中国科学院大学
夏季强化课程
2022

Fast Solvers for Large Algebraic Systems

Lecture 5. Mixed-precision methods

5

混合精度算法

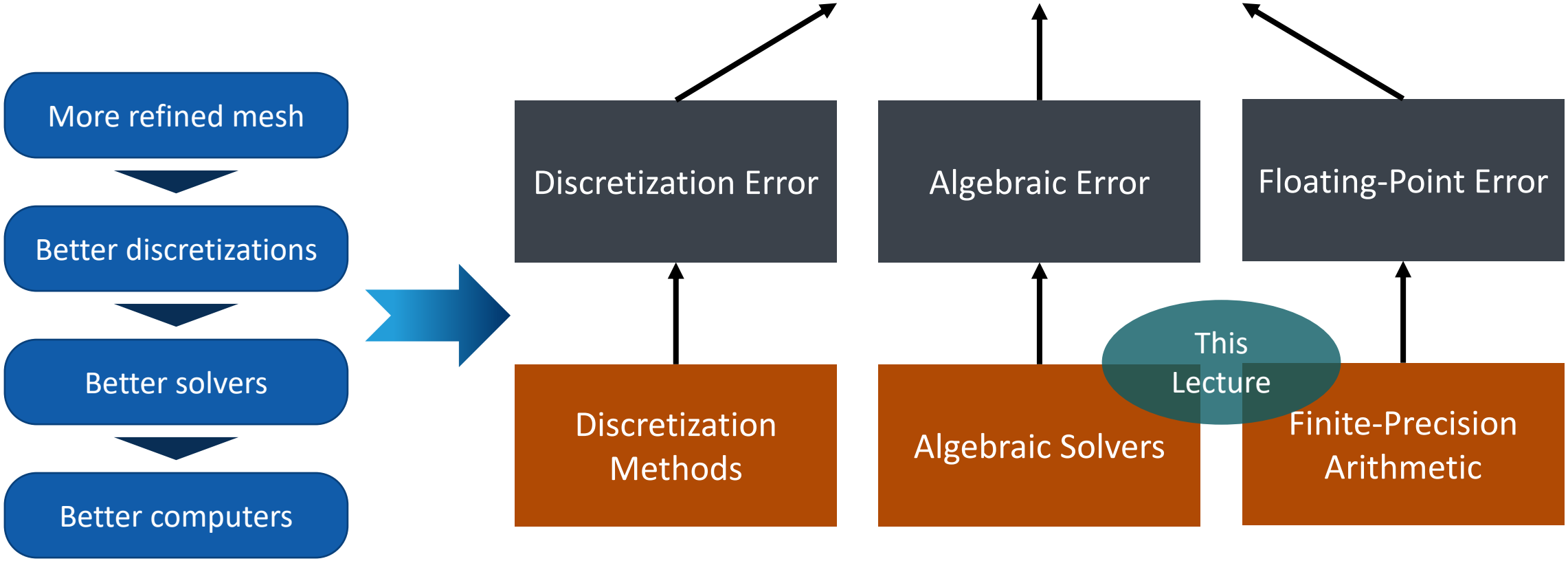
Chensong Zhang, AMSS

http://lsec.cc.ac.cn/~zhangcs

Table of Contents

Sources of Error in Simulation

Approximation: $u(x) = U_h(x) + \mathcal{E}_{\text{dis}} + \mathcal{E}_{\text{alg}} + \mathcal{E}_{\text{fp}}$



Introduction

Time to engage mixed-precision computing

/01

Accuracy of Direct Solvers, Revisited

N. Higham, 2002

- Floating-point computation (IEEE compatible)

$$\mathrm{fl}(x \ \mathrm{op} \ y) = (x \ \mathrm{op} \ y)(1+\delta), \quad |\delta| \le u$$

- Forward error of matrix-multiplications (depends on num of inner multiplications)

$$\text{Compute } C = AB, A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times k} \Longrightarrow |\hat{C} - C| \le \gamma_n |A||B| \qquad \gamma_n := \frac{nu}{1-nu}$$

- Backward perturbation analysis of the LU decomposition

$$(A+\delta A)\hat{x} = b, \quad |\delta A| \le \gamma_{3n} |\hat{L}||\hat{U}|$$

- Accuracy of direct solution methods

$$\frac{\|x-\hat{x}\|_\infty}{\|x\|_\infty} \le \gamma_{3n} \frac{\||\hat{L}||\hat{U}|\|_\infty}{\|A\|_\infty} \kappa_\infty(A)$$

Rounding errors exist and accumulate! This may prevent the computation from achieving sufficient accuracy.

Direct Methods + Iterative Methods, Revisited

- Multigrid methods: Using direct method as coarse grid solver
- Domain decomposition methods: Using direct method on local subdomains and “direct” preconditioner on interfaces
- Partial factorization: Incomplete factorizations as preconditioners (like ILU methods)
- Direct method used in part of a preconditioner (like MSP for the well part)
- Block iterative methods: Using direct method on sub-blocks (like Schwarz preconditioners)
- Low-precision direct method as a preconditioner (like GMRES-IR)
- Iterative refinement: improve precision of direct solvers (like LU-IR)
- Factorization of nearby problem as a preconditioner

Iain Duff. The SIAM Conference on Applied Linear Algebra. October 26-29, 2009. Monterey, California.

Some Historical Comments

- Long long ago … computing machines are very different from now
  - Fixed-point arithmetic ➔ Floating-point: SP, DP, QP, …
  - Floating-point arithmetic: Different standards for different machines
  - IEEE 754 Standard, Intel 8087 coprocessor, …
- Multi-precision arithmetic simulators available for a long time
  - Maple, Mathematica, …
  - Julia, half, numpy, …
- Solving an algebraic system to certain accuracy on a computer:
  - If the problem is ill-conditioned (sensitive to perturbation) …; see Lecture 2
  - If the precision is low, quality of solution could be very bad
  - The way to improve quality is using higher precision (too costly) or IR

Why Mixed-Precision Now

	NVIDIA A100	NVIDIA Turing	NVIDIA Volta
Supported Tensor Core Precisions	FP64, TF32, bfloat16, FP16, INT8, INT4, INT1	FP16, INT8, INT4, INT1	FP16
Supported CUDA® Core Precisions	FP64, FP32, FP16, bfloat16, INT8	FP64, FP32, FP16, INT8	FP64, FP32, FP16, INT8

- *"Because it's there…"* George Mallory, 1923
- Fast hardware with mixed-precision capability is available
- Make less data movement (see Lecture 6)
- Fill more data in the cache
- Traditional algorithms need to be modified
- Traditional algorithms must take advantage
- The main driving force is DL and it will continue

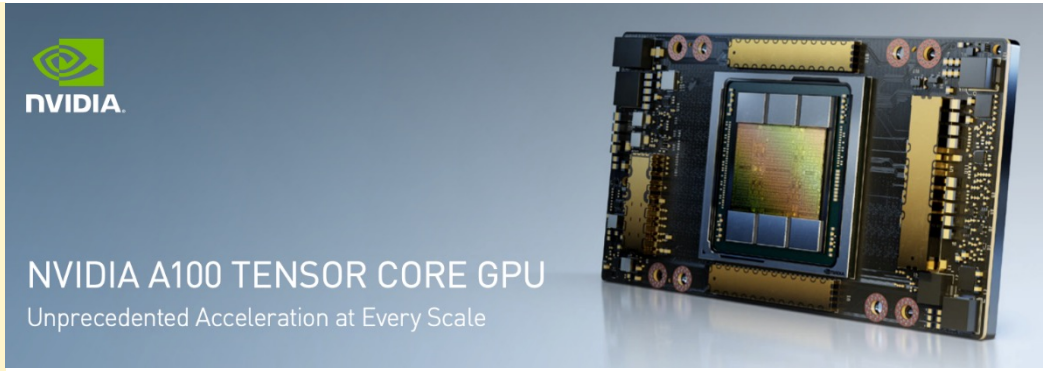
Tensor Cores introduced in Volta, 4X faster than regular FP16

	V100 PCIe	V100 SXM2	V100S PCIe
GPU Architecture	NVIDIA Volta		
NVIDIA Tensor Cores	640		
NVIDIA CUDA® Cores	5,120		
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS	8.2 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS	16.4 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS	130 TFLOPS
GPU Memory	32 GB /16 GB HBM2		32 GB HBM2
Memory Bandwidth	900 GB/sec		1134 GB/sec
ECC	Yes		
Interconnect Bandwidth	32 GB/sec	300 GB/sec	32 GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink™	PCIe Gen3
Form Factor	PCIe Full Height/Length	SXM2	PCIe Full Height/Length
Max Power Comsumption	250 W	300 W	250 W
Thermal Solution	Passive		
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC®		

Half precision, but not 2X faster

Driving Force: Deep Learning

- They are very tolerant of reduced-precision computations
- The computations performed by most models are compositions of a relatively small handful of dense linear algebra calculations . . .
- Many of the mechanisms developed over the past 40 years to enable general-purpose programs to run with high performance on modern CPUs . . . are unnecessary for machine learning computations



$$\underbrace{D}_{\text{fp16 or fp32}} = \underbrace{C}_{\text{fp16 or fp32}} + \underbrace{A}_{\text{fp16}} \underbrace{B}_{\text{fp16}}$$

Fused multiply–add (FMA) used in Tensor Core

Ref: J. Dean, "The Deep Learning Revolution and Its Implications for Computer Architecture and Chip Design," 2020 IEEE International Solid-State Circuits Conference (ISSCC), 2020, pp. 8-14.

Range of FP Representation

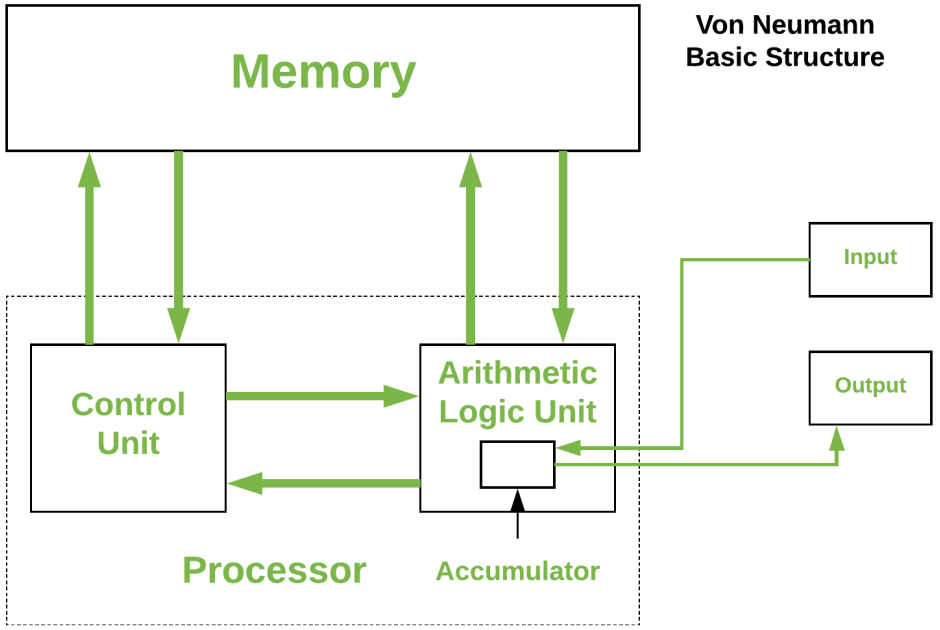
	signif.	exp.	u	$x_{\min}^{(s)}$	$x_{\min}$	$x_{\max}$
bfloat16	8	8	3.91×10^{-3}	9.18×10^{-41}	1.18×10^{-38}	3.39×10^{38}
fp16	11	5	4.88×10^{-4}	5.96×10^{-8}	6.10×10^{-5}	6.55×10^{4}
fp32	24	8	5.96×10^{-8}	1.40×10^{-45}	1.18×10^{-38}	3.40×10^{38}
fp64	53	11	1.11×10^{-16}	4.94×10^{-324}	2.22×10^{-308}	1.80×10^{308}
fp128	113	15	9.63×10^{-35}	6.48×10^{-4966}	3.36×10^{-4932}	1.19×10^{4932}

Number of bits + 1

- For fp16, it is very likely to overflow / underflow; for bfloat16, low precision
- We can ignore the small numbers (underflow)
- But the real trouble is: how to handle the large numbers?
- Need to scale the problem first

Taking A Detour

- Mixed-precision or multi-precision: More accurate result ➔ Better performance
- How to measure performance? We can compare wall time (actual computing time)!
- But we might solve different problems with different solvers on different computers …



Computer Hardware Model

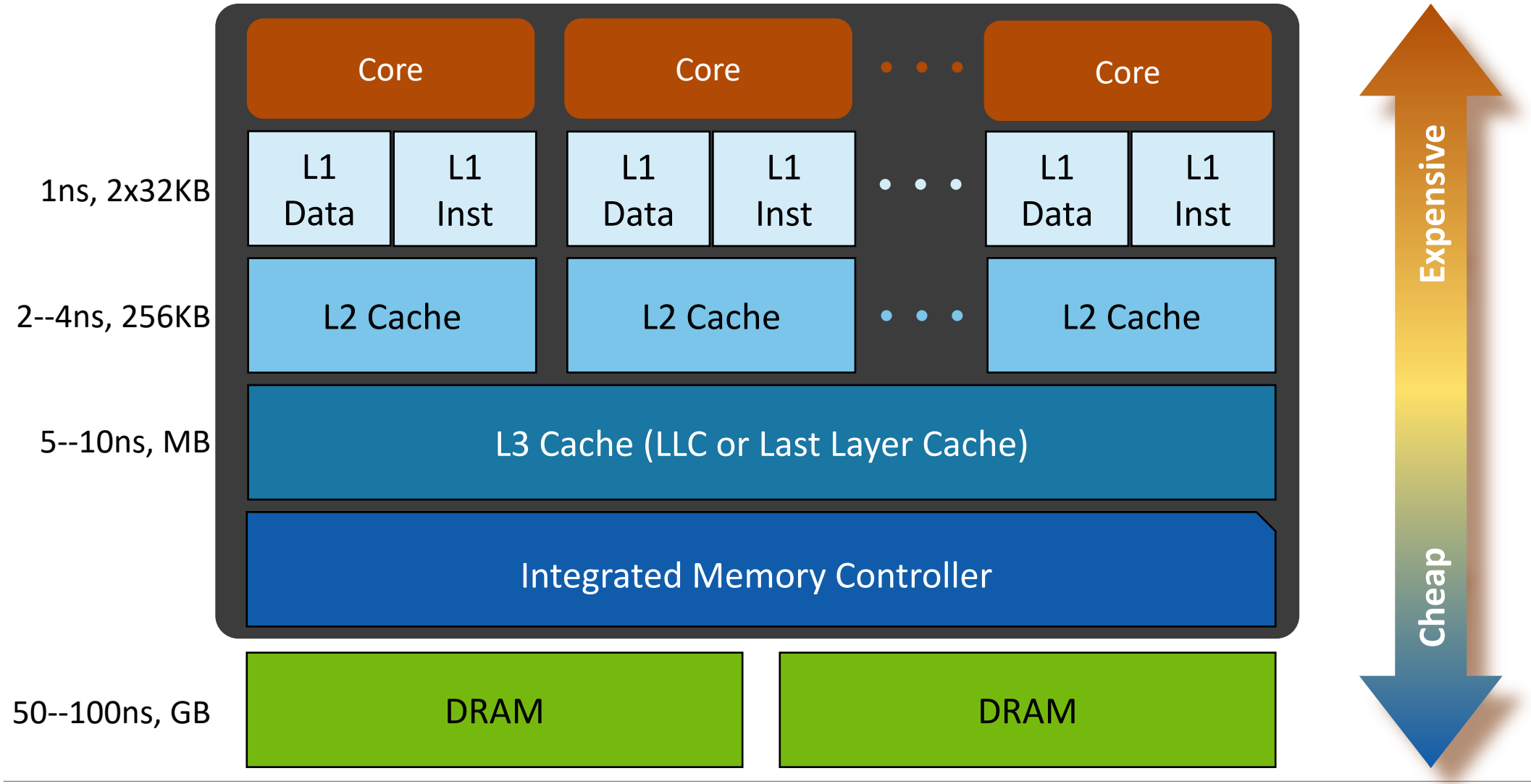
Computer Performance Model

Consider the performance of a computer algorithm

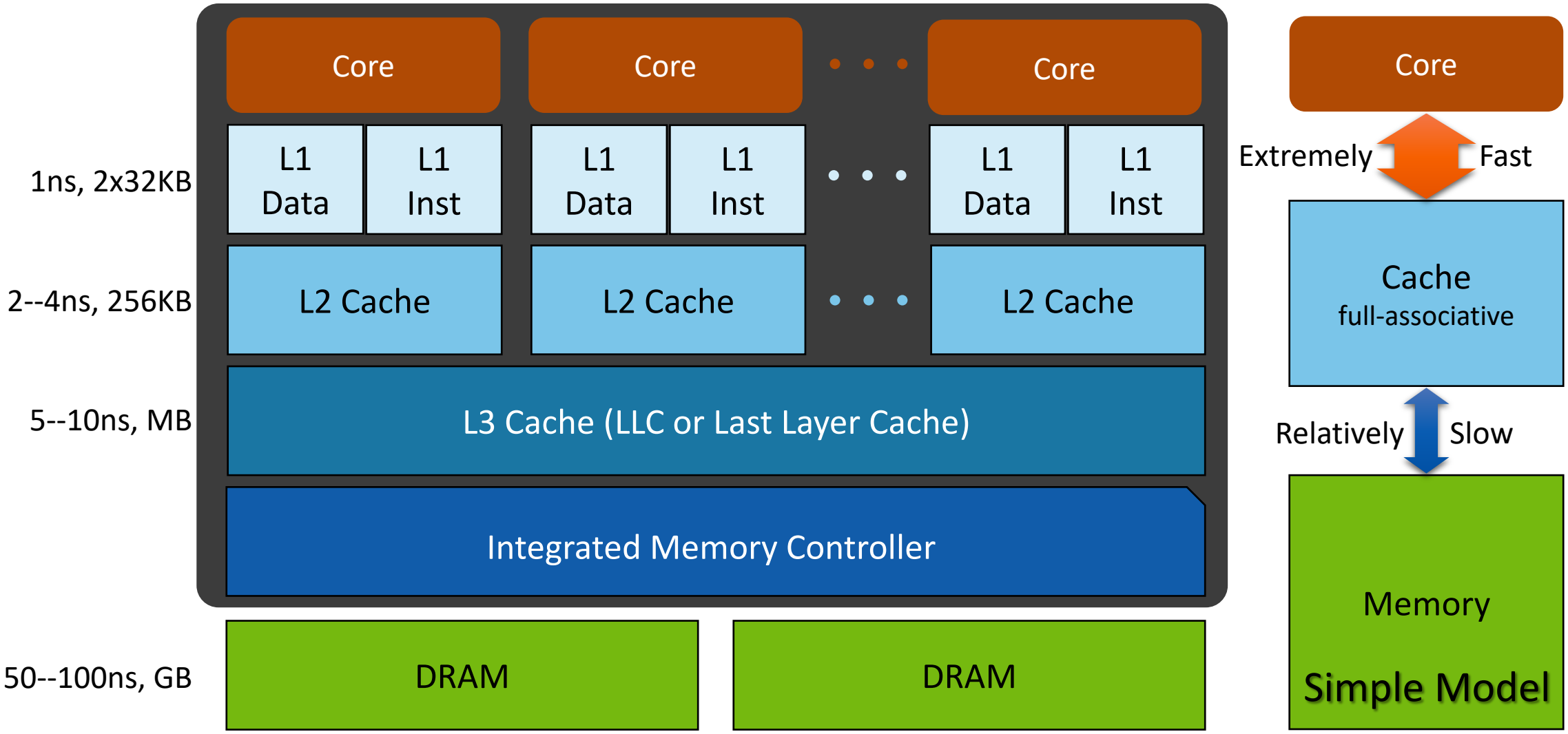
- ➢ computer
- ➢ algorithm
- ➢ problem

Source: https://www.geeksforgeeks.org/computer-organization-von-neumann-architecture/

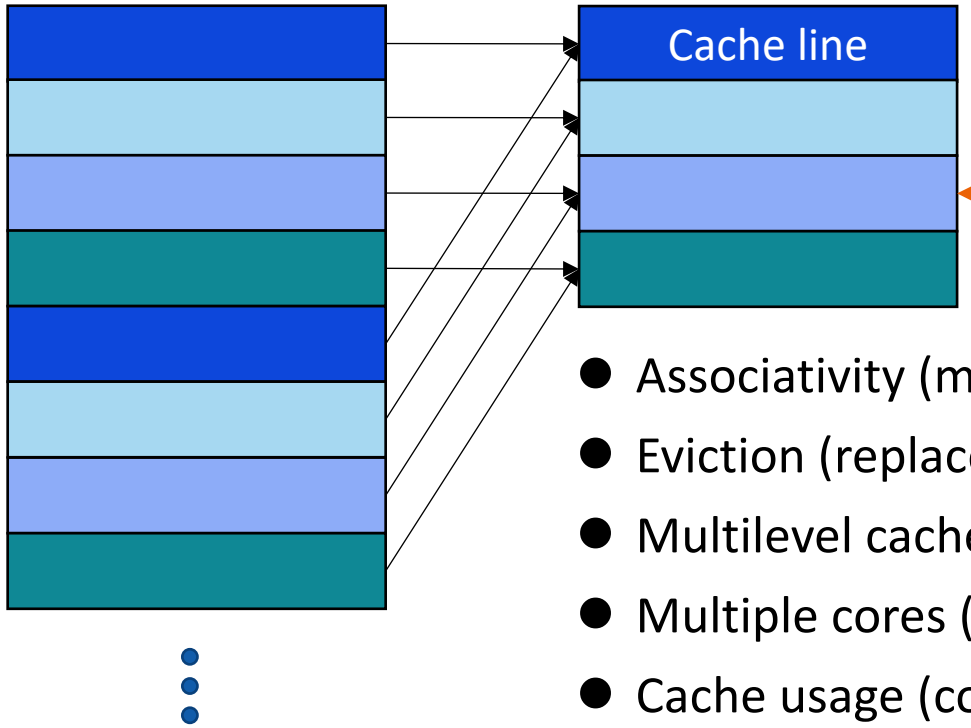
Memory Hierarchy



Memory Hierarchy

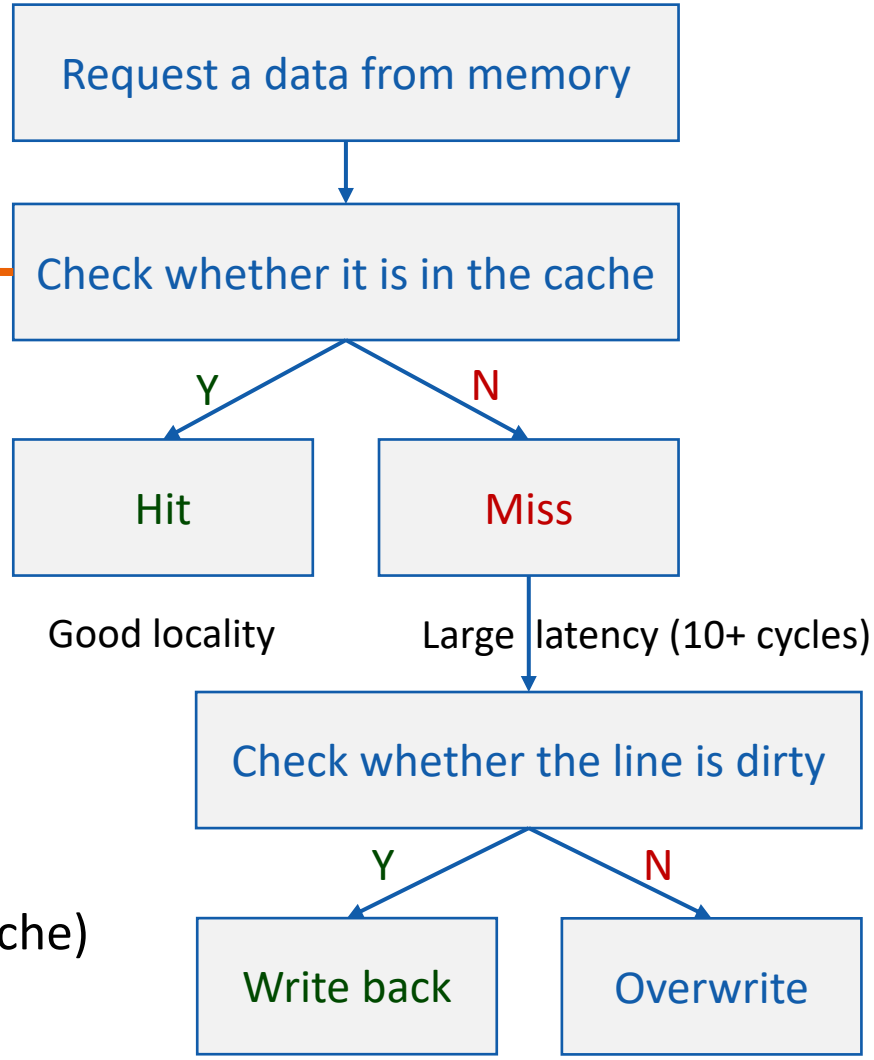


A Simplified Cache Mechanism



An overly simplified example

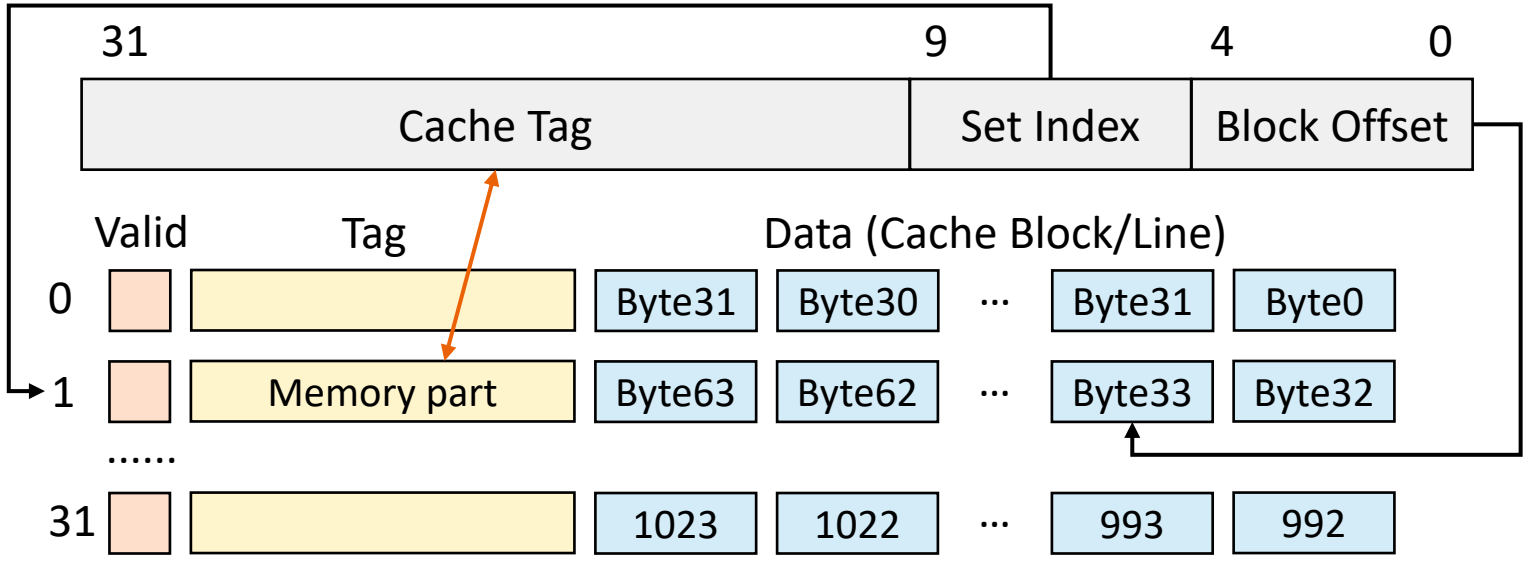
- Associativity (mapping)
- Eviction (replacement) policy
- Multilevel caches (L1, L2, L3)
- Multiple cores (false sharing)
- Cache usage (code, data)
- Size of cache (performance vs price)
- Size of cache line (64B for Intel L1 cache)
- Byte or word addressing



Cache and Locality

- Program usually access a relatively small portion of the memory
  - Temporal Locality: If an item is referenced, it might be referenced again soon
  - Spatial Locality: If an item is referenced, "nearby" items tend to be referenced
- Misses at start up: cold start
- Hit time: 1 - 4 cycles ←→ Miss penalty: 8 - 32 cycles (extra work)

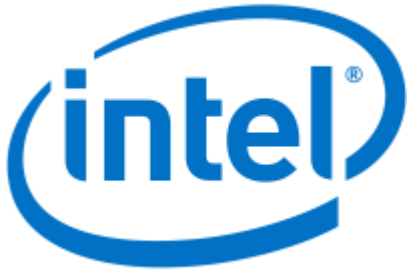
Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011



An example:

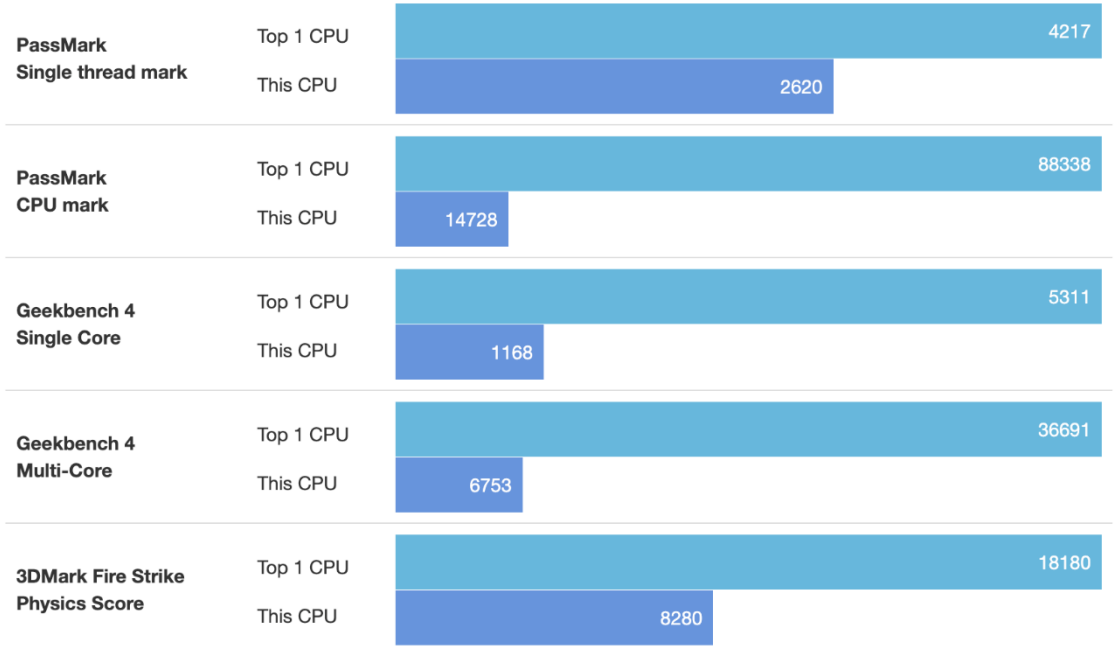
- 1KB direct-mapped cache
- 32 bits memory (Byte) address
- Block offset: 5 bits ➔ size: 32
- Set index: 5 bits ➔ #Lines: 32
- Cache line size = 32 Bytes

A Sample CPU Specific



2019 Production

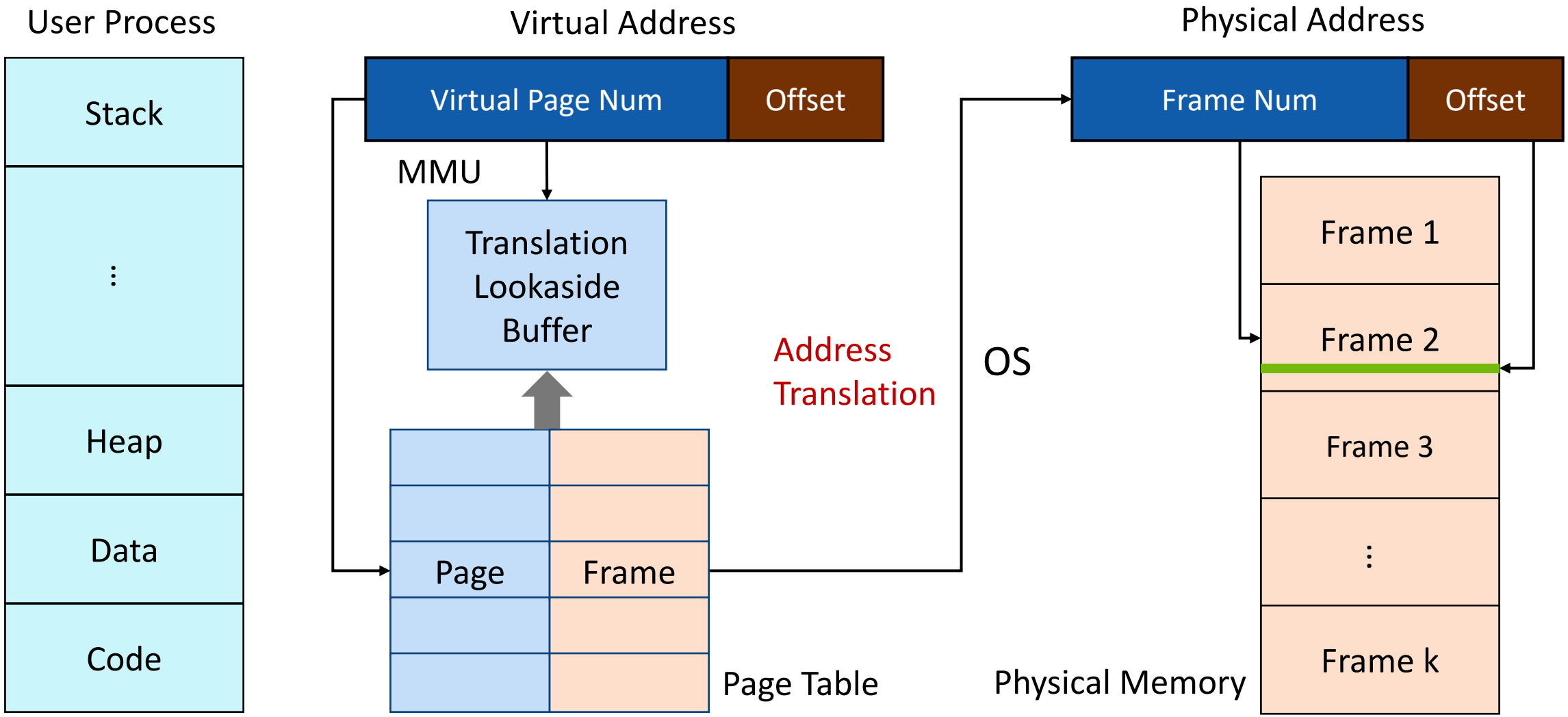
Benchmarks https://askgeek.io/en/cpus/Intel/Core-i9-9980HK



Intel® Core i9-9980HK

- Total num of cores = 8, num of threads = 16
- Maximum CPU clock speed = 5.00 GHz
- Intel Smart Cache:
  - ✓ L1 = 512 KB (cache line: 64 Bytes)
  - ✓ L2 = 2 MB
  - ✓ L3 = 16 MB
- Maximum memory size = 128 GB
- Power consumption (TDP) = 45 Watt
- Maximum operating temperature = 100 °C
- Manufacturing process technology = 14 nm

Memory Address Spaces



Communication, Revisited

Communication ➔ Data Movement

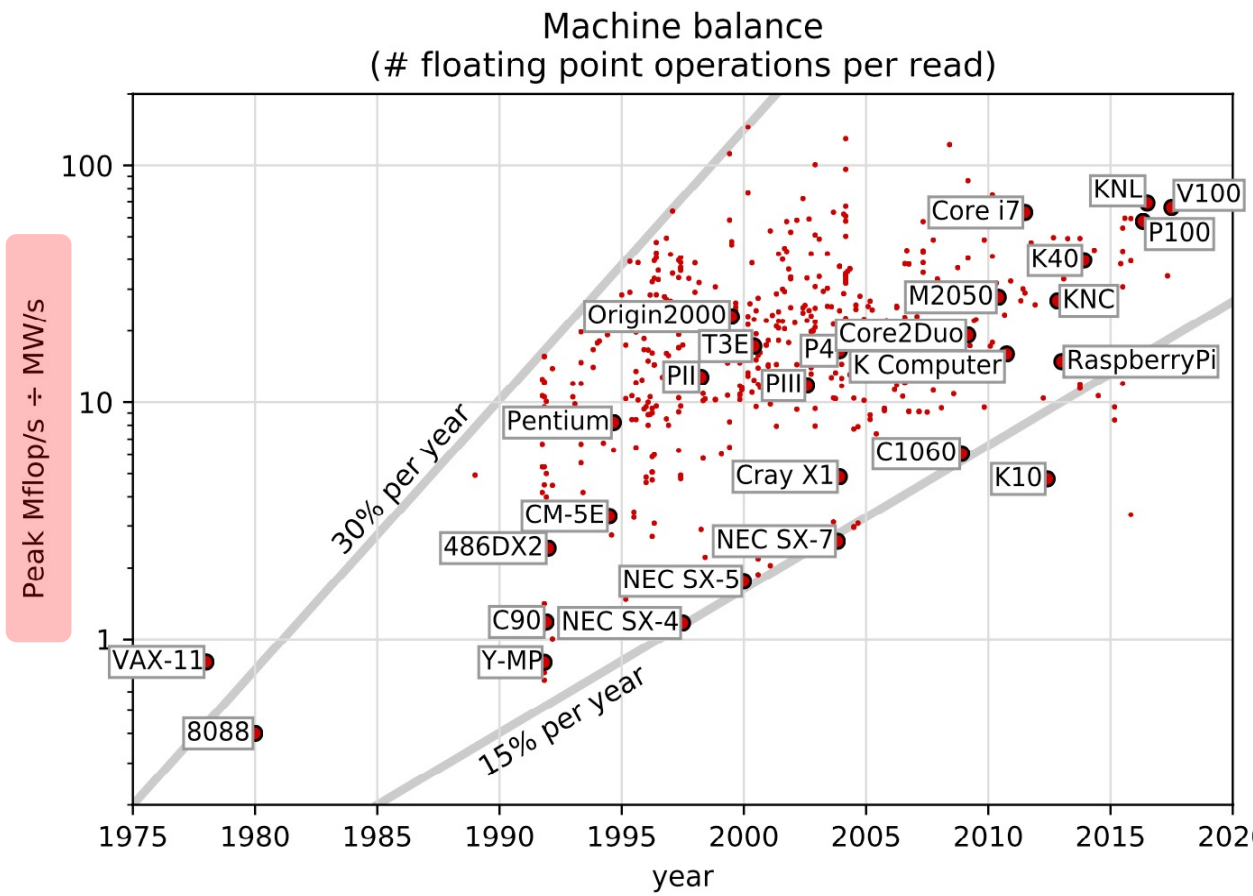
- Physics: Longer the distance, higher the cost
- Technology: Increase speed with different methods
- Economy: Manufacturing cost is also important
- Combined: Computing ability (FLOPS) improves faster than communication ability

	Today's Systems	Predicted Exascale Systems*	Factor Improvement
System Peak	10^{16} flops/s	10^{18} flops/s	100
Node Memory Bandwidth	10^{2} GB/s	10^{3} GB/s	10
Interconnect Bandwidth	10^{1} GB/s	10^{2} GB/s	10
Memory Latency	10^{-7} s	$5 \cdot 10^{-8}$ s	2
Interconnect Latency	10^{-6} s	$5 \cdot 10^{-7}$ s	2

*Sources: from P. Beckman (ANL), J. Shalf (LBL), and D. Unat (LBL)

Annual improvements			
Time/flop		Bandwidth	Latency
59%	Network	26%	15%
	DRAM	23%	5%

Machine Balance Over Time



The gap between the computing power and the memory bandwidth keeps increasing in the last 45 years

- Computing power is (relatively) cheap
- Data movement is slow
- Data movement will be slower
- Take-home message:

Make less data movement!

Source: A. Abdelfattah et al. “A survey of numerical linear algebra methods utilizing mixed-precision arithmetic”. The International Journal of High Performance Computing Applications. 2021;35(4):344-369.

Measuring Performance

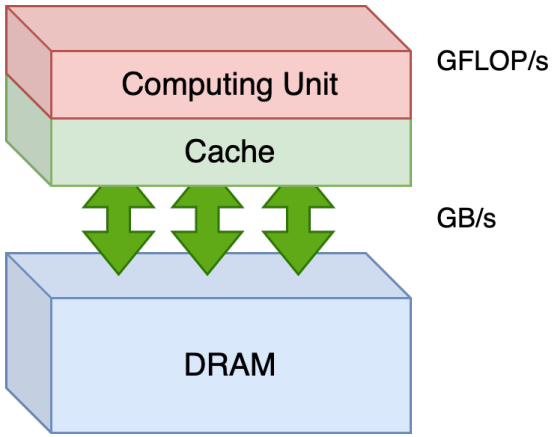
How much time SHOULD a program cost? Is my code fast ENOUGH? Will my boss be happy to see this code?

$$\text{WallTime} = \max\left(\frac{\#\text{FP GFLOP}}{\text{Peak GFLOP/s}}, \frac{\text{DataMove GB}}{\text{Bandwidth GB/s}}\right)$$

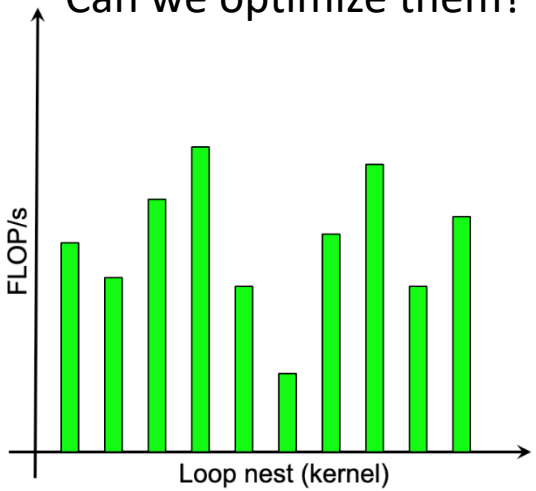
$$\frac{\#\text{FP}}{\text{WallTime}} = \min\left(\text{Peak GFLOP/s}, \frac{\#\text{FP FLOP}}{\text{DataMove Byte}} \times \text{Bandwidth GB/s}\right) \rightarrow \text{Arithmetic Intensity (AI or CI)}$$

Roofline Model

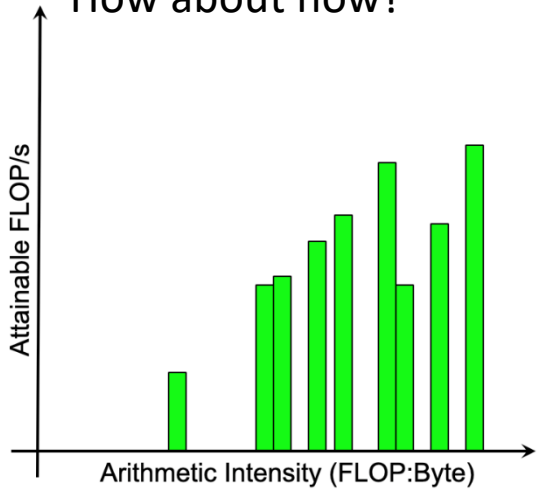
A Simpler Architecture



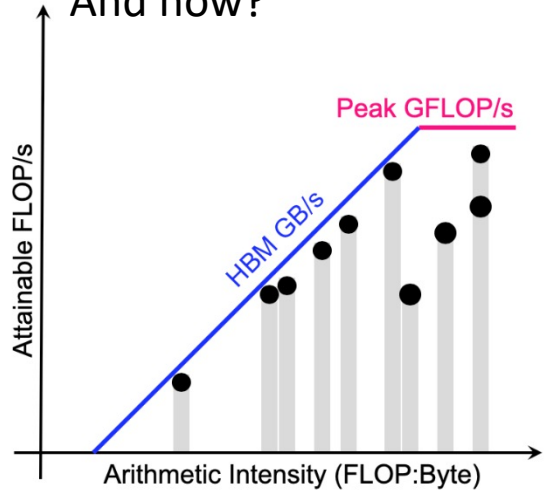
Can we optimize them?



How about now?



And now?

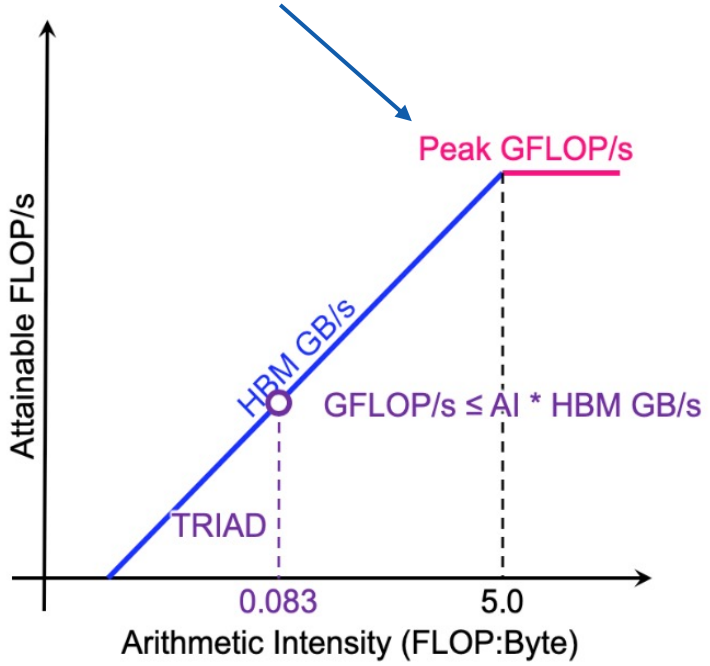


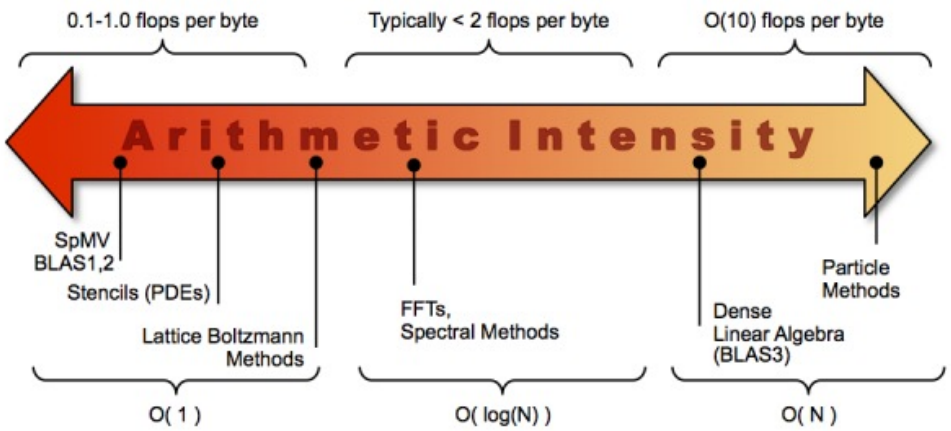
Source: Introduction to the Roofline Model, by Samuel Williams, Lawrence Berkeley National Lab

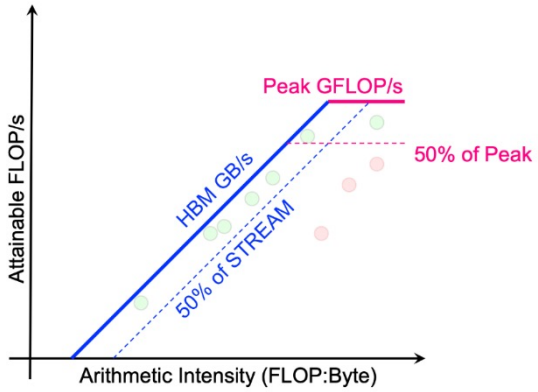
CPU-Bound vs Memory-Bound

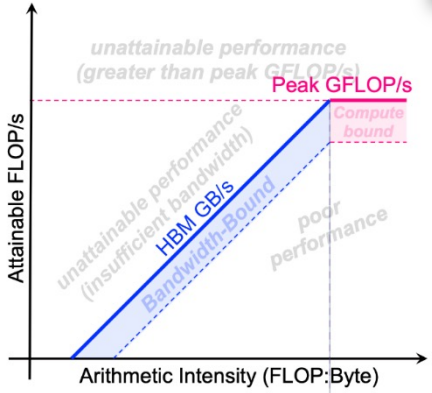
Roofline Model

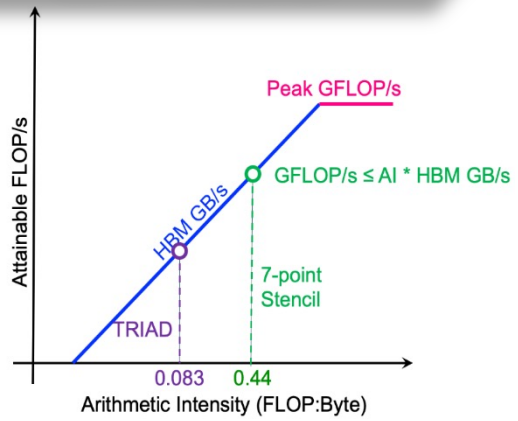
Machine balance is typically 5—10 FLOP/Byte











Why is my code slow?

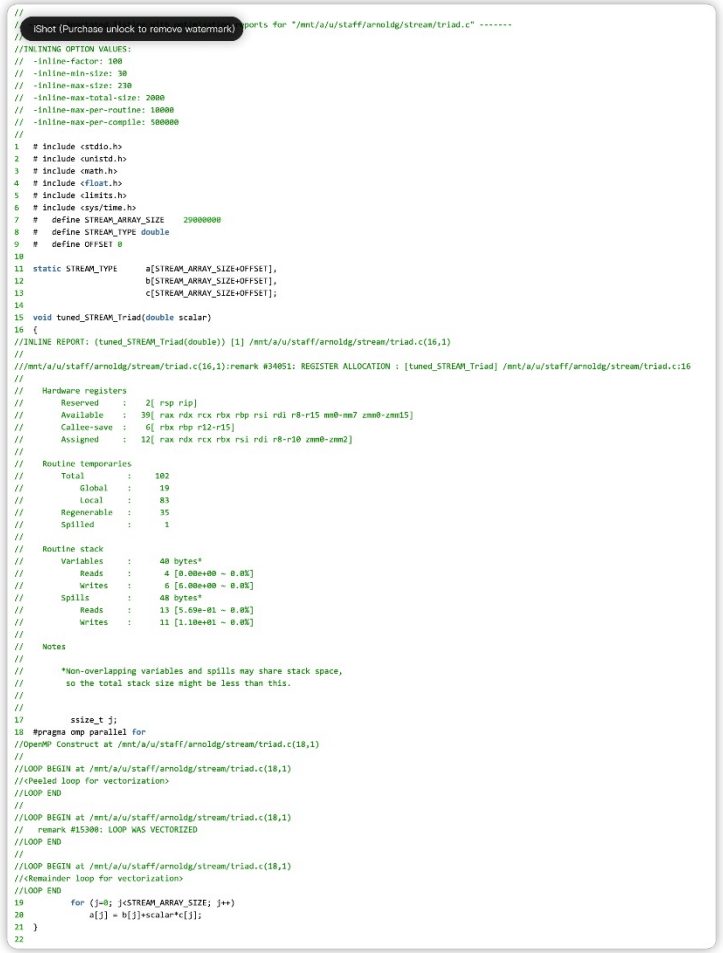
- Does it go anywhere near the roofline?
- Is it CPU or Memory or Comm bound?

Source: Samuel Williams, LBNL
https://crd.lbl.gov/divisions/amcr/computer-science-amcr/par/research/roofline/introduction/

a[i] = b[i] + scalar * c[i]: 2 FLOP, 2 Read, 1 Write, CI = 2/(3*8) = 0.083

7-point stencil: 7 Flop, 1 Read (cache-hit), 1 Write, CI= 7/(2*8) = 0.438

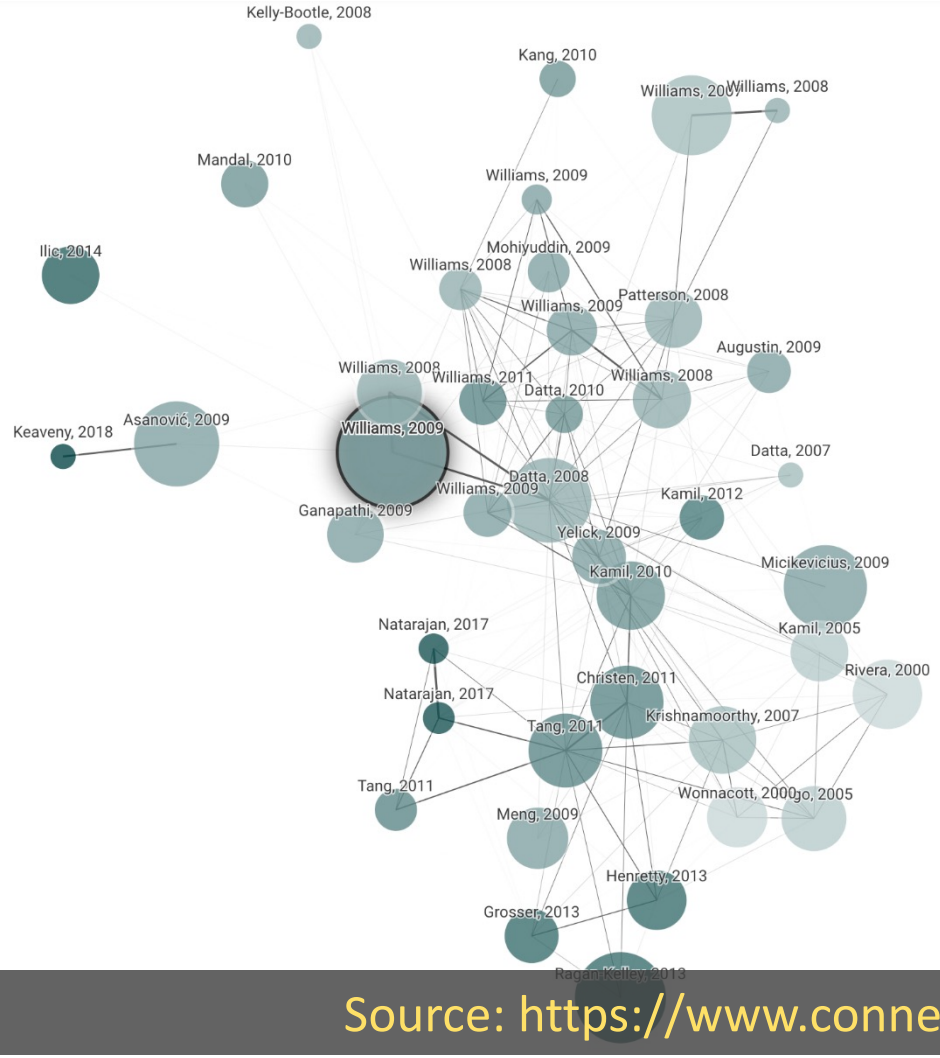
A Sample TRIAD Code



```
18  #pragma omp parallel for
//OpenMP Construct at /mnt/a/u/staff/arnoldg/stream/triad.c(18,1)
//remark #16200: OpenMP DEFINED LOOP WAS PARALLELIZED
//
//LOOP BEGIN at /mnt/a/u/staff/arnoldg/stream/triad.c(18,1)
//<Peeled loop for vectorization>
//LOOP END
//
//LOOP BEGIN at /mnt/a/u/staff/arnoldg/stream/triad.c(18,1)
//   remark #15300: LOOP WAS VECTORIZED
//LOOP END
//
//LOOP BEGIN at /mnt/a/u/staff/arnoldg/stream/triad.c(18,1)
//<Remainder loop for vectorization>
//LOOP END
19          for (j=0; j<STREAM_ARRAY_SIZE; j++)
20              a[j] = b[j]+scalar*c[j];
21  }
```

Source: https://wiki.ncsa.illinois.edu/display/AAG/Copy+of+triad.c+and+vector+optimization

Roofline Model Papers



Roofline: an insightful visual performance model for multicore architectures

Samuel Williams, Andrew Waterman, D. Patterson

2009, CACM

1786 Citations

Save

Open in:

The Roofline model offers insight on how to improve the performance of software and hardware.

Source: https://www.connectedpapers.com/

Iterative Refinement Techniques

Iteratively improve accuracy of linear solvers

Mixed-Precision Algorithms

- Research on mixed-precision algorithms nowadays:

  Try to accelerate the applications by using lower-precision formats while maintaining high accuracy results

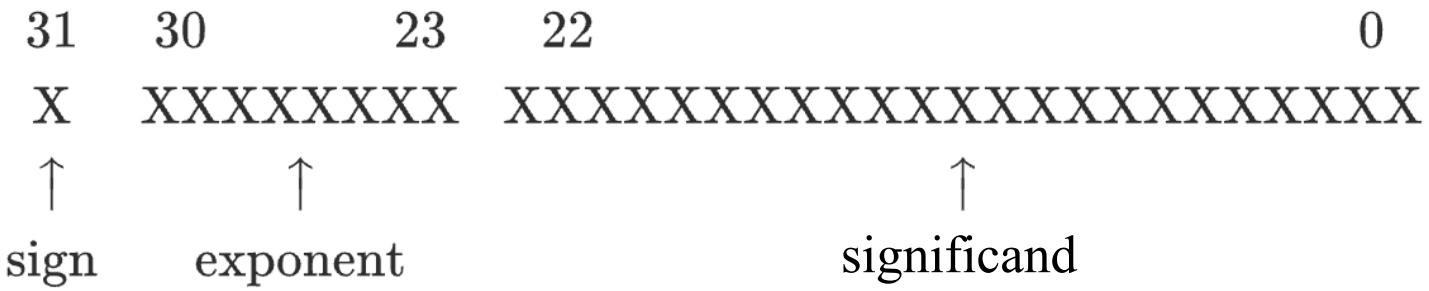
- Main goals:
  - Efficient memory access (also better cache performance)
  - Less data movement (also less energy consumption)
  - Shift gears: Optimize number of flops in good old days

    ➔➔➔ Fast Computation + Fast Communication

- Your generation will face “slower” communication (relative to computation)!

Floating-Point Number Storage Format

IEEE single-precision floating-point number



Normal

$$\text{Number} = (-1)^s 2^{e-127} \underbrace{1 \cdot f}_{\text{With leading 1, normalized}},$$

where $s = 0$ for positive numbers, 1 for negative numbers,

$$\text{e} = \text{exponent (between 0 and 255), and}$$

$$f = \text{mantissa.}$$

- Fixed-size FP format
- Representation range
- Significand / Mantissa
- Unit round-off / precision
- Machine epsilon
- Little-endian or big-endian
- IEEE 754-1985 (fp32, fp64)
- IEEE 754-2008 (fp16, fp128)

Ref: Higham, Nicholas J. and Mary, Theo. ”Mixed Precision Algorithms in Numerical Linear Algebra”, Acta Numerica, to appear

IEEE754 Single-Precision Numbers

Exponent	Decimal Value	Shift-127 Value
1111,1111	255	128
0111,1111	127	0
0000,0001	1	-126
0000,0000	0	???

Reserved INF/NAN

$1.0 * 2^{0111,1111}$

Reserved for 0.0 (all bits are 0)

Overflow $2^{128} \approx 3.40E38$

Min Normal $2^{(-126)} \approx 1.18E-38$

Exponent	Mantissa	Purpose
0	zero	0.0
0	**nonzero**	**subnormal**
1-254	anything	normal
255	zero	INF
255	nonzero	NaN

$1.f * 2^{0000,0000}$?

No! Need special treatment …

Max number $1.1……1 * 2^{1111,1110}$

Q: What do we do if a number falls out of the range?

IEEE754 2008 Version

Table 1. Parameters for floating-point arithmetics: number of bits in significand (mantissa), including the implicit most significant bit; number of bits in exponent; and, to three significant figures, unit roundoff u, smallest positive (subnormal) number $x_{\min}^{(s)}$, smallest normalized positive number $x_{\min}$ and largest finite number $x_{\max}$. In Intel's bfloat16 specification, subnormal numbers are not supported [10], so any number less than $x_{\min}$ in magnitude is flushed to zero; the value shown holds if subnormal numbers are supported.

	signif.	exp.	u	$x_{\min}^{(s)}$	$x_{\min}$	$x_{\max}$
bfloat16	8	8	3.91×10^{-3}	9.18×10^{-41}	1.18×10^{-38}	3.39×10^{38}
fp16	11	5	4.88×10^{-4}	5.96×10^{-8}	6.10×10^{-5}	6.55×10^{4}
fp32	24	8	5.96×10^{-8}	1.40×10^{-45}	1.18×10^{-38}	3.40×10^{38}
fp64	53	11	1.11×10^{-16}	4.94×10^{-324}	2.22×10^{-308}	1.80×10^{308}
fp128	113	15	9.63×10^{-35}	6.48×10^{-4966}	3.36×10^{-4932}	1.19×10^{4932}

Source: Dongarra J, Grigori L, Higham NJ. "Numerical algorithms for high-performance computational science". Phil. Trans. R. Soc. A 378: 20190066, 2020

Subnormal Floating-Point Numbers

$$(-1)^s \times 2^{0-127+1} \times 0.f$$

- Subnormal numbers have all zero bits for the exponent
- Numbers that are smaller than the smallest normal positive number (consider positive only)
- Subnormal numbers are useful in filling gaps of floating point scale near zero
- Shift by -126 instead of -127! Why +1? Remember the exponent of the smallest normal number?
- Cannot add leading ONE any more: less bit ➔ less accurate
- The largest possible subnormal number is $0.111\ldots11 \times 2^{-126} < 1.0\ldots0 \times 2^{-126}$
- The smallest possible subnormal number is $0.000\ldots01 \times 2^{-126} \approx 1.40E\text{-}45$
- Floating-point operations in the subnormal region can be very very slow
- Flush subnormals (denormals) to zero using compiler arguments

Simple Examples of FP Arithmetic

Example using decimal

$A = 9.999 \times 10^{1}$, $B = 1.610 \times 10^{-1}$, $A+B$ =?

Step 1. Align the smaller exponent with the larger one.

$B = 0.0161 \times 10^{1} = 0.016 \times 10^{1}$ (round off)

Step 2. Add significands

$9.999 + 0.016 = 10.015$, so $A+B = 10.015 \times 10^{1}$

Step 3. Normalize

$A+B = 1.0015 \times 10^{2}$

Step 4. Round off

$A+B = 1.002 \times 10^{2}$

Unit round-off

Example using decimal

$A = 1.110 \times 10^{10}$, $B = 9.200 \times 10^{-5}$ $A \times B$ =?

Step 1. Exponent of $A \times B = 10 + (-5) = 5$

Step 2. Multiply significands

$1.110 \times 9.200 = 10.212000$

Step 3. Normalize the product

$10.212 \times 10^{5} = 1.0212 \times 10^{6}$

Step 4. Round off

$A \times B = 1.021 \times 10^{6}$

Step 5. Decide the sign of $A \times B$ (+ x + = +)

So, $A \times B = + 1.021 \times 10^{6}$

Source: https://homepage.divms.uiowa.edu/~ghosh/

Working With Mixed Precisions

N. Higham, 2002

- Compute matrix-vector multiplications in finite precision (forward error):

$$\mathrm{fl}(Ax) = Ax + \delta, \quad |\delta| \leq \gamma_n |A||x|$$

- Compute residual in finite precision (forward error):

$$\mathrm{fl}(b - Ax) = b - Ax + \delta, \quad |\delta| \leq \gamma_{n+1} \left(|b| + |A||x| \right)$$

- Pessimistic bounds! Improvement using blocking, probabilistic error bounds, ...

- How about working on two-precision computation?

First compute residual in precision $\overline{u}$ and then store it in precision u

$$\mathrm{fl}(b - Ax) = b - Ax + \delta, \quad |\delta| \leq u|b - Ax| + (1 + u)\overline{\gamma}_{n+1} \left(|b| + |A||x| \right)$$

Accuracy of Direct Solvers, Revisited

N. Higham, 2002

- Floating-point computation (IEEE compatible)

$$\mathrm{fl}(x \ \mathrm{op} \ y) = (x \ \mathrm{op} \ y)(1+\delta), \quad |\delta| \le u$$

- Forward error of matrix-multiplications (depends on num of inner multiplications)

$$\text{Compute } C = AB, A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times k} \Longrightarrow |\hat{C} - C| \le \gamma_n |A||B| \qquad \gamma_n := \frac{nu}{1-nu}$$

- Backward perturbation analysis of the LU decomposition

$$(A+\delta A)\hat{x} = b, \quad |\delta A| \le \gamma_{3n} |\hat{L}||\hat{U}|$$

- Accuracy of direct solution methods

$$\frac{\|x-\hat{x}\|_\infty}{\|x\|_\infty} \le \gamma_{3n} \frac{\||\hat{L}||\hat{U}|\|_\infty}{\|A\|_\infty} \kappa_\infty(A)$$

Iterative Refinement Methods

- Sometimes, the condition number of a linear system could be very large
- Direct solvers might not be able to give a solution as accurate as we wanted (or thought)
- This has been noticed long ago (Wilkinson 1948), but using high-precision everywhere is costly
- We can improve precision iteratively using the iterative refinement

Algorithm 9: General iterative refinement algorithm

```
%% Given a nonsingular matrix A ∈ ℝ^{n×n}, b ∈ ℝ^n, and an initial guess x_0 ∈ ℝ^n;
%% Set three precisions u_r ⩽ u ⩽ u_ℓ for residual, work, and solve, respectively;
for i = 0 : MaxIter or converged
    Compute r_i ← b − A x_i in precision u_r;
    Solve the error equation A e_i = r_i in precision u_ℓ;
    Update x_{i+1} ← x_i + e_i in precision u;
end
```

Higham, N. and Mary, T. “ Mixed Precision Algorithms in Numerical Linear Algebra”, Acta Numerica

Accuracy of IR Methods

Error Estimate (Carson and Higham 2018)

The limiting accuracy depends on the **residual and work precisions**, but does not depend on the **solve precision**, the initial guess, or how the system (Line 5) is solved.

$$\frac{\|\hat{x} - x\|_\infty}{\|x\|_\infty} \lesssim u + 4\,p\,\mathrm{cond}(A, x)u_r$$

Need to be sufficiently small

Convergence factor

$$\phi_i := \xi_i + 2\min\left(\mathrm{cond}(A), \kappa_\infty(A)\mu_i\right)u_\ell$$

$$\xi_i := \frac{\|\hat{e}_i - e_i\|_\infty}{\|e_i\|_\infty}$$

$$\mu_i := \frac{\|A(\hat{x}_i - x_i)\|_\infty}{\|A\|_\infty \|\hat{x}_i - x_i\|_\infty} \leqslant 1$$

Maximum nnz in any row of the equation

Condition number

$$\mathrm{cond}(A) := \||A^{-1}||A|\|_\infty$$

$$\mathrm{cond}(A, x) := \frac{\||A^{-1}||A||x|\|_\infty}{\|x\|_\infty}$$

$$\mathrm{cond}(A, x) \leqslant \mathrm{cond}(A) \leqslant \kappa_\infty(A)$$

LU-IR

Algorithm 10: LU-IR method

```
%% Given a nonsingular matrix A ∈ R^{n×n} and b ∈ R^n;
%% Set three precisions u_r ⩽ u ⩽ u_ℓ for residual, work, and solve, respectively;
Compute the factorization A = LU in precision u_ℓ;
Solve LUx_0 = b by substitution in precision u_ℓ;
for i = 0 : MaxIter or converged
    Compute r_i ← b − Ax_i in precision u_r;
    Solve LUe_i = r_i by substitution in precision u_ℓ;
    Update x_{i+1} ← x_i + e_i, in precision u;
end
```

- LU factorization only need to be computed once for each coefficient matrix
- The Carson-Higham theorem can be applied to this method
- Less stable but faster factorization methods can be used to improve efficiency (e.g. in SuperLU and PARDISO) [Li and Demmel 1998; Arioli, Duff, Gratton, and Pralet 2007]

GMRES-IR

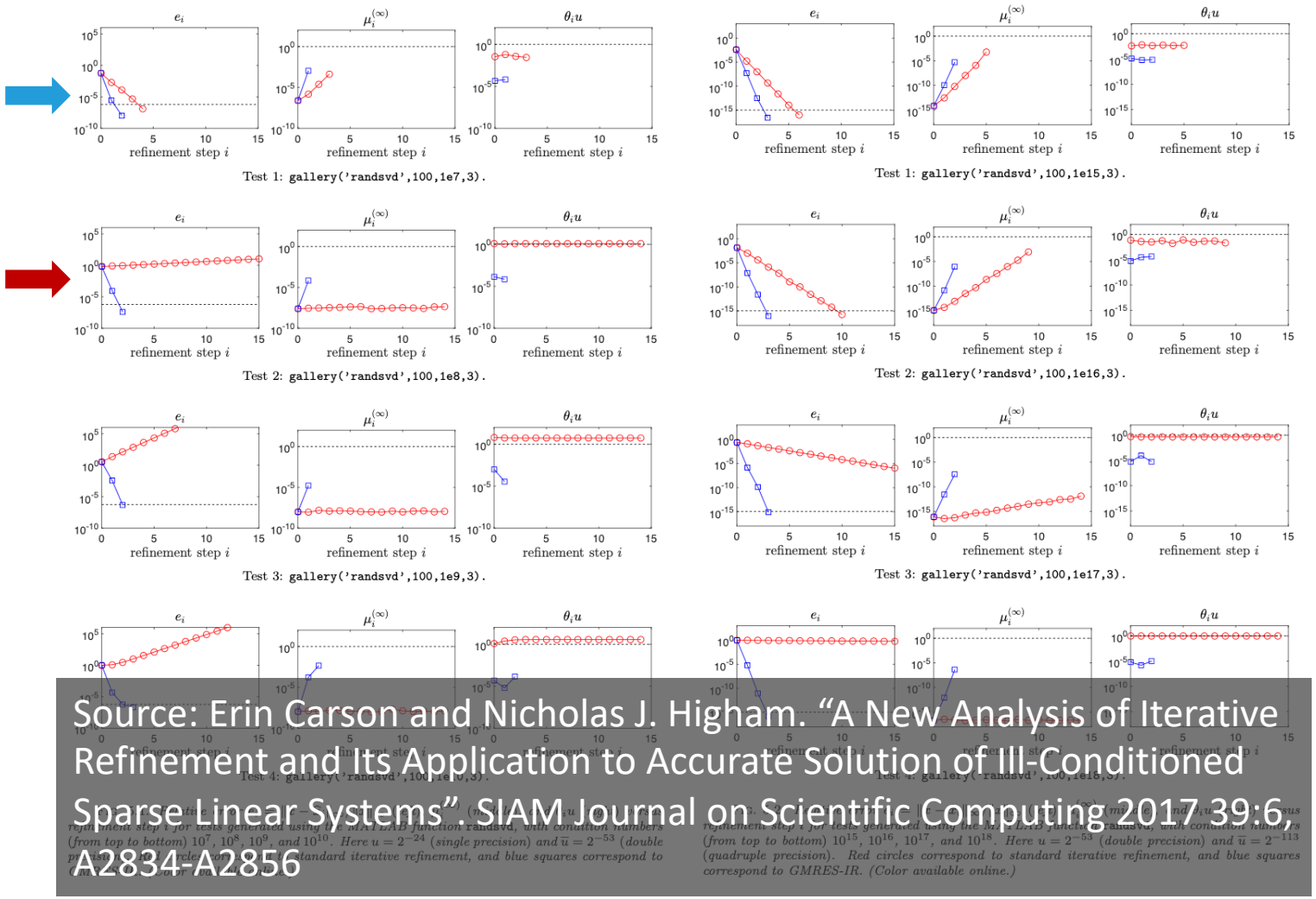
Algorithm 11: Preconditioned GMRES-IR method

```
%% Given a nonsingular matrix A ∈ R^{n×n} and b ∈ R^n;
%% Set five precisions u_r, u_g, u_p, u, u_ℓ for different purposes;
Compute the factorization A = LU in precision u_ℓ;
Solve LUx_0 = b by substitution in precision u_ℓ;
for i = 0 : MaxIter or converged
    Compute r_i ← b − Ax_i in precision u_r;
    Solve Ae_i = r_i by PGMRES in precision u_g;
    In PGMRES, preconditioner U^{-1}L^{-1} applied in precision u_p;
    Update x_{i+1} ← x_i + e_i in precision u;
end
```

If $\kappa(A)^2 u_\ell^2 (u_g + \kappa(A) u_p)$ is small, GMRES-IR will converge till

$$\frac{\|\hat{x} - x\|_\infty}{\|x\|_\infty} \lesssim u + 4\,p\,\mathrm{cond}(A, x) u_r$$

Accuracy of IR Methods

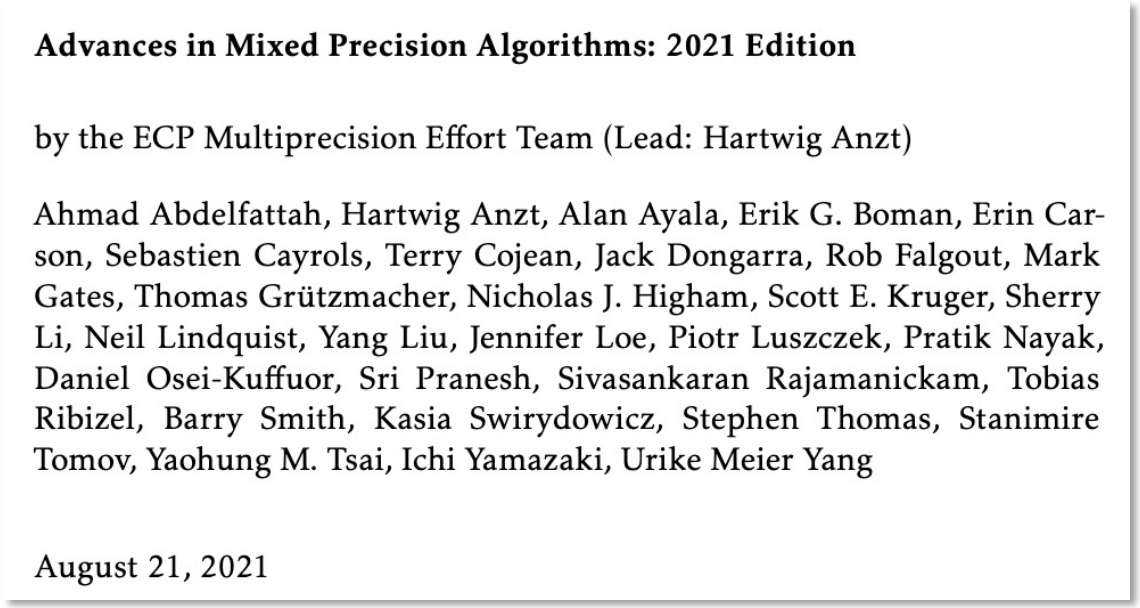


Source: Erin Carson and Nicholas J. Higham. "A New Analysis of Iterative Refinement and Its Application to Accurate Solution of Ill-Conditioned Sparse Linear Systems". SIAM Journal on Scientific Computing 2017 39:6, A2834-A2856

- GMRES-IR shows advantages compared with LU-IR in practice
- GMRES-IR is used in HPL-AI
- One can also use GMRES for the original equation instead of the error equation
- One can use low precision for GMRES iterations
- Carson and Higham, SISC, 2017; Haidar et al. 2020; …

Different Versions of IR

- **Traditional IR methods** for ancient hardware (some computers can compute residual using higher precision than working precision without extra work); since 1970s these methods die due to such hardware does not exist any more.
- **Fixed-precision IR methods** (improve stability)
- **IR methods with low-precision solvers**: emphasize on efficiency and energy consumption

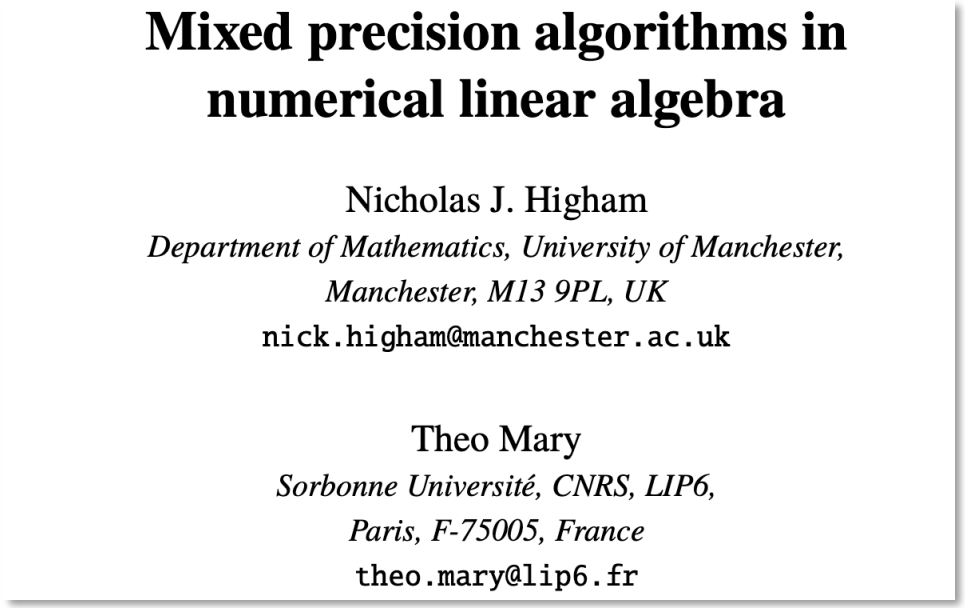


Advances in Mixed Precision Algorithms: 2021 Edition

by the ECP Multiprecision Effort Team (Lead: Hartwig Anzt)

Ahmad Abdelfattah, Hartwig Anzt, Alan Ayala, Erik G. Boman, Erin Carson, Sebastien Cayrols, Terry Cojean, Jack Dongarra, Rob Falgout, Mark Gates, Thomas Grützmacher, Nicholas J. Higham, Scott E. Kruger, Sherry Li, Neil Lindquist, Yang Liu, Jennifer Loe, Piotr Luszczek, Pratik Nayak, Daniel Osei-Kuffuor, Sri Pranesh, Sivasankaran Rajamanickam, Tobias Ribizel, Barry Smith, Kasia Swirydowicz, Stephen Thomas, Stanimire Tomov, Yaohung M. Tsai, Ichi Yamazaki, Urike Meier Yang

August 21, 2021



Mixed precision algorithms in numerical linear algebra

Nicholas J. Higham
Department of Mathematics, University of Manchester, Manchester, M13 9PL, UK
nick.higham@manchester.ac.uk

Theo Mary
Sorbonne Université, CNRS, LIP6, Paris, F-75005, France
theo.mary@lip6.fr

Mixed-Precision Algorithms

A few selected topics on mixed-precision numerical linear algebra

/03

Blocked Algorithms

- Simple summation (forward error analysis)

$$\gamma_n = nu + O(u^2)$$

$$s = x^T y = \sum_{i=1}^{n} x_i\, y_i \quad \Longrightarrow \quad |s - \hat{s}| \le n\,u\,|x|^T\,|y|$$

Algorithm 12: Blocked summation method

```
1  %% Given two vectos x, y ∈ R^n, n = k × b, and s = 0.0;
2  for i = 1 : k
3      iend ← i × b;
4      istart ← iend − b + 1;
5      s_i ← x(istart:iend)^T y(istart:iend);
6      s ← s + s_i;
7  end
```

- Blocked summation improves accuracy $\quad |s - \hat{s}| \le \left(\frac{n}{k} + k\right) u\,|x|^T|y|$

Further Improvement on Summation

- So the basic idea is to reduce the number of additions in the summation
- A more advanced algorithm (FABsum) can give error bound independent of n to first order
- Using hardware: Fused Multiply-Add (FMA)

No FMA: $\mathrm{fl}(x + yz) = \big(x + yz(1+\delta_1)\big)(1+\delta_2), \quad |\delta_1|, |\delta_2| \le u$

With FMA: $\mathrm{fl}(x + yz) = (x + yz)(1+\delta), \quad |\delta| \le u$

Not only faster in computation

But also more accurate (by a factor of 2)!

Blanchard, Pierre, Nicholas John Higham and Théo Mary. “A Class of Fast and Accurate Summation Algorithms.” SIAM J. Sci. Comput. 42 (2020): A1541-A1557.

Considerations for Sparse Problems

- Much harder to predict savings of MP iterative methods for sparse matrices
- LU factorization of a sparse matrix
  - Reordering and analysis phase of the algorithm does not involve floating-point arithmetic and so does not benefit from reducing the precision
  - LU can generate cascading fill-ins (small multipliers combine to produce subnormal numbers)
- Even if matrix entries can be scaled from double to normalized single precision numbers, subnormal numbers may still be generated during the factorization
- Floating-point operations on subnormal numbers can be very slow ➔ This can cause significant performance loss

Ref: Zounon M, Higham NJ, Lucas C, Tisseur F. "Performance impact of precision reduction in sparse linear systems solvers". Peer J Computer Science 2022, 8:e778

Mixed-Precision GMRES

Algorithm 14: General Two-Precision GMRES

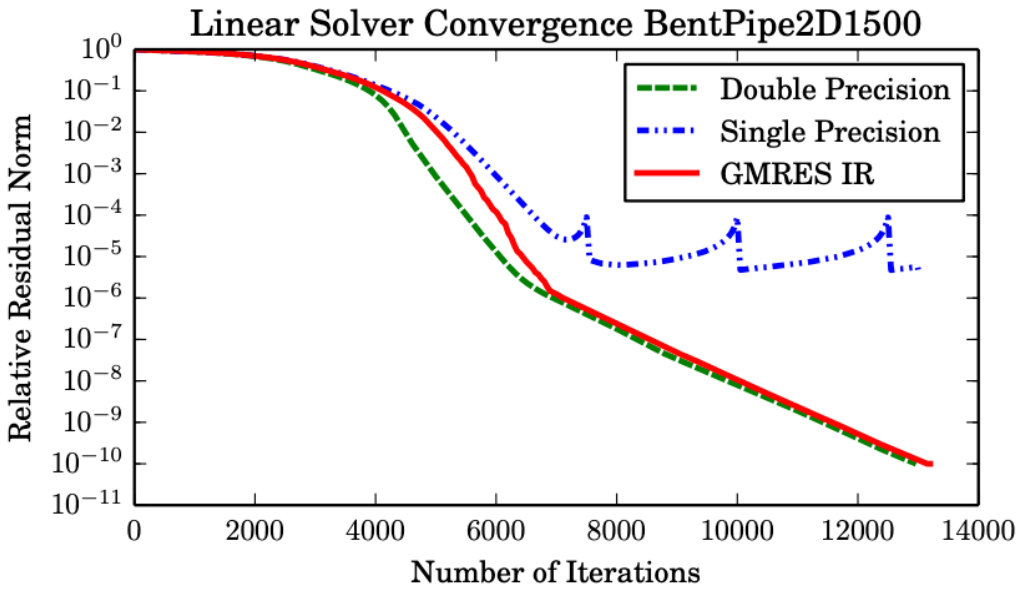
```
1 %% Given a nonsingular matrix A ∈ R^{n×n}, b ∈ R^n, and an initial guess x_0 ∈ R^n;
2 %% Set restart number m and tolerance τ for GMRES
3 %% Set two precisions u ⩽ u_ℓ for different purposes;
4 for i = 0 : MaxIter or converged
5     Compute r_i ← b − Ax_i in precision u;
6     Solve Ae_i = r_i by GMRES(m) to tolerance τ in precision u_ℓ;
7     Update x_{i+1} ← x_i + e_i in precision u;
8 end
```

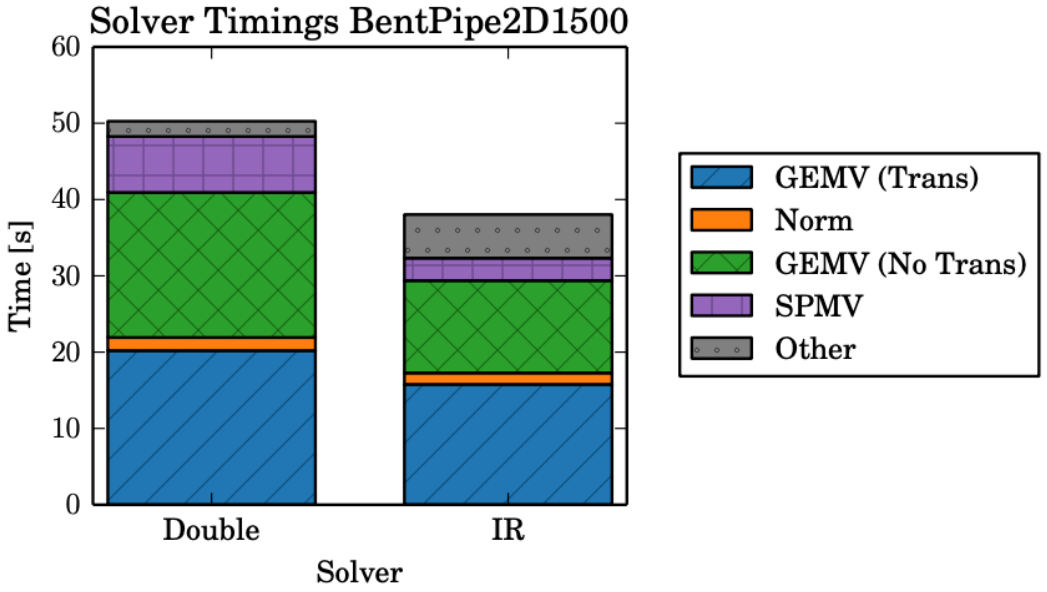
Inner-outer iterations

- If using same precision, then this is the restarted GMRES method (see Lecture 3)
- FGMRES as outer iteration and GMRES as inner solver (Buttari et al. 2008)
- FGMRES as outer iteration and low precision LU as inner solver is stable (Arioli and Duff 2009)

A. Abdelfattah et al., “Advances in Mixed Precision Algorithms: 2021 Edition”. doi:10.2172/1814677.

Performance of MP-GMRES

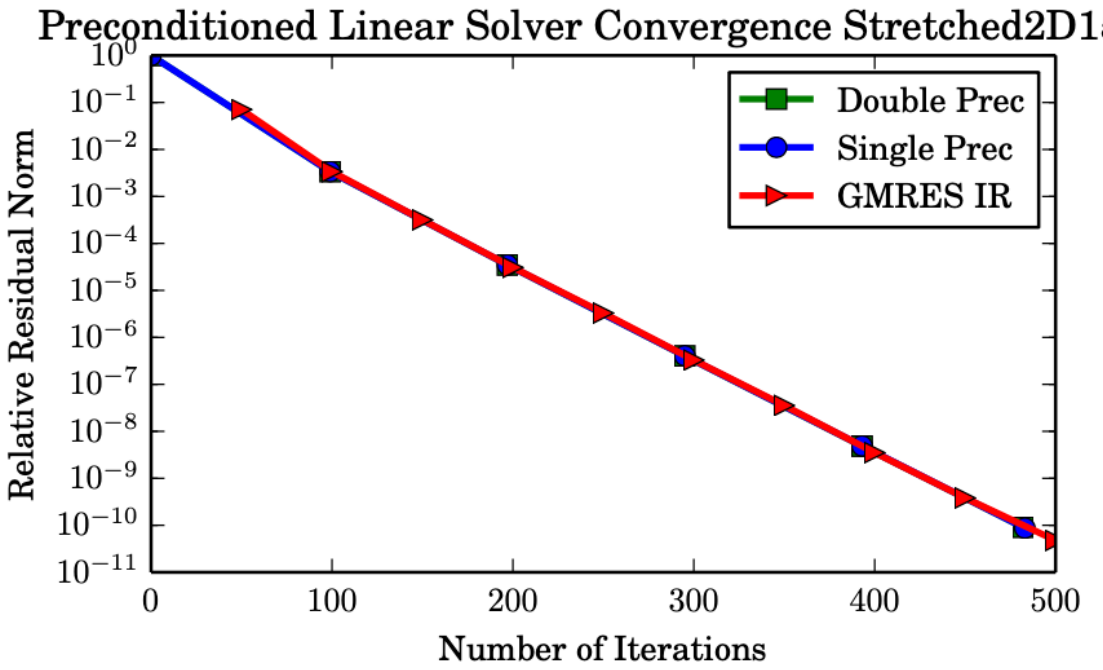




- MP-GMRES has almost same convergence behavior as DP-GMRES, much better than SP-GMRES
- GEMV can be speed-up by 1.28 and 1.57 times
- SpMV can be speed-up by 2.48 times!!!
- But … such information is not enough! How about Preconditioned GMRES?

Source: A. Abdelfattah et al., “Advances in Mixed Precision Algorithms: 2021 Edition”. doi:10.2172/1814677.

Performance of Preconditioned MP-GMRES



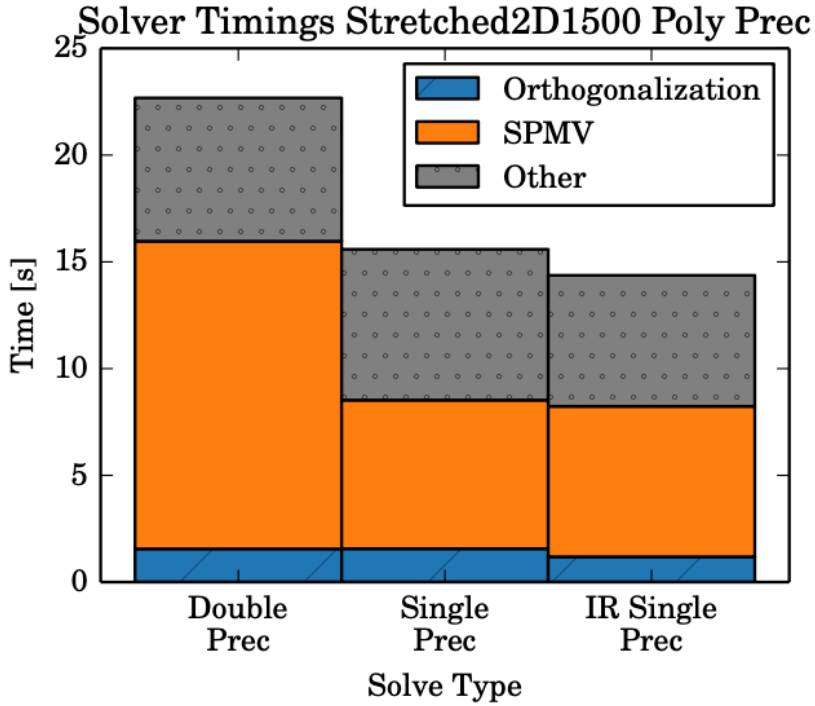


Figure 7: Convergence (left) and solve times (right) for a Laplacian on a stretched grid with a degree 40 polynomial preconditioner. We tested i) fp64 GMRES with fp64 polynomial [Double Prec], ii) fp64 GMRES with fp32 polynomial [Single Prec], and iii) GMRES-IR with an fp32 polynomial [IR Single Prec].

Source: Abdelfattah, A. et al, "Advances in Mixed Precision Algorithms: 2021 Edition". doi:10.2172/1814677.

Robustness of Scaling Algorithms

Listing 12: Scaling method

```
%% Given a nonsingular matrix A ∈ R^{n×n} and a parameter θ ∈ (0, 1];
R = diag(‖(A(i, :))‖_∞^{-1})
A ← RA %% row equilibration
P = diag(‖(A(:, j))‖_∞^{-1})
A ← AP %% column equilibration
A ← fl(θ x_max β^{-1} A) %% β is the maximum magnitude of |A|
```

Source: Haidar A, et al., Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems. Proc. R. Soc. 2020

Table 3. Number of iterations required by the proposed iterative refinement solver having the `FP32` as lower precision (`dsgesv`) using the `GM`, `IR` and `IRGM` refinement methods. Also, we show the effect of the different scaling techniques in each block-column.

			no scaling			scalar scaling			diagonal scaling				diag + scalar scaling		
name	size	$\kappa_\infty(A)$	IR	IRGM	GM	IR	IRGM	GM	$\kappa_\infty(A^S)$	IR	IRGM	GM	IR	IRGM	GM
crystk03	24 696	2.09×10^2	1	1	2	1	1	2	2.5×10^1	1	1	2	1	1	2
crystm03	24 696	8.24×10^1	1	1	2	1	1	2	7.7×10^0	1	1	2	1	1	2
TEM27623	27 623	4.70×10^3	1	1	1	1	1	2	2.1×10^1	1	1	2	1	1	2
Goodwin054	32 510	4.49×10^6	2	2	3	2	2	3	2.2×10^4	2	2	3	2	2	3
thermal1	17 880	1.60×10^3	1	1	3	1	1	2	8.0×10^1	1	2	3	2	2	2
Zhao1	33 861	4.45×10^2	1	1	3	2	1	2	1.6×10^1	2	2	2	2	2	2
nd6k	18 000	3.82×10^2	1	1	2	1	1	2	1.0×10^2	1	1	2	1	1	2
nd12k	36 000	3.51×10^2	1	1	2	1	1	2	1.0×10^2	1	1	2	1	1	2
epb1	14 734	2.62×10^4	2	2	4	2	2	3	4.4×10^4	2	2	3	2	2	3
appu	14 000	1.02×10^4	2	2	3	2	2	2	5.8×10^3	2	2	3	2	2	2
ns3Da	20 414	3.79×10^3	2	2	3	2	2	2	3.4×10^3	2	2	3	2	2	2
mixtank new	29 957	4.69×10^{11}	2	2	4	2	2	3	2.1×10^5	2	2	4	2	2	3
e40r0100	17 281	2.23×10^8	2	3	4	3	4	4	1.1×10^4	2	2	3	2	2	2
bcsstk25	15 439	1.02×10^{10}	1	1	1	1	1	2	6.2×10^2	1	1	2	1	1	2
bcsstk37	25 503	2.27×10^9	1	1	2	1	1	2	1.5×10^2	1	1	2	1	1	2
raefsky3	21 200	5.18×10^{11}	1	1	3	1	1	2	1.2×10^4	2	2	3	2	2	2
thread	29 736	1.78×10^8	1	1	1	1	1	2	1.3×10^3	1	1	2	1	1	2
sme3Db	29 067	1.05×10^8	15	10	9	15	11	9	3.8×10^7	16	11	12	16	11	9
mult dcop 01	25 187	9.55×10^{11}	3	2	6	3	4	4	1.5×10^8	2	2	5	2	2	3
ramage02	16 830	4.52×10^8	—	60	39	—	58	38	4.5×10^7	11	12	15	37	28	22

Table 4. Number of iterations required by the proposed iterative refinement solver having the `FP16-TC` as lower precision (`dhgesv-TC`) using the `GM`, `IR` and `IRGM` refinement methods. Also, we show the effect of the different scaling techniques in each block-column.

			no scaling			scalar scaling			diagonal scaling				diag + scalar scaling		
name	size	$\kappa_\infty(A)$	IR	IRGM	GM	IR	IRGM	GM	$\kappa_\infty(A^S)$	IR	IRGM	GM	IR	IRGM	GM
crystk03	24 696	2.09×10^2	1	1	2	1	1	2	2.5×10^1	1	1	2	1	1	2
crystm03	24 696	8.24×10^1	1	1	2	1	1	2	7.7×10^0	1	1	2	1	1	2
TEM27623	27 623	4.70×10^3	1	1	1	1	1	2	2.1×10^1	1	1	2	1	1	2
Goodwin054	32 510	4.49×10^6	8	5	6	8	5	5	2.2×10^4	5	5	6	9	5	4
thermal1	17 880	1.60×10^3	3	4	4	3	4	3	8.0×10^1	3	3	4	3	3	3
Zhao1	33 861	4.45×10^2	3	4	4	3	4	3	1.6×10^1	3	4	4	3	4	3
nd6k	18 000	3.82×10^2	3	4	4	3	4	3	1.0×10^2	2	2	3	2	2	2
nd12k	36 000	3.51×10^2	3	4	4	3	4	3	1.0×10^2	1	1	2	1	1	2
epb1	14 734	2.62×10^4	3	4	4	3	4	3	4.4×10^4	3	4	4	4	4	3
appu	14 000	1.02×10^4	4	4	5	3	4	3	5.8×10^3	4	4	5	3	4	3
ns3Da	20 414	3.79×10^3	3	4	4	3	4	3	3.4×10^3	3	4	4	—	—	—
mixtank new	29 957	4.69×10^{11}	—	36	31	13	11	10	2.1×10^5	8	9	8	—	—	—
e40r0100	17 281	2.23×10^8	44	9	11	—	9	8	1.1×10^4	5	6	6	—	—	—
bcsstk25	15 439	1.02×10^{10}	—	—	—	1	2	4	6.2×10^2	3	4	4	3	4	3
bcsstk37	25 503	2.27×10^9	—	—	—	2	2	4	1.5×10^2	3	4	4	3	4	3
raefsky3	21 200	5.18×10^{11}	—	—	—	4	4	11	1.2×10^4	7	6	8	6	6	5
thread	29 736	1.78×10^8	—	—	—	2	2	3	1.3×10^3	3	4	4	3	3	3
sme3Db	29 067	1.05×10^8	—	15	9	74	13	10	3.8×10^7	—	20	11	—	18	9
mult dcop 01	25 187	9.55×10^{11}	—	—	—	—	108	62	1.5×10^8	—	47	34	—	63	26
ramage02	16 830	4.52×10^8	—	—	—	—	—	—	4.5×10^7	—	179	83	—	—	—

Mixed-Precision Multigrid

Algorithm 15: MPV (multi-precision multigrid V-cycle)

```
%% For any level ℓ, give a nonsingular matrix A_ℓ and prolongation P_ℓ;
%% Compute the residual r_ℓ of the current iteration and set precisions u_ℓ;
Store residual r_ℓ in precision u_ℓ;
x_ℓ ← M r_ℓ in precision u_ℓ; %% pre-smoothing
if ℓ > 0 %% not the coarsest level yet
    r_ℓ ← r_ℓ − A x_ℓ in precision u_ℓ; %% form residual
    r_{ℓ−1} ← P_ℓ^T r_ℓ in precision u_ℓ; %% restriction
    e_{ℓ−1} ← V(A_{ℓ−1}, r_{ℓ−1}, P_{ℓ−1}, ℓ − 1); %% recursive call
    e_ℓ ← P_ℓ e_{ℓ−1} in precision u_ℓ; %% prolongation
    x_ℓ ← x_ℓ + e_ℓ in precision u_ℓ; %% correction
    r_ℓ ← r_ℓ − A e_ℓ in precision u_ℓ; %% update residual (optional)
    x_ℓ ← x_ℓ + M^T r_ℓ in precision u_ℓ; %% post-smoothing (optional)
end
return x_ℓ
```

Ref: McCormick, Stephen F., Joseph Benzaken, and Rasmus Tamstorf. “Algebraic error analysis for mixed-precision multigrid solvers”. SIAM Journal on Scientific Computing 43.5 (2021): S392-S419.

Sources of Error in Simulation, Revisited

Floating-point error

$$u(x) = U_h(x) + \mathcal{E}_{\text{dis}} + \mathcal{E}_{\text{alg}} + \overbrace{\mathcal{E}_{\text{quant}} + \mathcal{E}_{\text{round}}}$$

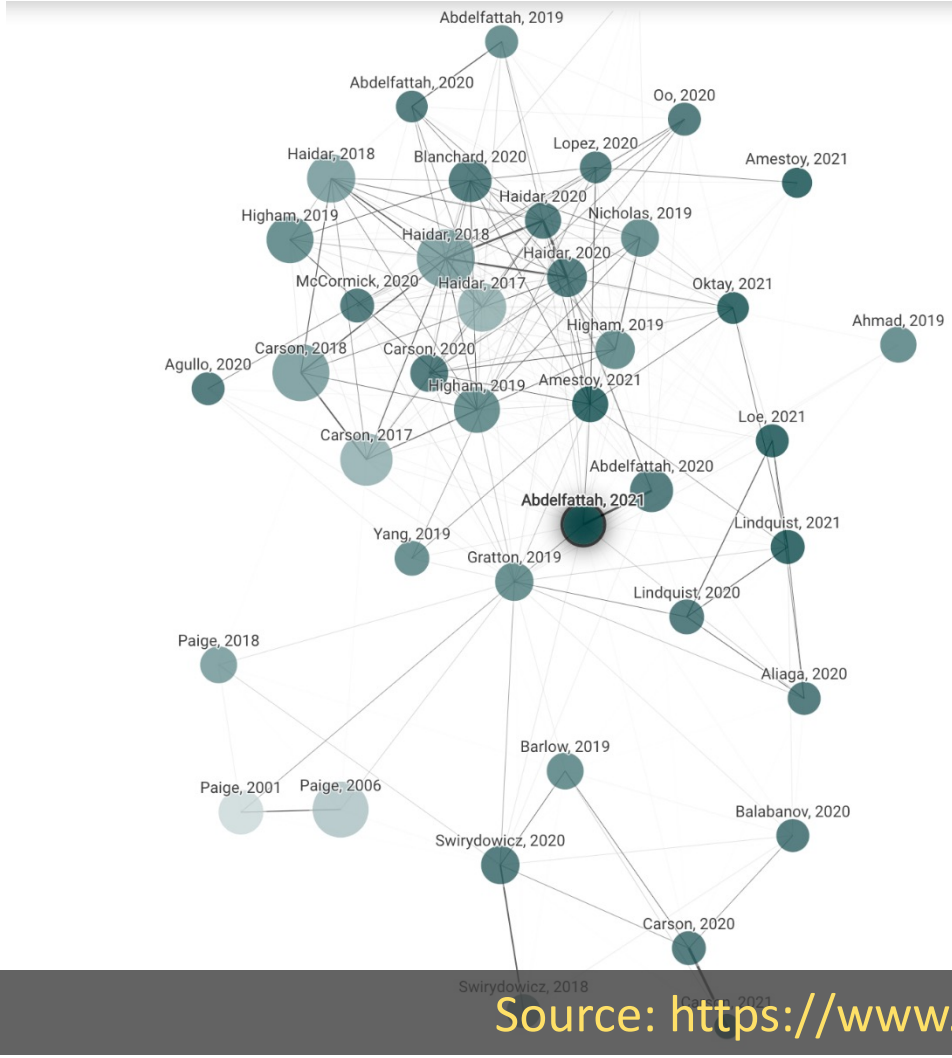
- Quantization error: The error caused by representing algebraic system in floating-point storage formats
- Rounding error: The error caused by storing an exact solution to the algebraic system in floating-point storage formats
- Quantization error might be the dominating term!?

Ref: Rasmus Tamstorf, Joseph Benzaken, and Stephen F. McCormick. “Discretization-Error-Accurate Mixed-Precision Multigrid Solvers”. SIAM Journal on Scientific Computing 2021 43:5, S420-S447

Some Software with MP Support

Software	Description	MP Support
GINKGO	A modern linear algebra library engineered towards performance portability, and productivity	✓ memory accessor that encapsulates on-the-fly compression for decoupling the memory precision from the arithmetic precision ✓ mix & match precisions in the sense of combining linear operators and vectors stored in different precision formats without explicit conversion ✓ mixed precision SAI preconditioning or CB-GMRES
Trilinos, Kokkos	An object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems on new and emerging high-performance computing (HPC) architectures.	✓ half precision capabilities in Kokkos Core ✓ single precision MueLu (multigrid) and IfPack2 (factorization- based) preconditioners ✓ GMRES-IR solvers
MAGMA	dense (BLAS and LAPACK), sparse, and batched linear algebra routines for single nodes with GPUs	✓ mixed-precision iterative refinement solvers based on LU, Cholesky, and QR factorizations

Mix-Precision NLA Papers



A survey of numerical linear algebra methods utilizing mixed-precision arithmetic

A. Abdelfattah, H. Anzt, E. Boman, E. Carson, T. Cojean, J. Dongarra, Alyson Fox, M. Gates, N. Higham, Xiaoye S. Li, J. Loe, P. Luszczek, S. Pranesh, S. Rajamanickam, T. Ribizel, Barry Smith, K. Swirydowicz, Stephen J. Thomas, S. Tomov, Y. Tsai, U. Yang

2021, Int. J. High Perform. Comput. Appl. ...

13 Citations Save

Open in:

The efficient utilization of mixed-precision numerical linear algebra algorithms can offer attractive acceleration to scientific computing applications. Especially with the hardware integration of low-precision special-function units designed for machine learning applications, the traditional numerical algorithms community urgently needs to reconsider the floating point formats used in the distinct operations to efficiently leverage the available compute power. In this work, we provide a comprehensive survey of mixed-precision numerical linear algebra routines, including the underlying concepts, theoretical background, and experimental results for both dense and sparse linear algebra problems.

Source: https://www.connectedpapers.com/

Reading and Thinking

Acta Numerica (2021), pp. 1–69
doi:10.1017/S09624929XXXXXXXX

© Cambridge University Press, 2021
Printed in the United Kingdom

Mixed precision algorithms in numerical linear algebra

Nicholas J. Higham
Department of Mathematics, University of Manchester, Manchester, M13 9PL, UK
nick.higham@manchester.ac.uk

Theo Mary
Sorbonne Université, CNRS, LIP6, Paris, F-75005, France
theo.mary@lip6.fr

- Did you have trouble with floating-point precision related problems?
- Did you experience trouble to reach desired precision in computation?
- What was the reason? How did you handle them?
- Is the solvers or other algorithms you use numerically stable?
- Can you use mixed precision some where in your code?

Contact Me

- Office hours: Mon 14:00—15:00
- Walk-in or online with appointment
- zhangcs@lsec.cc.ac.cn
- http://lsec.cc.ac.cn/~zhangcs

My sincere gratitude to:

Xiaowen Xu, Ningxi Tian, Ting Lai, Bin Dai

中国科学院大学
夏季强化课程
2022

Fast Solvers for Large Algebraic Systems

Chensong Zhang, AMSS

http://lsec.cc.ac.cn/~zhangcs

Release version 2022.07.02