HPC Hardware

for

Numerical Linear Algebra Software

Piotr Luszczek

HPC Hardware Highlights

Looking at the Gordon Bell Prize

- Founding principles and goals

  Recognize outstanding achievement in high-performance computing applications and encourage development of parallel processing

- 1988 1 Gflop/s Cray Y-MP 8 processors
  - Static finite element analysis
- 1998 1 Tflop/s Cray T3E 1024 processors
  - Modeling of metallic magnet atoms, using a variation of the locally self-consistent multiple scattering method
- 2008 1 Pflop/s Cray XT5 1.5×10^5 processors
  - Superconductive materials
- 2017 20 Pflop/s Sunway TaihuLight 10×10^6 cores
  - Earthquake simulation
- 2018 1.33 Eflop/s (FP16) Titan 2×10^6 cores
  - Genome and brain science
- 2020 … 2022 Projected Exascale machines

Evolution Over the Last 30 Years

- Initially, commodity PCs where decentralized systems
- As chip manufacturing process shank to less than a micron, they started to integrate features on-die:
  - 1989: FPU (Intel 80486DX)
  - 1999: SRAM (Intel Pentium III)
  - 2005: on-chip cache coherency interconnect (AMD HyperTransport)
  - 2007: CUDA
  - 2009: GPU (AMD Fusion)
  - 2016: DRAM on chip (3D stacking)

Units of Measure and Prefixes

- High Performance Computing (HPC) units are:
  - flop: floating point operation, usually double precision unless noted
  - flop/s: floating point operations per second
  - bytes: size of data (a double precision floating point number is 8)
  - Note: computer scientists like to abbreviate byte(s) as B forgetting about units of loudness (Bells): dB.
- Typical sizes are millions, billions, trillions…

Mega	Mflop/s = 10^6 flop/s	MiB = 2^{20} = 1048576 ~ 10^6 bytes	nano ns = 10^{-9} seconds
Giga	Gflop/s = 10^9 flop/s	GiB = 2^{30} ~ 10^9 bytes	micro µs = 10^{-6} seconds
Tera	Tflop/s = 10^{12} flop/s	TiB = 2^{40} ~ 10^{12} bytes	milli ms = 10^{-3} seconds
Peta	Pflop/s = 10^{15} flop/s	PiB = 2^{50} ~ 10^{15} bytes	
Exa	Eflop/s = 10^{18} flop/s	EiB = 2^{60} ~ 10^{18} bytes	
Zetta	Zflop/s = 10^{21} flop/s		
Yotta	Yflop/s = 10^{24} flop/s		

- Currently fastest (public) machine ~ 150 Pflop/s (www.top500.org)

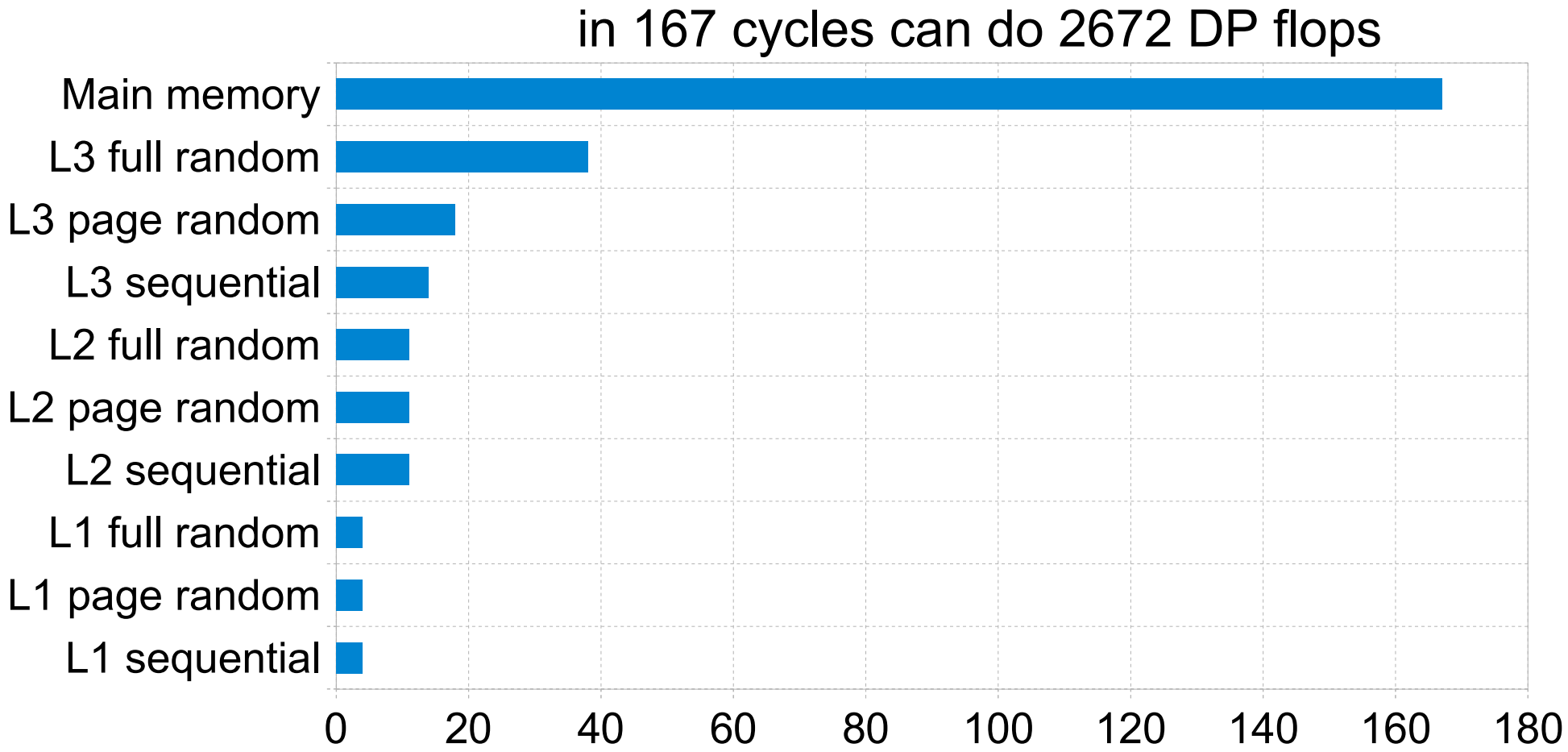
More Units: Power and Energy

- Power
  - Watt
    - Typical server CPU: 100-200W
    - Typical server GPU: 200-300 W
  - MW
    - Fueling a car: 5-10 MW
    - Supercomputer: 10+ MW
    - Data center: 50 MW
  - GW
    - Power house: 1-2 GW
- Energy
  - Joule
    - Single register transfer/instruction: 1 pJ
  - kWh
    - Electricity cost: 10 cents per kWh

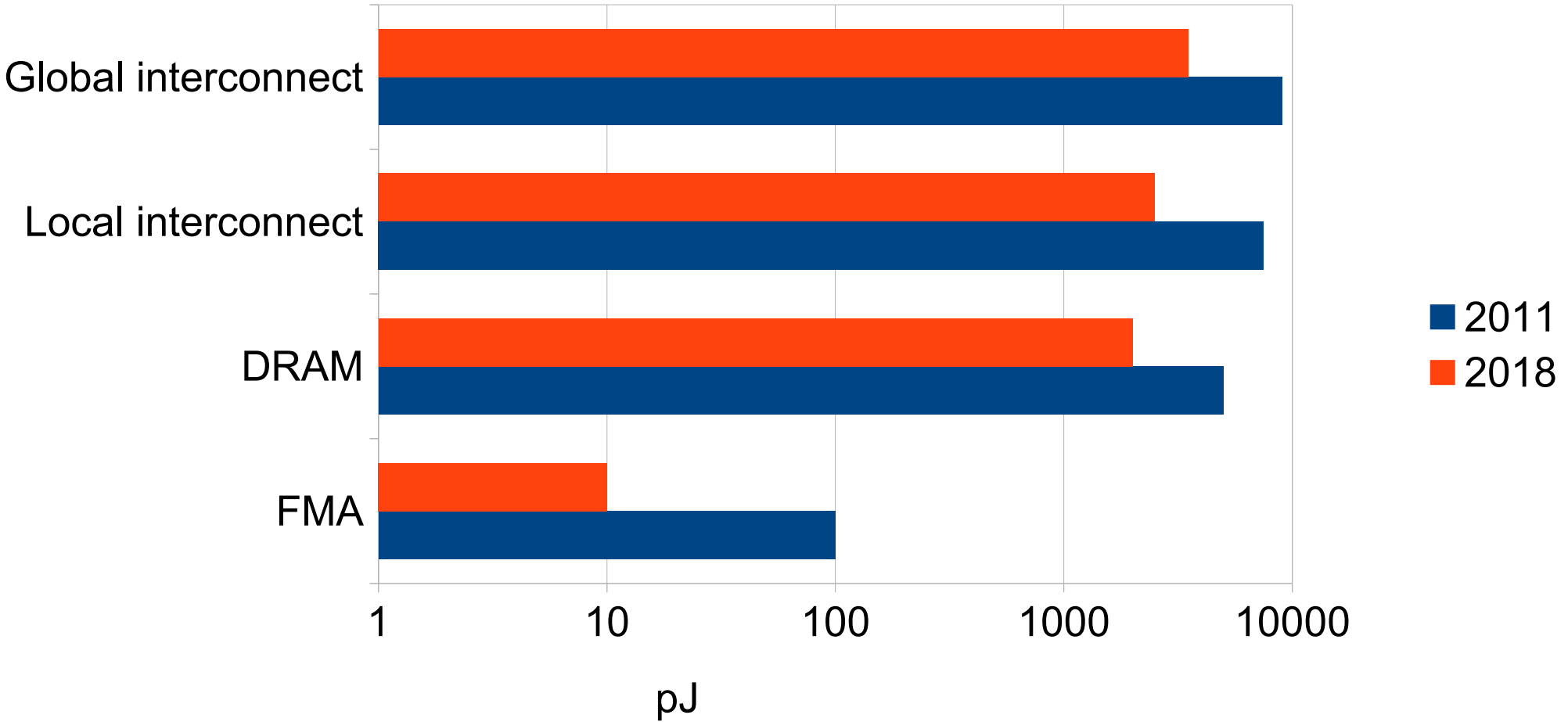
Floating Point Operations Per-Cycle Per-Core

- Most of the recent computers have FMA (Fused multiple add):
- x ←x + y*z in one cycle without intermediate rounding
- Intel Xeon (earlier models) and AMD Opteron have SSE2
  - 2 flops/cycle DP, 4 flops/cycle SP
- Intel Xeon Nehalem (’09) & Westmere (’10) have SSE4
  - 4 flops/cycle DP, 8 flops/cycle SP
- Intel Xeon Sandy Bridge(’11) and Ivy Bridge (’12) have AVX
  - 8 flops/cycle DP & 16 flops/cycle SP
- Intel Xeon Haswell (’13) & Broadwell (’14) AVX2
  - 16 flops/cycle DP & 32 flops/cycle SP
- Intel Xeon Phi (per core, '16)
  - 16 flops/cycle DP & 32 flops/cycle SP
- Intel Xeon Skylake (server) and Knight’s Landing AVX 512
  - 32 flops/cycle DP & 64 flops/cycle SP

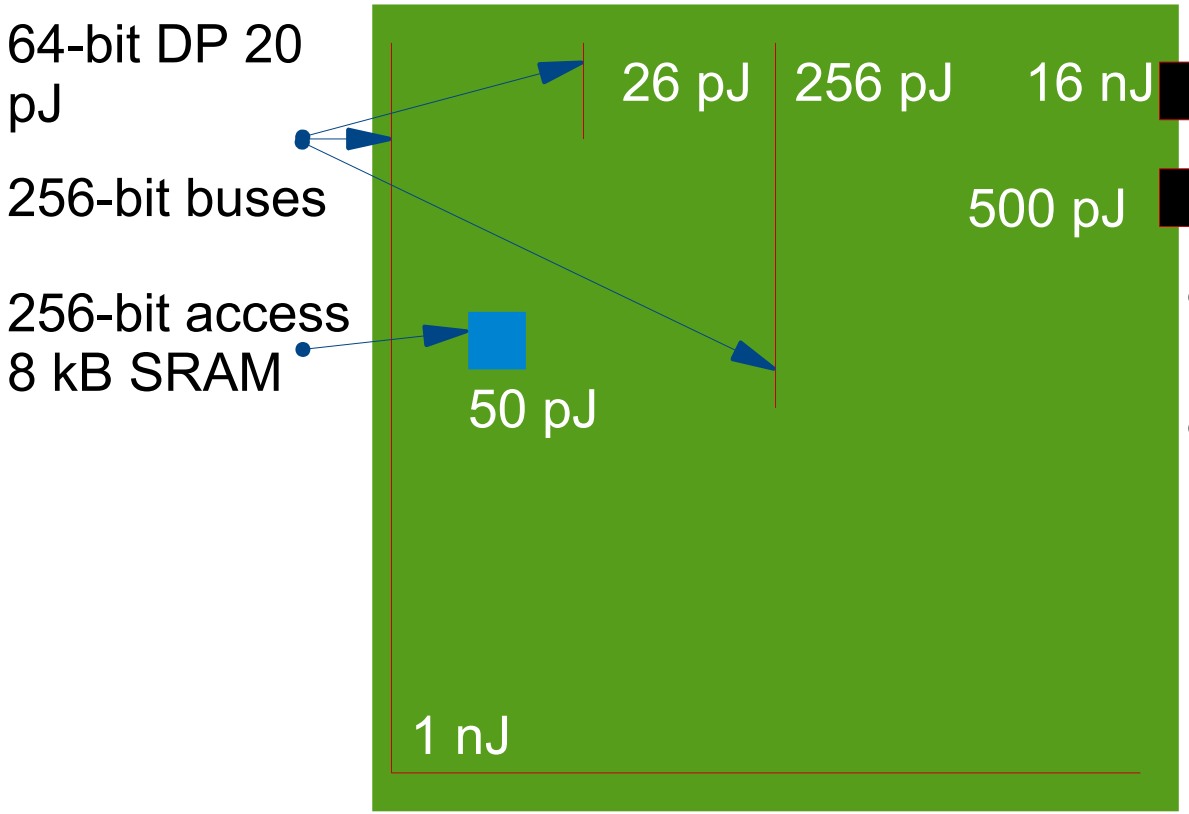
Data Access Latency in CPU Clock Cycles



Power Cost of Computation and Communication



Power Costs for NVIDIA 28nm Chips



- Floating point operation costs 20pJ
- Getting the operands from local memory
  - Situated 1mm away consumes 26pJ
  - If the operands need to be obtained from the other end of the die, it requires 1nJ
  - If the operands need to be read from DRAM, the cost is 16 nJ

Measuring Performance

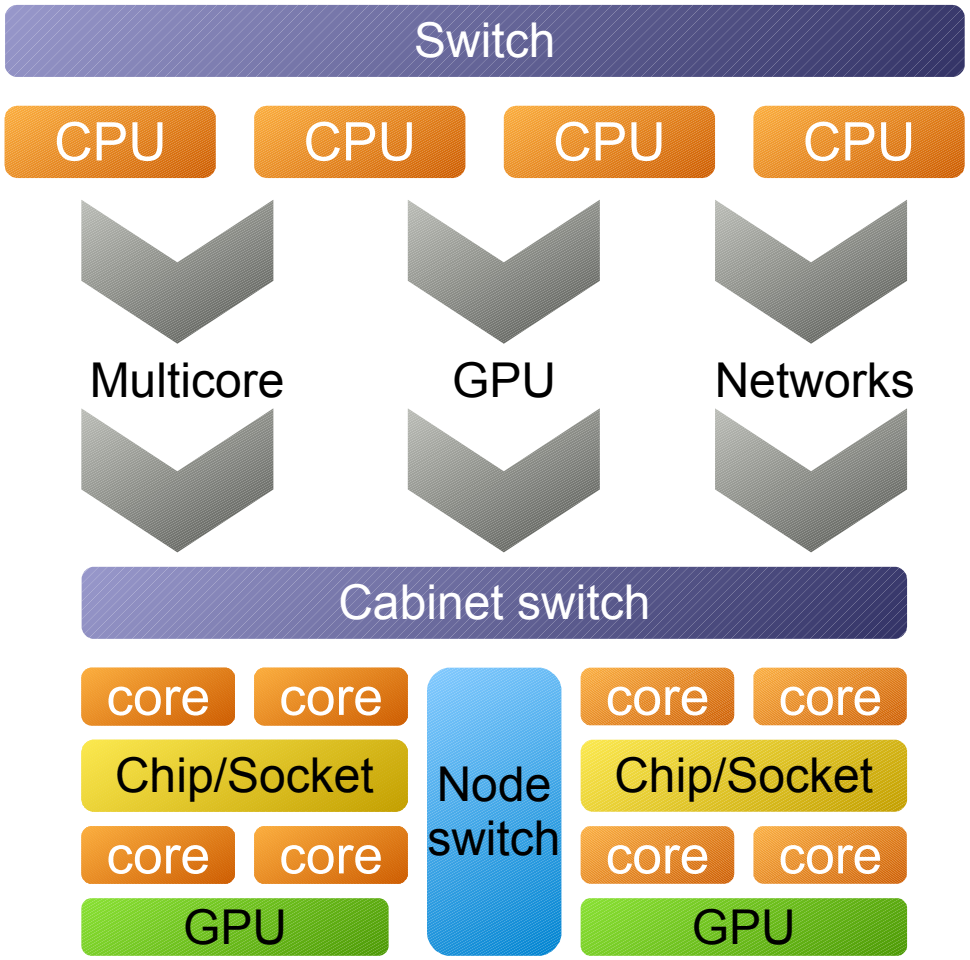
- Typical performance unit: flop/s
  - Precision: 64-bits (also called full precision)
  - Pipelining: not assumed (but often implemented)
    - FPU starts an instruction every 1 or 2 cycles
    - Instructions complete (retire) later
    - New instruction can be issued by FPU while others not finished
- Theoretical peak performance
  - Guaranteed by vendor not be exceeded
  - Counts additions and multiplications (or a combination)
- Example for Intel Xeon 5570 at 2.93 GHz
  - 4 flop's per cycle
    - Vectorization, FPUs, FMA
  - 11.72 Gflop/s per core
  - 46.88 Gflop/s per socket (maximum core count)

Classes of Floating Point Instructions

- Basic instructions
  - Addition and subtraction
    - A few cycles
    - Almost guaranteed to be pipelined
  - Multiplication
    - Around 10 cycles
    - Sometimes not pipelined
  - Division
    - Can take tens of cycles
  - Compound instructions
    - FMA
      - Less cycles than the total of add and multiplication
      - No intermediate rounding
    - MAD
      - Intermediate rounding does not provide advantage in precision

- Special instructions
  - Square root
  - Transcendental functions
    - sin, cos, exp, log, log2, log1
  - Quick estimates
  - Floating format inquiry and manipulation
    - isnan(), isinf()
    - conversions

Historical Changes in Typical Parallel Machines



Moving Data Between CPU and Memory

- x = A[0]
- Compiler generates a load instruction
- Hardware has to deliver the data
  - Try L1 cache (local to core)
  - Try L2 cache (local to few cores)
  - Try L3 cache (local to socket)
  - If data found in cache then
    - Update higher level caches
    - Move data to register
  - If not found: must be in main memory
- In addition to cache misses, page misses may happen as well
- Entire cache line (64 bytes) is moved through memory
- It is possible to bypass cache
  - Streaming or non-temporal loads

- A[0] = x
- Compiler generates store instruction
- Hardware takes over all memory levels
  - Store data in L1 cache
  - Store data in L2 cache
  - Store data in L3 cache
- Caches are updated for some caches
  - Wright through policy delivers data down one level (write-back doesn't)
  - Caches may use the same policy type
- If the data is not in cache then
  - Data is loaded for inclusive caches
  - Not be loaded for exclusive caches
- Non-temporal stores bypass cache
  - Faster for large data (no cache overhead)

Today's Compute Node: Commodity+Accelerator

- Server CPU
  - Intel Xeon Skylake
  - 18+ cores
  - 3+ GHz ALU
  - 3+ GHz FPU
    - Now different than ALU frequency
  - 18*32 ops/cycle
  - 576 Tflop/s (FP64)
  - 100 GB/s to main memory

Interconnect
PCIexpress 3
16 lanes,16 GB/s

- Accelerator (GPU)
  - NVIDIA Tesla P100 Pascal
    - 64 CUDA cores/SM
    - 3584 CUDA cores total
    - Base clock 1.328 GHz
    - Boost clock 1.480
    - 5304 Gflop/s (FP64)
    - 900 GB/s to main memory

NVLink
100,200 GB/s

Problem with Multicore Chip Design

- Today's design limitations
  - Compute performance increases with
    - Moore's Law
    - Increased core count
  - Bandwidth increases slowly
    - Limited by pins
  - Latency decreases at the slowest rate
    - Limited by copper messaging speed
- Next generation will be more integrated
  - Better protocols for accelerator computing
    - Support for unified memory and OS by-pass
  - 3D memory design
    - HBM, HBM2, HMC
  - Photonic on-chip networks
    - For example: HPE photonics

Future Computer Systems

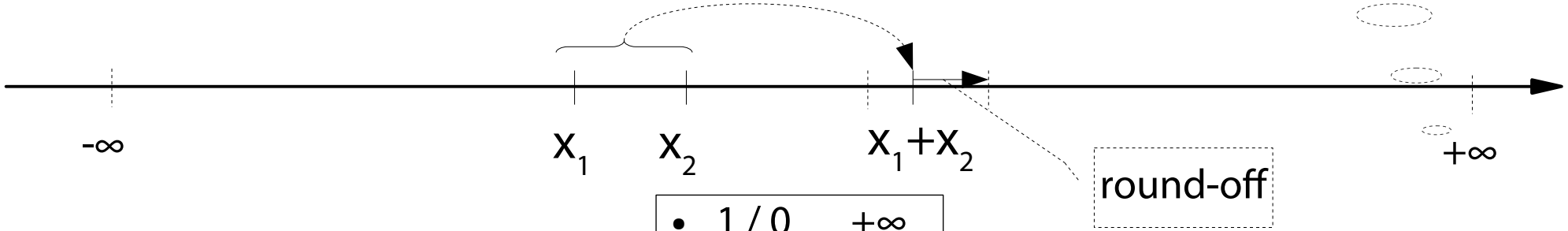
- Hybrid design (most likely)
  - Think standard multicore chips and accelerator (GPUs)
- Today's accelerators attached over slow links (PCIe)
  - Next generation more integrated
    - NVLink
    - OmniPath
    - Open CAPI
- Compute/Communication
  - flop/s: 59%, B/s: 23%
  - latency -5%
- AMD's Fusion, HSA, RocM
  - APU Multicore with embedded graphics ATI
  - New Zen and Ryzen
- Intel's Xeon Phi
  - Terascale, SCC, Larrabee, Knights Ferry
  - Knights Corner:
    - 61 cores, 244 hyper-threads
  - Knights Landing
    - 72 cores, 4 quadrants
  - Knights Mill
    - Machine learning instructions
  - Knights Hill:DOE workloads, cancelled
- NVIDIA Tesla
  - Fermi, Kepler:~15 SM, ~3000 CUDA cores
  - Pascal, Volta
    - 60-80 SM, 3800+ CUDA cores (FP32)
    - 100+ Tflop/s (FP16)

Representation of Errors in Numerical Libraries

Floating-Point Representation and Math Operations

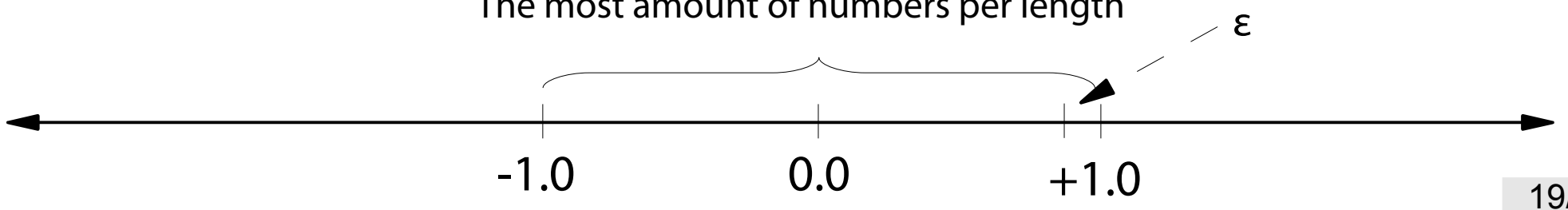
round up to:
$(x_1+x_2)+\varepsilon = x_1 \; C \; x_2$

Infinity is a (special) number



- 1 / 0 +∞
- -1 / 0 - ∞
- 0 / 0 NaN

The most amount of numbers per length



Problem Conditioning, Errors

- How is perturbation in input affecting the output?
  - Ideal: $y = f(x)$
  - Implementation: $\bar{y} = f(\bar{x})$
- Condition number
  - $|\bar{y}-y| / |y| \sim K_f(x) |\bar{x}-x| / |x|$
- Problem: we rarely know x and y
- Backward error for Ax = b
  - $||Ax-b|| / (||A|| ||x|| + ||b||) / \text{epsilon} / O(n) < 1$
  - Works even if A is singular

IEEE-754 2008 Formats

Mandatory and hardware-supported (fast)

Precision	Width	Sign bits	Mantissa bits	Exponent bits
Double	64	1	52	11
Single	32	1	23	8

Optional and software-supported (slow)

Precision	Width	Sign bits	Mantissa bits	Exponent bits
Quadruple	128	1	112	15
Double-Extended	80	1	64	15
Single-Extended	48	1	39	10
Half	16	1	10	5

IEEE Components

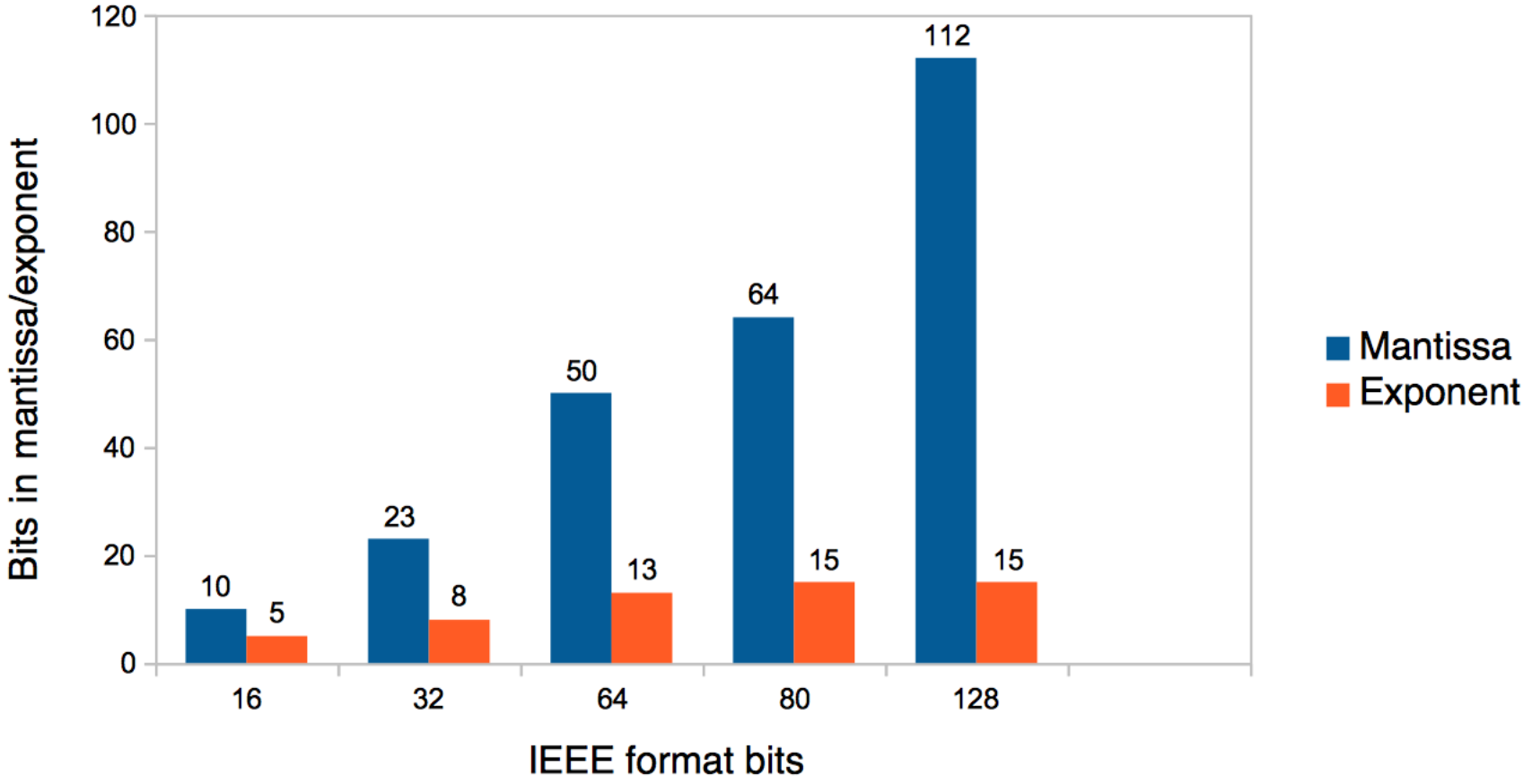
- Sign, mantissa (significant), exponent
- Precision
  - Size of mantissa
- Machine epsilon
  - Smallest distance between 1 and number closest to 1
- ULP(x)
  - Unit in the last place
  - Distance to “nearest number”
- Normalization
- Zero
  - Signed, exponent=0
- Denormals
  - No flushing!
  - Gradual underflow
- Rounding
  - Up/Down
  - Towards zero
  - To nearest (default)
- Exceptions: invalid op., x/0, over/under-flow, inexact
- Special values
  - NaN (quite, signaling)
  - Infinity (positive, negative)
- Endianess
  - Big
  - Little

Floating Point Formats IEEE 754 (2008)

Precision	Width	Exponent bits	Mantissa bits	Epsilon	Max
Quadruple	128	15	112	$O(10^{-34})$	$1.2x10^{4932}$
Extended	80	15	64	$O(10^{-19})$	
Double	64	11	52	$O(10^{-16})$	$1.8x10^{308}$
Single	32	8	23	$O(10^{-7})$	$3.4x10^{38}$
Half*	16	5	10	$O(10^{-3})$	65504

*Only storage format is specified

IEEE Floating Point Highlights



Initial Work with FP16

- ArXiv: 1502.02551v1 [cs.LG] 9 Feb 2015
- Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan (IBM T J Watson)
- Pritish Narayanan (IBM Almaden)
- Deep Learning with Limited Numerical Precision
- “[...] we observe the rounding scheme to play a crucial role [...]”
- “using stochastic rounding”
- “accelerator that implements low-precision fixed-point arithmetic with stochastic rounding”

1.88 Exa-Ops on Summit

- Algorithm packaged in the comparative genomics application Combinatorial Metrics (CoMet)
  - Custom Correlation Coefficient method, specializes in comparing variations of the same genes (alleles) present in a given population
  - Analyzes datasets composed of millions of genomes (previously impossible)
  - Study variations between all possible combinations of two or three alleles at a time.
  - Uncovers hidden networks of genes in plants and animals that contribute to observable traits
    - Biomarkers for drought-resistance in plants or disease in humans
- Principal investigators
  - Daniel Jacobson (Computational Biologist) and
  - Wayne Joubert (computational scientist)

FP16 Hardware (Current and Future)

- AMD
  - MI5, MI8, MI25
- ARM
  - NEON VFP has FP16 in V8.2-A
- Intel
  - Xeon CPUs (vectorized conversions)
- NVIDIA
  - Pascal: P100, TX1, TX2
  - Volta: Tensor Core (DGX-2)

- Supercomputers
  - Piz Daint
  - Summit
  - TSUBAME 3.0
  - Tokyo Tech
- Cloud
  - Google Cloud with P100 & V100
  - Azure: 3x P100 & V100
  - Amazon Web Services: 8x V100

Coding Data Types: C/C++, Fortran

- long double
  - 80 or 128 bits
- double
  - 64 bits
- float
  - 32 bits
- _Float16 (short float in C++20)
  - 16 bits
- __half, __half2 (cuda_fp16.h)
  - 2x16 bits
- half in Apple's Metal

- real*16
  - 128 bits
- real*8
  - 64 bits
- real*4
  - 32 bits
- real*2
  - 16 bits

Hardware/Software Support: Assembly and Intrinsics

- x86 assembly language
  - CVTSH_SS, CVTSS_SH
  - emmintrin.h
    - _cvtss_sh(), _cvtsh_ss()
  - f16cintrin.h → x86intrin.h
    - _mm_cvtph_ps(), _mm_cvtps_ph(), _mm256_cvtph_ps(), _mm256_cvtps_ph()
- PTX assembly language
  - cvt.f16.*
  - fma.f16x2
- ARM assembly language
  - vld1_f16, vst1_f16, vcvt_f16_f32

Current FP16 Hardware Implementations

- FLOAT16
  - NVIDIA Pascal
    - Intermediate rounding for FMA
  - NVIDIA Volta
    - Tensor Cores
      - May process FMA in FP32
- BFLOAT16
  - Google TPU 2
  - Intel Xeon Cascade

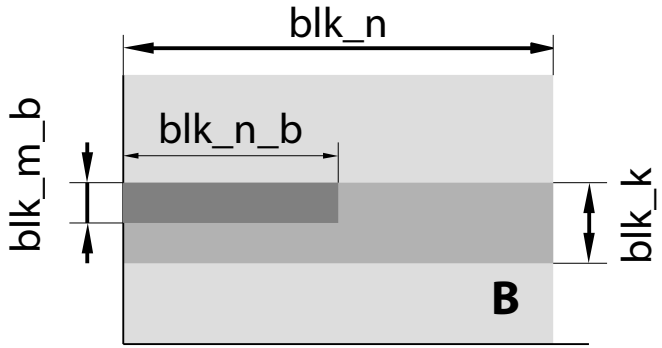
Sample of Optimization Techniques

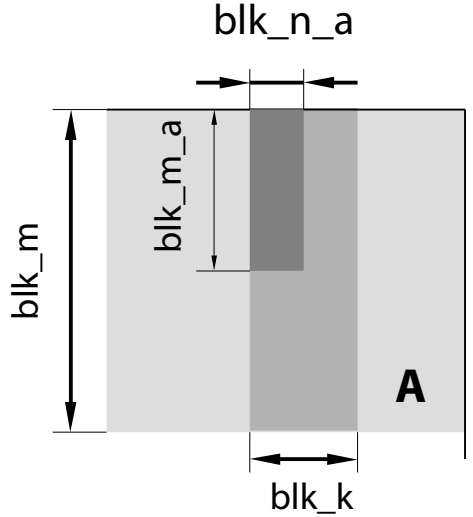
Sample Vector Addition and Possible Optimizations

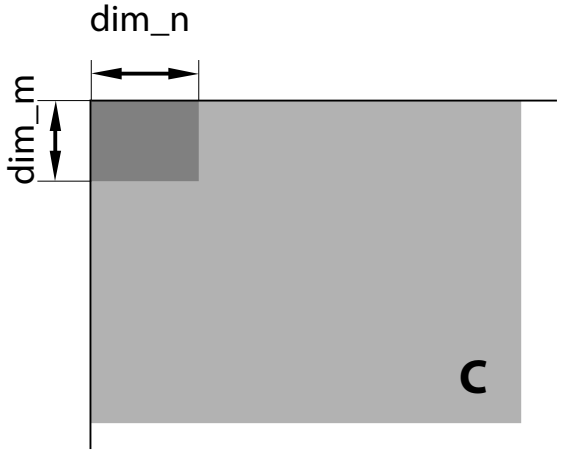
- For i=0,1,...N-1
  - C[i] = A[i] + B[i]
- Translates to loop around
  - Load A, B
  - Add
  - Store C
  - Check loop counter
- If we use FMA instruction
  - We can do more work without additional cost
    - C[i] = X[i] * A[i] + B[i]
- Vectorization may be done by the compiler or by hand
  - For i=0, 2, 4, … N-1
    - C[i+0] = A[i+0]+B[i+0]
    - C[i+1] = A[i+1]+B[i+1]
- If memory accesses are nearby, vectorize
  - Vector Load A[...], B[...]
  - Vector Add
  - Vector Store C[...]
  - Check loop counter
- There are vector equivalents of many instructions
- Vector length depends on hardware
  - AMD/Intel SSE, AVX, AVX-512
  - ARM Neon
  - IBM AltiVec

Example: Optimizing C = A*B

$$c_{ij} = \sum_k a_{ik} b_{kj}$$

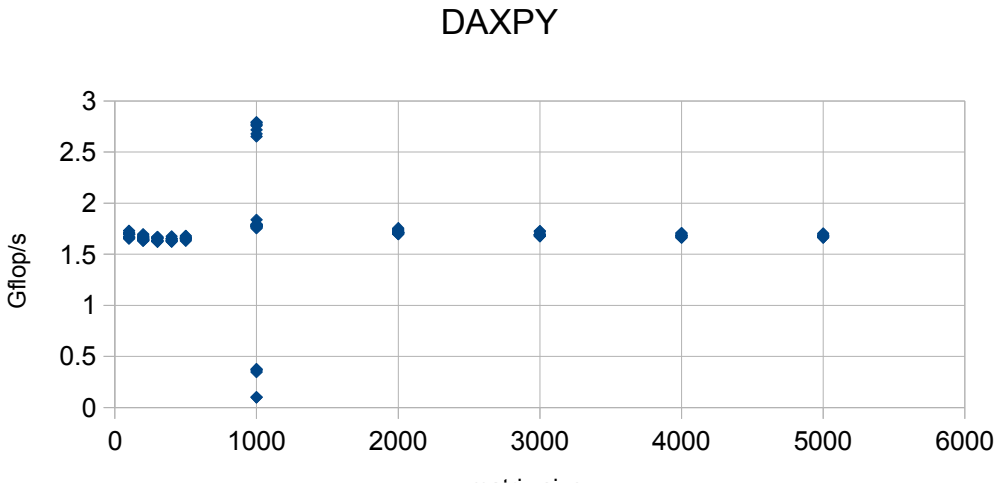


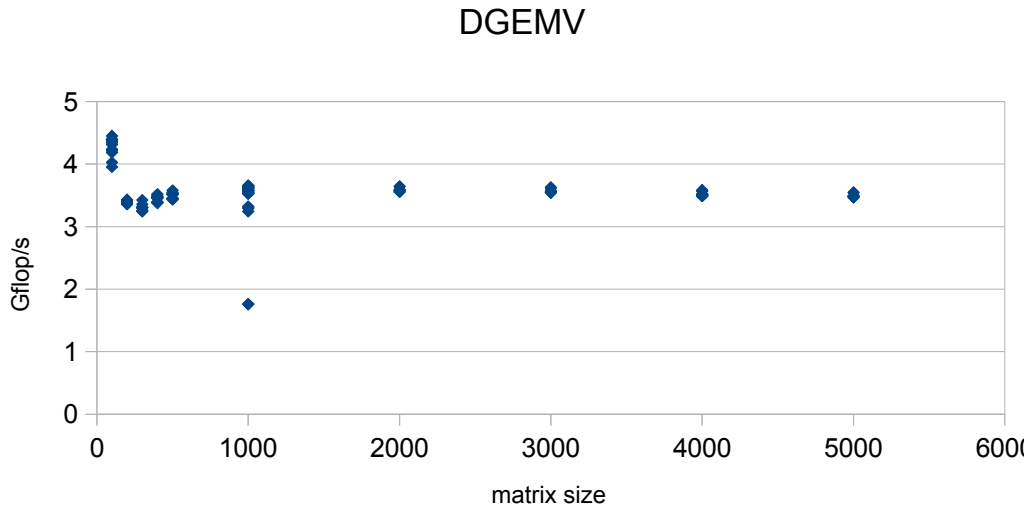


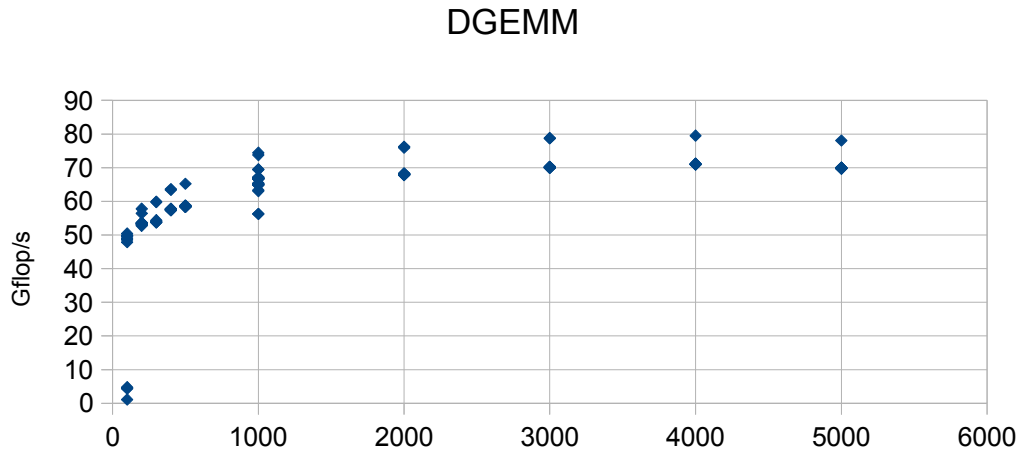


Measuring Time to Run Code and Computing Performance

In your time measurements, you often get a range of values. It is useful to look at the raw data.







Modeling the Expected Time

- For multiple measurements t_1, t_2, t_3, t_4, we need to find the useful value
  - Often the shortest time is the most interesting
    - All negative overheads were removed
    - All positive effects contributed
    - But you might get faster run if keep trying
- You should compute basic statistics to see
  - How far median is from average
  - What variance, moments, and outliers are
  - It may be possible that you see effects specific to your hardware
- Modeling time is also useful
  - $T1 = aN_1^3 + bN_1^2 + cN_1 + d$
  - $T2 = aN_2^3 + bN_2^2 + cN_2 + d$
  - …
  - If you have measurements T1, … then you may do a least square fit that minimizes error