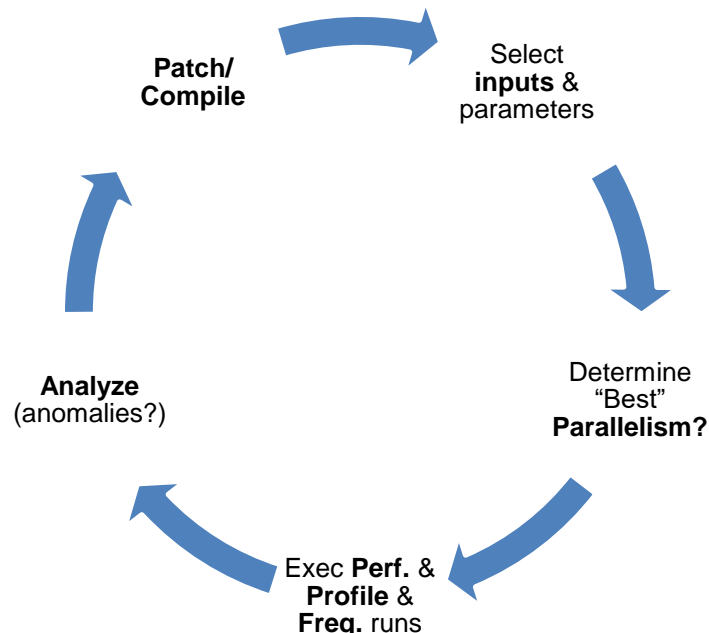
- Other mini-app suites exist:
 - PRACE (UEABS), NERSC DOE mini-apps, LLNL Co-Design ASC proxy-apps and CORAL codes, Mantevo suite, ...

- 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

- 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)

- Step 1: Check benchmark settings for **strong-scaling** runs (☹ none for MiniAMR) (→ important for fair comparison!)
- 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. proxy-app **once with each performance tool**



Early observation

- Relatively high runtime in initializing / post-processing within proxy-apps
 - E.g. HPCG only 11% – 30% in solver (dep. on system)
 - Measuring complete application yields misleading results
- ➔ Need to wrap kernel and on/off instructions for tools:

PseudoCode 1: Injecting analysis instructions

```
#define START_ASSAY {measure time; toggle on [PCM | SDE | VTune]}  
#define STOP_ASSAY {measure time; toggle off [PCM | SDE | VTune]}
```

Function *main* is

```
    STOP_ASSAY  
    Initialize benchmark  
    foreach solver loop do  
        START_ASSAY  
        Call benchmark solver/kernel  
        STOP_ASSAY  
        Post-processing  
    Verify benchmark result  
    START_ASSAY
```


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 ☹)

Raw Metric	Method/Tool
Runtime [s]	MPI_Wtime()
{FP / integer operations}	Software Development Emulator
{Branches operations}	SDE
Memory throughput [B/s]	PCM (pcm-memory.x)
{L2/LLC cache hits/misses}	PCM (pcm.x)
Consumed Power [Watt]	PCM (pcm-power.x)
SIMD instructions per cycle	perf + VTune ('hpc-performance')
Memory/Back-end boundedness	perf + VTune ('memory-access')

Many times we were stuck (a few examples below)

- VTune crashing machines (w/ Intel's sampling driver ☹️ → use *perf*)
 - Worked on older kernels (pre Spectre and Meltdown patch)
- Changing core frequency leads to change in uncore frequency
 - Use LikWid to fix uncore frequency
 - LikWid itself requires changing a kernel parameter (`intel_pstate=off`)
- Many applications crashed for different reason
 - E.g.: AMG's iteration count is inconsistent with AVX512 optimization; NGSa only compiled w/ GNU gcc; we fixed MACSio's segfaults for Intel compiler
- Several apps have different input datasets
 - “Right” choice tricky (but req. for strong-scaling sweep of threads/processes)
 - Some enforce the `#thread/#proc` based on domain decomposition scheme
- Measuring performance metric for solver phase in apps
 - For some (like CANDLE written in Python) not straightforward

What are we looking for?

- Breakdown of applications requirements/characteristics
- **Performance metrics**
- **Memory-bound vs. compute-bound**
- Power profile

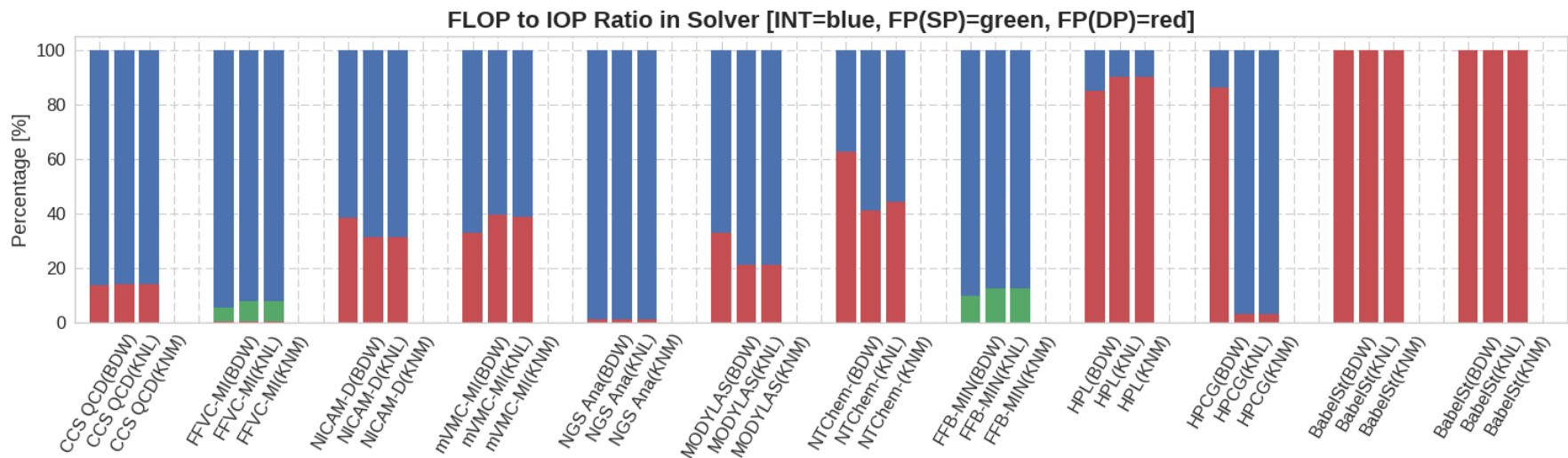
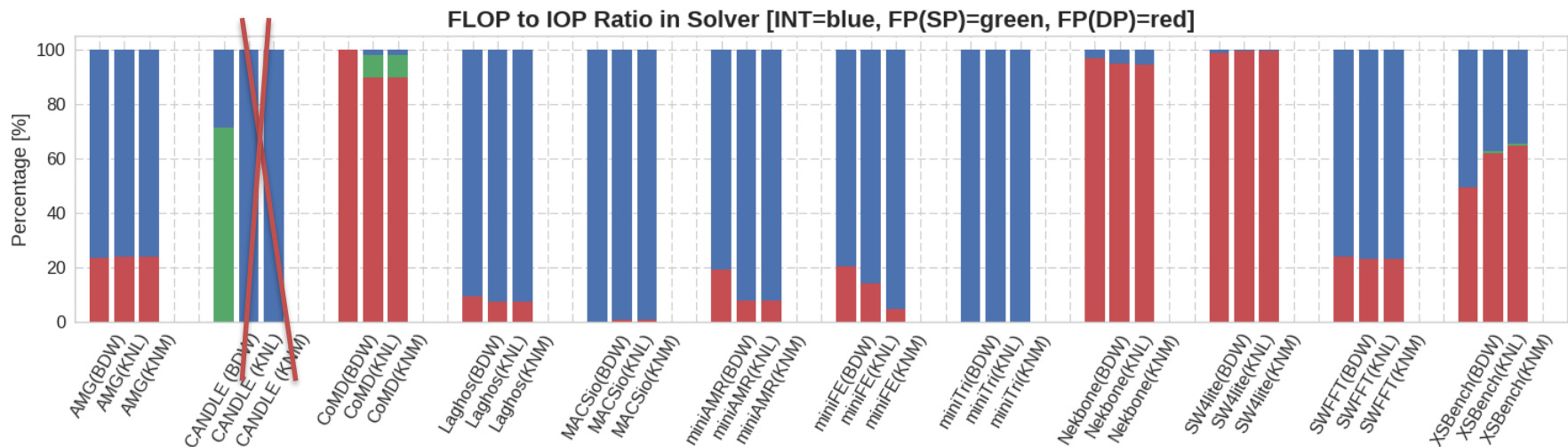
If we measure the things on top, we can get:

- Indications of **impact of # FPUs** on performance (and power)
- Understanding what are the **real requirements of HPC** applications
 - Data-centric?
- Indications of what can be optimized on current hardware
 - Manipulate frequency? (➔ similar to **READEX?**)
- Indications of how supercomputers, as a utility is impacted

Results – Breakdown %FP32 vs. %FP64 vs. %Integer

● Following: few examples of >25 metrics (many more in raw data)

➔ Integer (+DP) heavy (>50%; 16 of 22), only 4 w/ FP32, only 1 mixed precision



Results – Compare Time-to-Solution in Solver

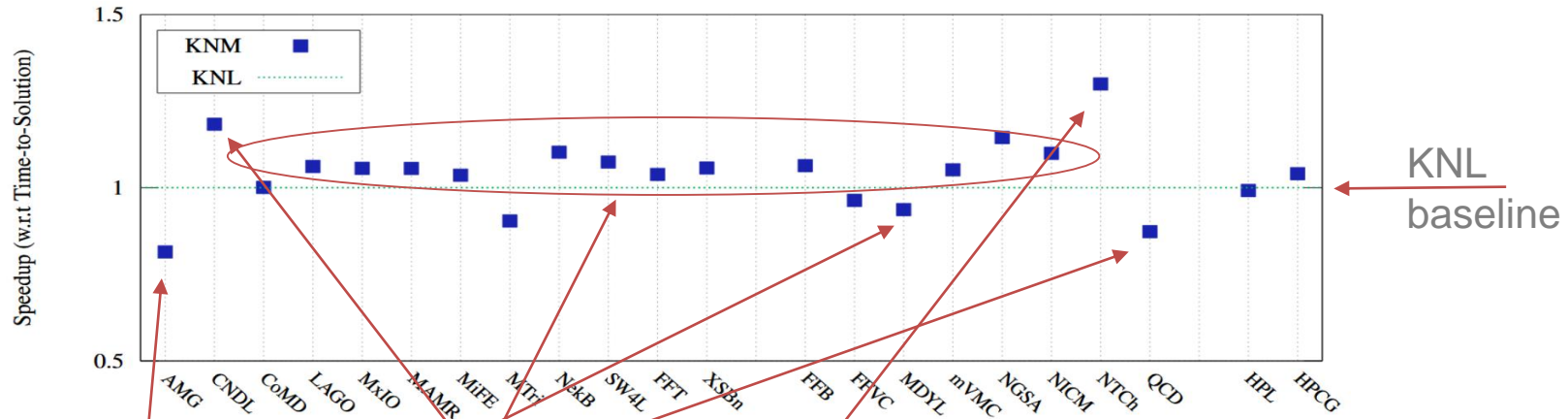


Fig. 4. Speedup of KNM over KNL as baseline. MiniAMR included since the input is the same for both Phi; Proxy-app abbreviations acc. to Section II-B

- Only 3 apps seem to suffer from missing DP (MiniTri: no FP; FFVC: only int+FP32)
- VNNI may help with CANDLE perf. on KNL; NTChem improvement unclear
- KNM overall better (due to 100MHz freq. incr.?)
- Memory throughput on Phi (in *cache mode*) doesn't reach peak of *flat mode* (only ~86% on KNL; ~75% on KNL)

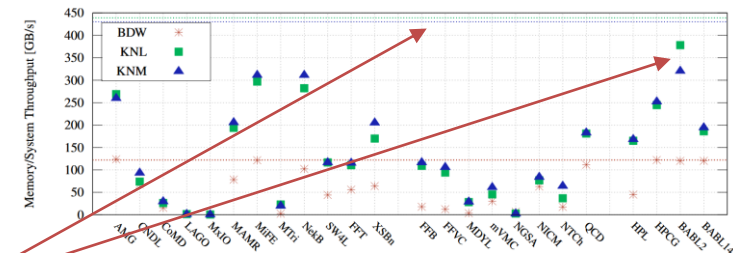
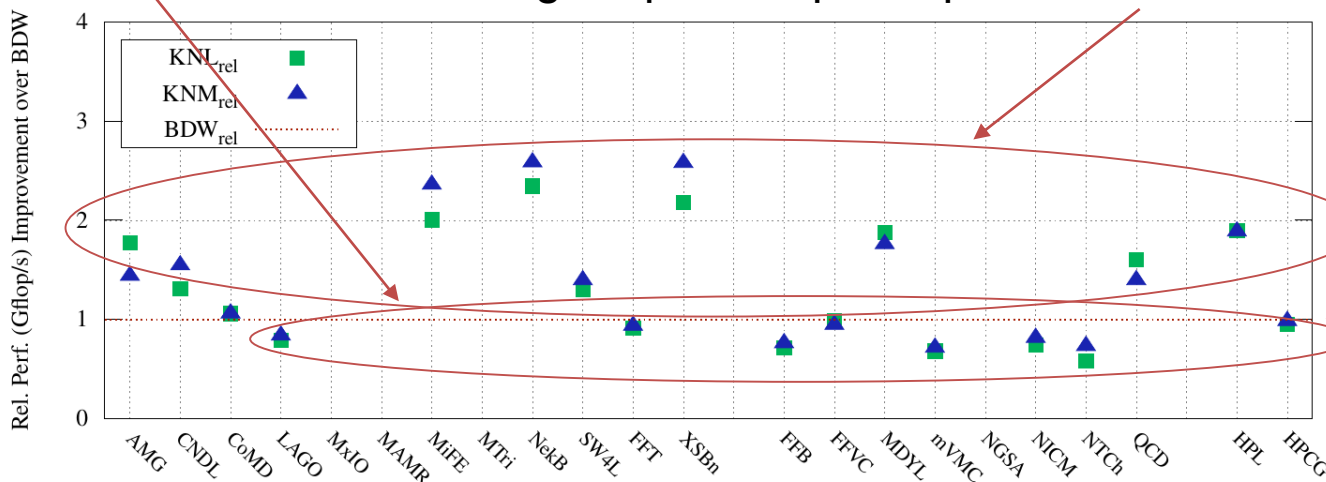


Fig. 5. Memory throughput (only DRAM for BDW, DRAM+MCDRAM for Phi) per proxy-app; Dotted lines indicate Triad stream bandwidth (flat mode, cf. Tab. I); BabelStream for 2 GiB (BABL2) and 14 GiB (BABL14) vector

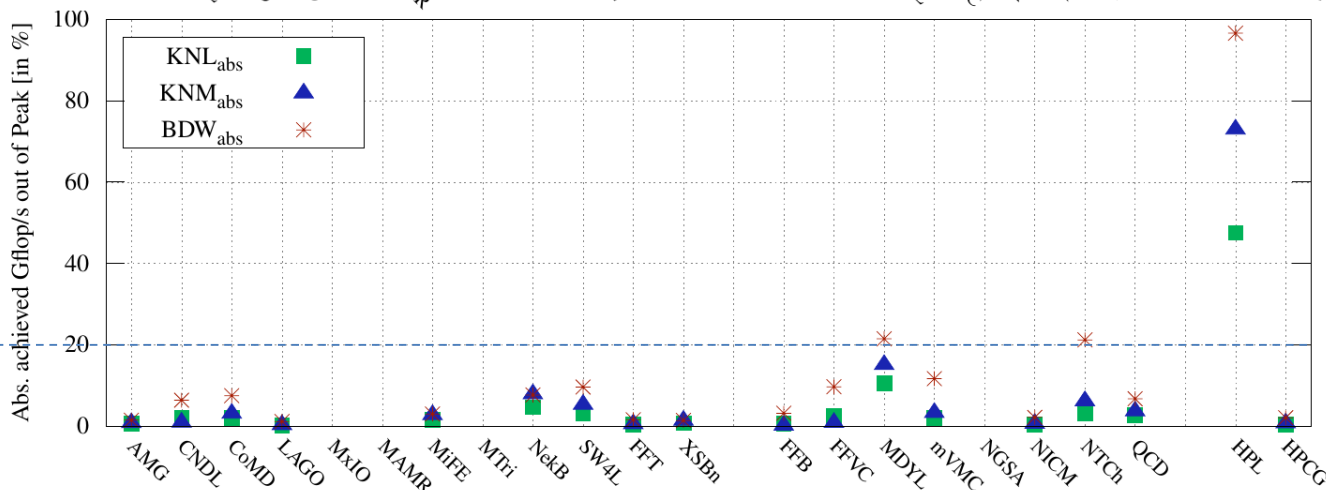
Note: MiniAMR not strong-scaling → limited comparability

Results – Compare Gflop/s in Comp. Kernel/Solver

- 8 apps out of 18: less Gflop/s on Phi than on BDW (ignoring I/O & Int-based apps)
- All apps (ignoring HPL) with **low FP efficiency**:
 $\leq 21.5\%$ on BDW, $\leq 10.5\%$ on KNL, $\leq 15.1\%$ on KNM (Why? → next slides)
- Phi performance comes from higher peak flop/s, lop/s and/or faster MCDRAM?



Relative
perf. over
BDW
baseline

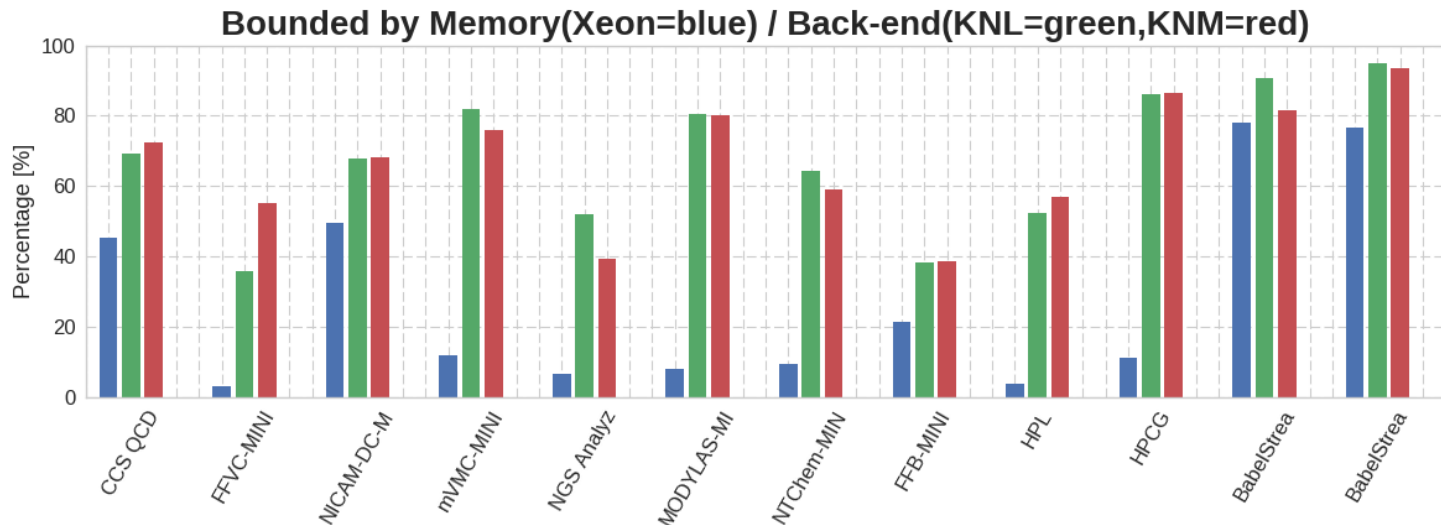
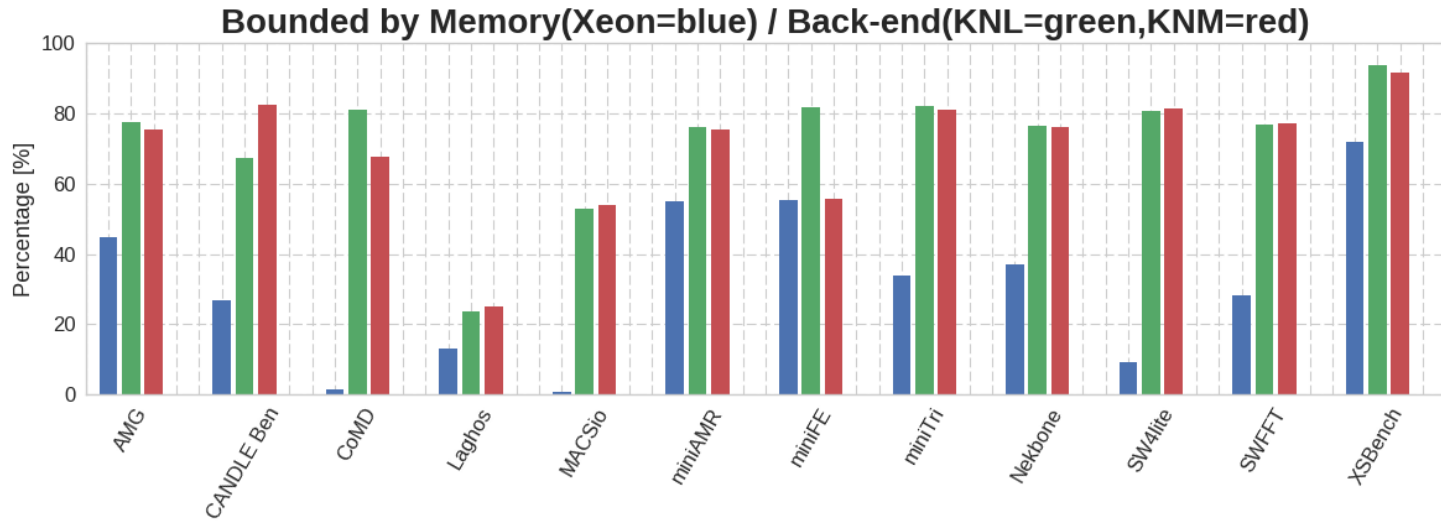


Absolute
Gflop/s perf.
compared to
theor. peak

20% of
theor. peak

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:

- Theory: Higher CPUfreq
 → faster compute?
 → compute-bound?
- 20 of 22 of apps below ideal scaling on BDW
 → not compute-bound
 → memory-bound?
- HPCG on Phi (vs. BDW):
 - no improve. w/ freq.
 - $\approx 2x$ mem. throughput
 - runtime $\approx 10\%$ lower
 → **memory-latency bound** (so, MCDRAM is bigger bottleneck)
 (→ one of Dongarra's original design goals)
- BDW: TurboBoost (TB) mostly useless for apps

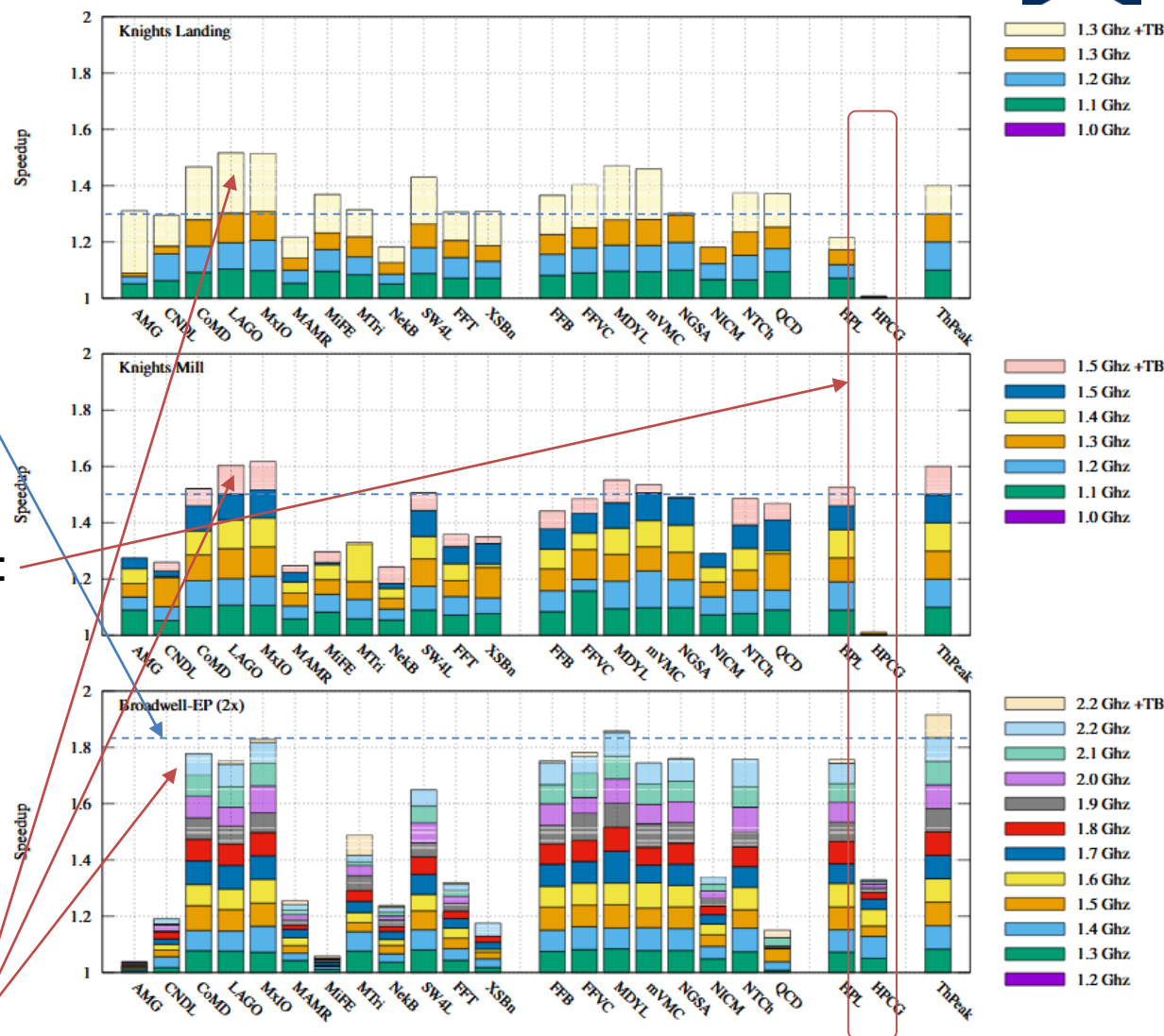


Fig. 6. Speedup obtained through increased CPU frequency (w.r.t baseline frequency of 1.0 Ghz on KNL/KNM and 1.2 Ghz on BDW); **Top plot: KNL, middle plot: KNL, bottom plot: BDW**; Theoretical peak (ThPeak): furthest

- **Supports** our previous **hypothesis** that most of the proxy-/mini-apps are **memory-bound**
- Outlier: only Laghos seems (intentionally?) poorly optimized
- Verifies our assumption about **optimization status of the apps** (→ similar to other HPC roofline plots)
- KNL/KNM roofline plots show nearly same results (omitted to avoid visual clutter)

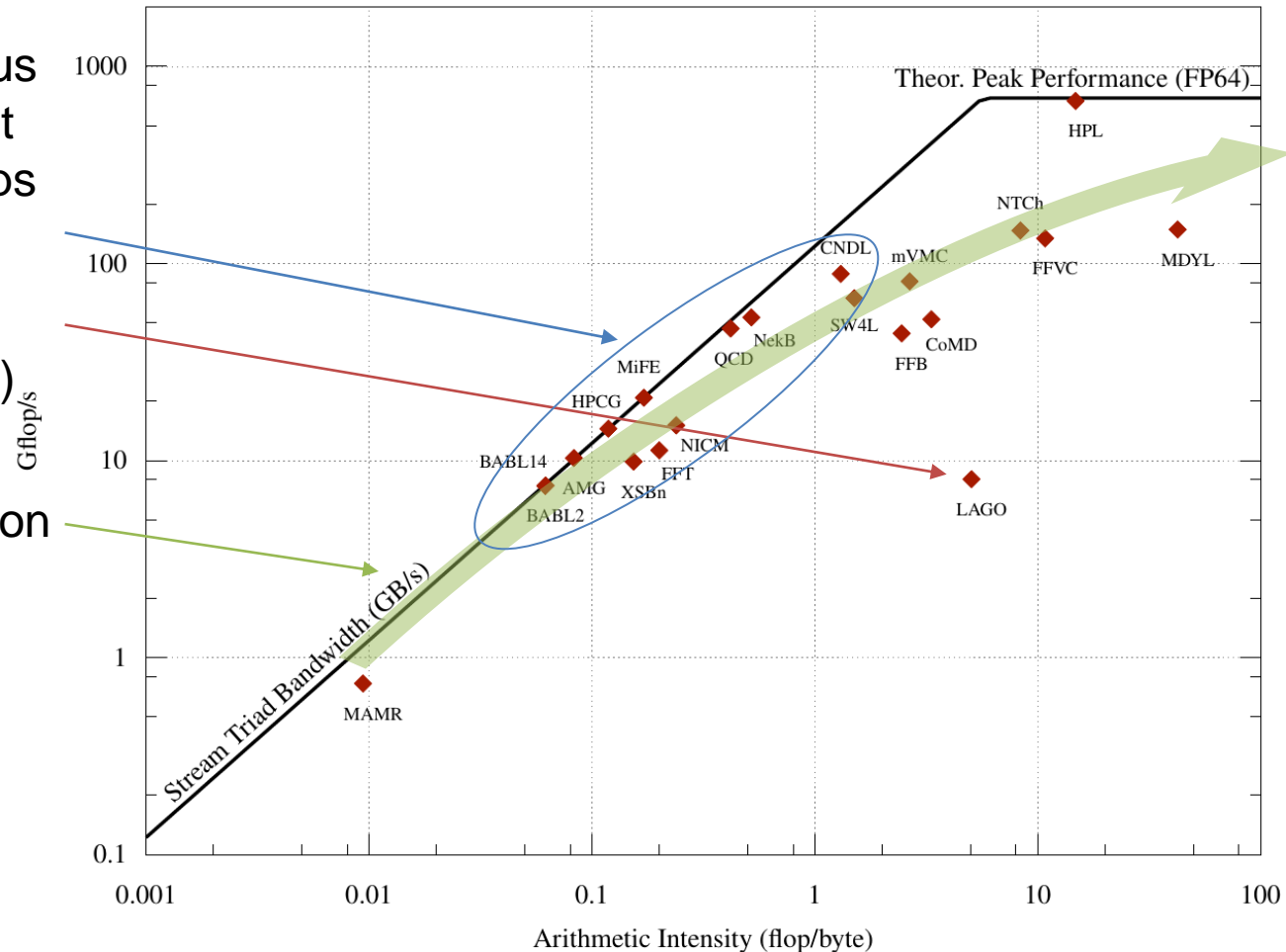


Fig. 5. Roofline plot (w.r.t dominant FP operations and DRAM bandwidth) for Broadwell-EP reference system; Filtered proxy-apps with negligible FP operations: MxIO, MTri, and NGSA; Proxy-app labels acc. to Section II-B

Results – Requirement for a “Weighted Look” at Results

- Studied HPC utilization reports of 8 centers across 5 countries
- **Not every app equally important** (most HPC cycles dominated by Eng. (Mech./CFD), Physics, Material Sci., QCD)

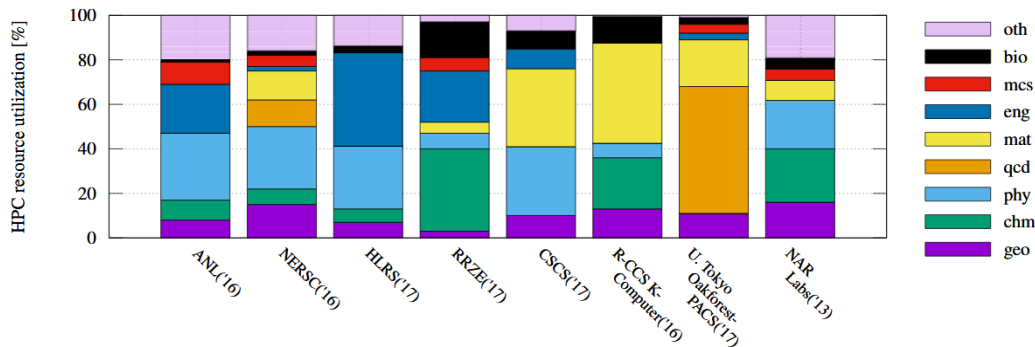


TABLE II

APPLICATION CATEGORIZATION, COMPUTE PATTERNS, AND MAIN PROGRAMMING LANGUAGES USED; MACSIO, HPL, HPCG, AND BABELSTREAM BENCHMARKS OMITTED

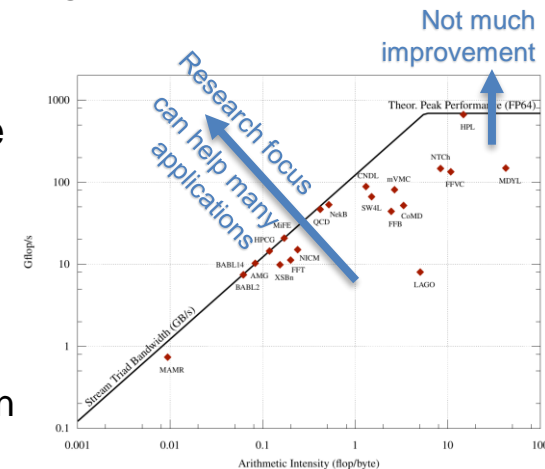
ECP	Scientific/Engineering Domain	Compute Pattern	Language
AMG	Physics and Bioscience	Stencil	C
CANDLE	Bioscience	Dense matrix	Python
CoMD	Material Science/Engineering	N-body	C
Laghos	Physics	Irregular	C++
miniAMR	Geoscience/Earthscience	Stencil	C
miniFE	Physics	Irregular	C++
miniTRI	Math/Computer Science	Irregular	C++
Nekbone	Math/Computer Science	Sparse matrix	Fortran
SW4lite	Geoscience/Earthscience	Stencil	C
SWFFT	Physics	FFT	C/Fortran
XSBench	Physics	Irregular	C
RIKEN	Scientific/Engineering Domain	Compute Pattern	Language
FFB	Engineering (Mechanics, CFD)	Stencil	Fortran
FFVC	Engineering (Mechanics, CFD)	Stencil	C++/Fortran
mVMC	Physics	Dense matrix	C
NICAM	Geoscience/Earthscience	Stencil	Fortran
NGSA	Bioscience	Irregular	C
MODYLAS	Physics and Chemistry	N-body	Fortran
NTChem	Chemistry	Dense matrix	Fortran
QCD	Lattice QCD	Stencil	Fortran/C

- Some supercomputers are “specialized”
 - **Dedicated HPC** (e.g.: weather forecast)
- For system *X* running **memory-bound** apps
 - **Why pay premium for FLOPS?**
 - NASA applies this pragmatic approach ²

² S. Saini et al., “Performance Evaluation of an Intel Haswell and Ivy Bridge-Based Supercomputer Using Scientific and Engineering Applications,” in HPCC/SmartCity/DSS, 2016



- FLOPS: de-facto performance metric in HPC ☹
 - Procurement (proxy)apps highly FP64 dependent, but often memory-bound?
 - Even for memory-bound apps (HPCG): Performance reported in FLOPS!!
 - ➔ Community move to **less FLOP-centric performance metrics?**
- Options for **memory-bound** applications:
 - Invest in memory-/data-centric architectures (and programming models)
 - Reduction of FP64 units acceptable ➔ reuse chip area
 - Move to FP32 or mixed precision ➔ less memory pressure
- Options for **compute-bound** applications:
 - Brace for less FP64 units (driven by market forces) and less “free” performance (10nm, 7nm, 3nm, ...then?)
 - **FP32 underutilized**
 - ➔ Research use of mixed/low precision without losing required accuracy
 - ➔ Remove and design FP64-only architectures
 - Libraries will pragmatically try to **utilize lower precision FPUs**
 - E.g.: use GPU FP16 TensorCores in GEMM (Dongarra’s paper at SC18)
 - If no library ➔ Take performance hit / rewrite code to use low precision units



Lessons-learned:

- IOP counting method may be misleading (→ instructions instead of ops?)
- **Fixing uncore** frequency is important
- Defining/**measuring memory boundedness is hard** ☹
- Intel MPI good on all Intel chips (i.e., default settings, rank/thread mapping)
- Intel's 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- vs. weak-scaling**
- Community effort into **one repo of HPC BMs** (similar to SPEC)?

Much more data/details in the full paper:

*Double-precision FPUs in High-Performance Computing:
An Embarrassment of Riches?*

Complete measurement **framework and all raw data** available:

<https://gitlab.com/domke/PAstudy>

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

Kazuaki Matsumura and Haoyu Zhang
Keita Yashima and Toshiki Tsuchikawa and Yohei Tsuji
(w/ add. input: Hamid R. Zohouri and Ryan Barton)

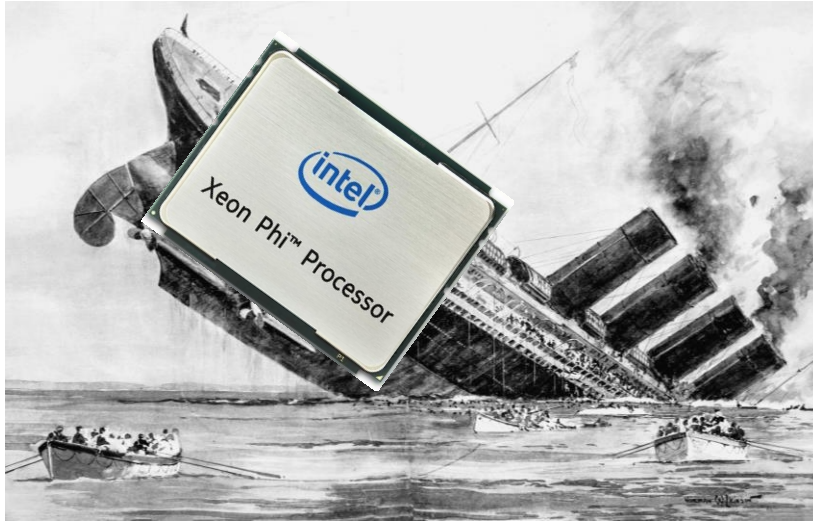
of the MATSUOKA Lab, Department of Mathematical and Computer Science,
Tokyo Tech and their supervisors:

Prof. Satoshi Matsuoka, Artur Podobas, and Mohamed Wahib (+ myself).

This work was supported by **MEXT**, **JST** special appointed survey **30593/2018**, **JST-CREST** Grant Number **JPMJCR1303**,
JSPS KAKENHI Grant Number **JP16F16764**, the New Energy and Industrial Technology Development Organization (**NEDO**),
and the **AIST/TokyoTech** Real-world Big-Data Computation Open Innovation Laboratory (**RWBC-OIL**).



Tokyo Tech



Postdoctoral Researcher

High Performance Big Data Research Team,
RIKEN Center for Computational Science,
Kobe, Japan

Tokyo Tech Research Fellow

Satoshi MATSUOKA Laboratory,
Tokyo Institute of Technology,
Tokyo, Japan

<http://domke.gitlab.io/>

jens.domke@riken.jp

