Reduced- and mixed-precision finite element kernels

M. Croci (Ikerbasque & BCAM)
Joint with: G. N. Wells (University of Cambridge)

FEniCS 2024. Simula Research Laboratory, Oslo, Norway, 14 June 2024

This work is supported by the European Union Horizon research and innovation program under the Marie Sklodowska-Curie postdoctoral fellowship grant 101103593 (GEOLEARN).

1. Background

Floating point formats

Recent trend in chip manufacturing: Many new chips supporting low-precision computations. Double precision not prioritised due to AI focus.

Format	unit roundoff u		Range
fp64 (double)	2^{-53}	$\approx 1.11 \times 10^{-16}$	$10^{\pm 308}$
fp32 (single)	2^{-24}	$\approx 5.96 \times 10^{-8}$	$10^{\pm 38}$
fp16 (half)	2^{-11}	$\approx 4.88 \times 10^{-4}$	$10^{\pm 5}$
bfloat16 (half)	2^{-8}	$\approx 3.91 \times 10^{-3}$	$10^{\pm 38}$

Half vs double max speedups: $\times 4$ on CPUs, $\underline{\times 32+ \text{ on tensor-cores/AMX}}$.

Mixed-precision algorithms

Mixed-precision algorithms

Mixed-precision algorithms combine low- and high-precision computations in order to benefit from the performance gains of reduced-precision while retaining good accuracy.

Mixed-precision algorithms

Mixed-precision algorithms

Mixed-precision algorithms combine low- and high-precision computations in order to benefit from the performance gains of reduced-precision while retaining good accuracy.

Today's focus

Designing efficient mixed-precision finite element cell kernels.

Motivation (see e.g., [Abdelfattah et al. 2021])

Many mixed-precision algorithms exploit reduced-precision operators for speedup: Preconditioning, linear/nonlinear solvers, and timestepping methods.

Mixed-precision algorithms

Mixed-precision algorithms

Mixed-precision algorithms combine low- and high-precision computations in order to benefit from the performance gains of reduced-precision while retaining good accuracy.

Today's focus

Designing efficient mixed-precision finite element cell kernels.

Motivation (see e.g., [Abdelfattah et al. 2021])

Many mixed-precision algorithms exploit reduced-precision operators for speedup: Preconditioning, linear/nonlinear solvers, and timestepping methods.

Warning: Work in progress!

AMX accelerators on Intel Xeon CPUs

AMX accelerators on Intel Xeon CPUs

AMX implement pairwise matrix-matrix multiply and accumulate as follows:

$$S \mathrel{+}= AB + CD, \quad \{A, C\} \subset \mathbb{R}^{m \times k}, \ \{B, D\} \subset \mathbb{R}^{k \times n},$$

where A, B, C, and D are stored in bf16 and S in single precision. Operations carried out in single. Max sizes: $m, k, n = 16$.

AMX accelerators on Intel Xeon CPUs

AMX implement pairwise matrix-matrix multiply and accumulate as follows:

$$S \mathrel{+}= AB + CD, \quad \{A, C\} \subset \mathbb{R}^{m\times k},\ \{B, D\} \subset \mathbb{R}^{k\times n},$$

where A, B, C, and D are stored in bf16 and S in single precision. Operations carried out in single. Max sizes: $m, k, n = 16$.

AMX can be up to $64\times$ faster than double precision vectorized computations!

AMX accelerators on Intel Xeon CPUs

AMX implement pairwise matrix-matrix multiply and accumulate as follows:

$$S \mathrel{+}= AB + CD, \quad \{A, C\} \subset \mathbb{R}^{m\times k},\ \{B, D\} \subset \mathbb{R}^{k\times n},$$

where A, B, C, and D are stored in bf16 and S in single precision. Operations carried out in single. Max sizes: $m, k, n = 16$.

AMX can be up to $64\times$ faster than double precision vectorized computations!

Of course, AMX cannot be used for everything.

2. Mixed-precision finite element kernels

Challenge: Exploiting MP accelerators in FE cell kernels

Challenge: Exploiting MP accelerators in FE cell kernels

The Masterplan

A) **Make mat-mat products the bottleneck.**
B) **Rounding error analysis.**
C) **Implementation.**

Challenge: Exploiting MP accelerators in FE cell kernels

The Masterplan

A) **Make mat-mat products the bottleneck.**

B) **Rounding error analysis.**

C) **Implementation.**

Example: Poisson form over a single cell $K \subset \mathbb{R}^d$,

$$a_K(u_h, v_h) = \int_K \nabla u_h \cdot \nabla v_h \,\mathrm{d}x, \quad u_h, v_h \in V_h|_K = \mathrm{span}(\{\phi_i\}_{i=1}^m).$$

Cell kernels: Compute local matrix resulting from the above form (can also do actions):

$$A_{ij} = a_K(\phi_j, \phi_i), \quad A \in \mathbb{R}^{m\times m},$$

A) Cell kernels as sum of mat-mat products

It can be shown that A can be expressed as a sum of triple matrix products:

$$A = \sum_s \sum_t B_s D_{st} B_t^T, \quad B_s \in \mathbb{R}^{m \times n_q}, \quad D_{st} \in \mathbb{R}^{n_q \times n_q},$$

$B =$ Derivatives of basis functions at quadrature points (3D tensor).
$D =$ Quadrature weights and geometry at quadrature points (4D sparse tensor).

For actions: Use cell batching $\implies$ Also obtain sum of mat-mat prods.

B) Rounding error analysis

Theorem (proof in progress)

Computing A using the following precisions:

- u_{store} for storing B and computing the action of D,
- u_q for performing and accumulating mat-mat products,
- u_g for computing the geometry tensor,
- u_p for evaluating basis functions on the reference cell,

yields instead $\hat{A}$ satisfying

$$\|A - \hat{A}\|_\infty \lesssim \left(u_{\text{store}} + (d^2 + n_q)u_q + \kappa_\infty(J)u_g + p^d u_p \right) \|A\|_\infty.$$

B) Rounding error analysis

Cost-accuracy trade off. Want to use low precision for speed, but stay accurate.

Objective: Obtain $O(u)$ accuracy where u is the half-precision unit roundoff.

B) Rounding error analysis

Cost-accuracy trade off. Want to use low precision for speed, but stay accurate.

Objective: Obtain $O(u)$ accuracy where u is the half-precision unit roundoff.

$$\|A - \hat{A}\|_\infty \lesssim \left(u_{\mathsf{store}} + (d^2 + n_q)u_q + \kappa_\infty(J)u_g + p^d u_p\right) \|A\|_\infty.$$

B) Rounding error analysis

Cost-accuracy trade off. Want to use low precision for speed, but stay accurate.

Objective: Obtain $O(u)$ accuracy where u is the half-precision unit roundoff.

$$\|A - \hat{A}\|_\infty \lesssim \left(u_{\mathsf{store}} + (d^2 + n_q)u_q + \kappa_\infty(J)u_g + p^d u_p\right) \|A\|_\infty.$$

AMX/tensor core strategy:

B) Rounding error analysis

Cost-accuracy trade off. Want to use low precision for speed, but stay accurate.

Objective: Obtain $O(u)$ accuracy where u is the half-precision unit roundoff.

$$\|A-\hat{A}\|_\infty \lesssim \left(u_{\mathsf{store}} + (d^2+n_q)u_q + \cancel{\kappa_\infty(J)u_g} + \cancel{p^d u_p}\right)\|A\|_\infty.$$

AMX/tensor core strategy:

1. High-precision for geometry and basis evaluation.

B) Rounding error analysis

Cost-accuracy trade off. Want to use low precision for speed, but stay accurate.

Objective: Obtain $O(u)$ accuracy where u is the half-precision unit roundoff.

$$\|A - \hat{A}\|_\infty \lesssim \left(u + \cancel{(d^2 + n_q)u_q} + \cancel{\kappa_\infty(J)u_g} + \cancel{p^d u_p}\right) \|A\|_\infty.$$

AMX/tensor core strategy:

1. High-precision for geometry and basis evaluation.
2. Cast to half and use AMX for the rest.

C) Implementation (C++)

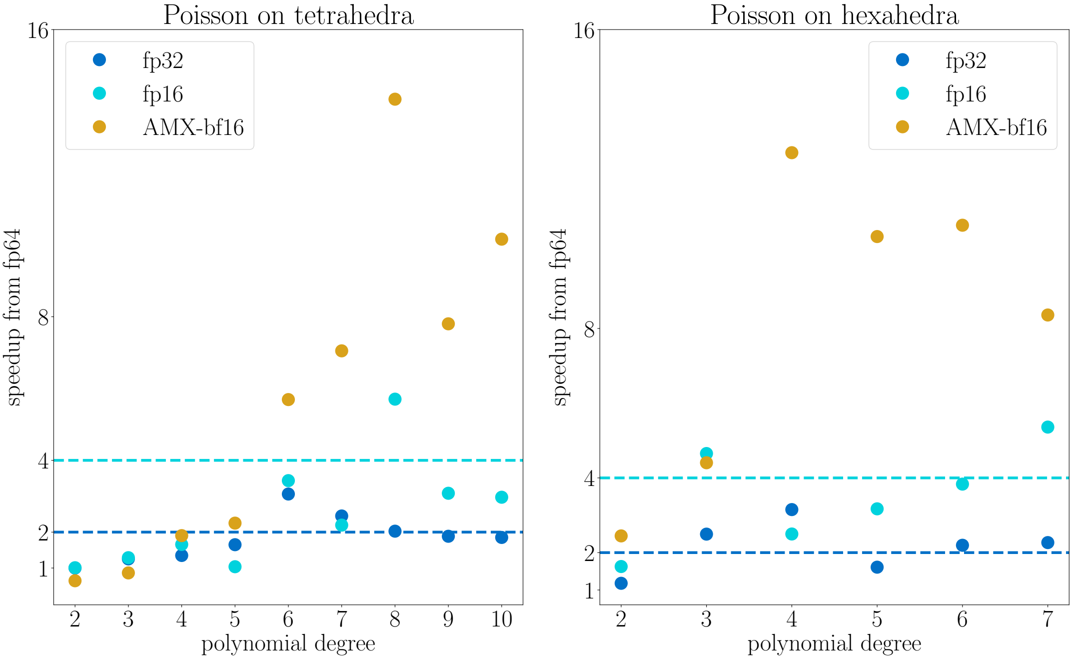
Implementation challenges

1. Brand new chip functionalities mean lack of compiler support and bugs.
2. Compilers won't use AMX for you. Exotic system calls and Intel intrinsics needed.
3. AMX library support limited/bugged, yet growing.

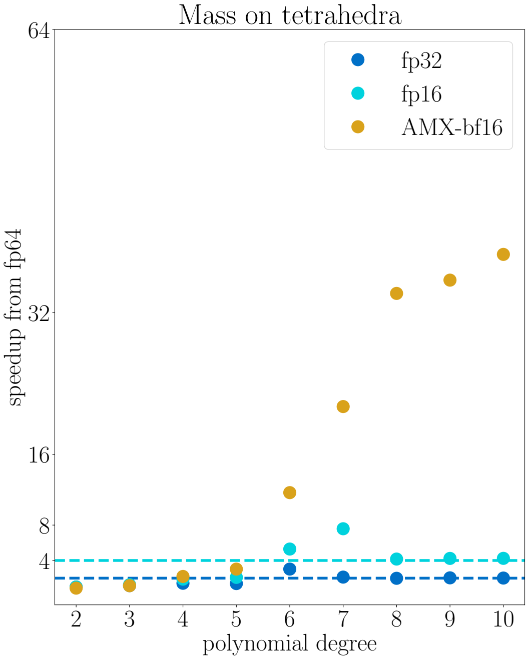
Testing and debugging: Use FFCX-generated kernels.

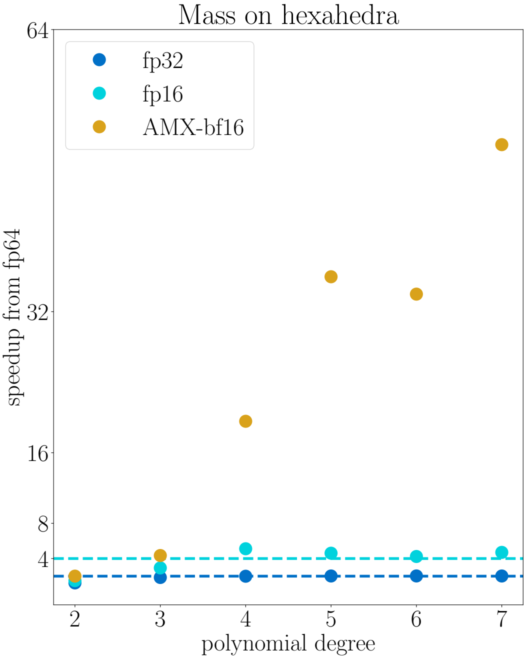
3. Numerical results

Numerical results - Poisson form in 3D



Numerical results - Mass form in 3D





4. Conclusions

Conclusions

To sum up

- We can compute FE kernels up to $10 - 50\times$ faster using AMX. Implementation was admittedly challenging. Luckily, AMX software support is growing.
- Multiple applications can benefit from faster FE kernels: Preconditioning, iterative refinement, inexact Krylov and Newton solvers, timestepping methods, etc.
- GPU kernels and algorithmic applications will be addressed in future work.

Conclusions

To sum up

- We can compute FE kernels up to $10-50\times$ faster using AMX. Implementation was admittedly challenging. Luckily, AMX software support is growing.
- Multiple applications can benefit from faster FE kernels: Preconditioning, iterative refinement, inexact Krylov and Newton solvers, timestepping methods, etc.
- GPU kernels and algorithmic applications will be addressed in future work.

Thank you for listening!

Thank you for listening!

More info about me and my work at: `croci.github.io`

---

[1] M. Croci and G. N. Wells. Mixed-precision finite element kernels and their acceleration. *In preparation*, 2024.

[2] I. A. Baratta, J. P. Dean, J. S. Dokken, M. Habera, J. Hale, C. Richardson, M. E. Rognes, M. W. Scroggs, N. Sime, and G. N. Wells. DOLFINx: The next generation FEniCS problem solving environment. 2023.

[3] M. Fasi, N. J. Higham, M. Mikaitis, and S. Pranesh. Numerical behavior of NVIDIA tensor cores. *PeerJ Computer Science*, 7:e330, 2021.

[4] Wikipedia entry on Advanced Matrix Extensions. URL `https://en.wikipedia.org/wiki/Advanced_Matrix_Extensions`.

[5] P. Blanchard, N. J. Higham, and T. Mary. A class of fast and accurate summation algorithms. *SIAM journal on scientific computing*, 42(3):A1541–A1557, 2020.

[6] N. J. Higham and T. Mary. Mixed precision algorithms in numerical linear algebra. *Acta Numerica*, 31:347–414, 2022.

[7] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, et al. A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *The International Journal of High Performance Computing Applications*, 35(4):344–369, 2021.

[8] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002.