STATE-OF-THE ART GPU NUMERICAL COMPUTING

IN HONOR OF STAN TOMOV

APRIL 24-25, 2025 KNOXVILLE, TN

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

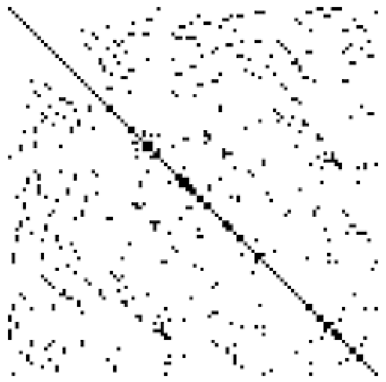
Batched Iterative Solvers for Sequences of Sparse Problems

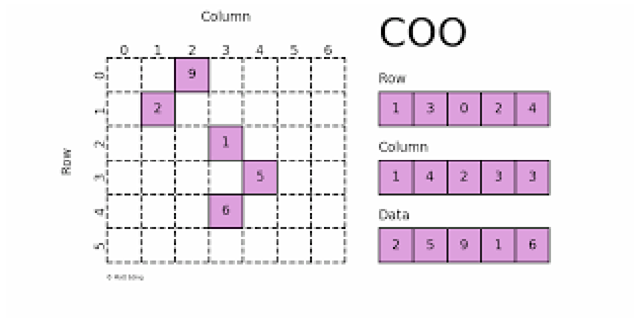
Hartwig Anzt, TU Munich & University of Tennessee

Technical University of Munich

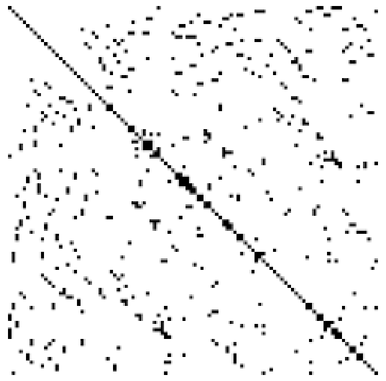
Sparse Linear Algebra

- In **sparse matrices** (and vectors) **most** of the matrix **values are zero.**
- Data structures for sparse matrices **ignore most or all explicit zeros.**
- **Sparse matrices** stored in sparse data structures are typically **large in size**.

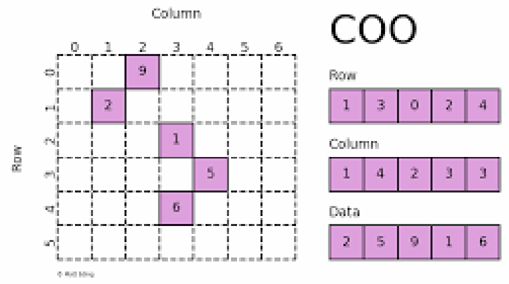




Sparse Linear Algebra



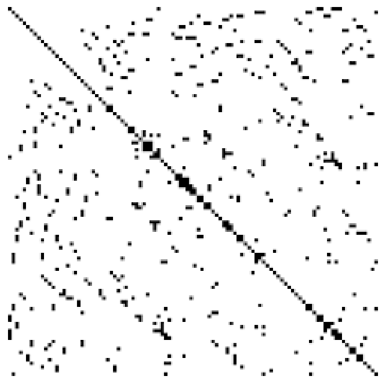
- In **sparse matrices** (and vectors) **most** of the matrix **values are zero.**
- Data structures for sparse matrices **ignore most or all explicit zeros.**
- **Sparse matrices** stored in sparse data structures are typically **large in size**.



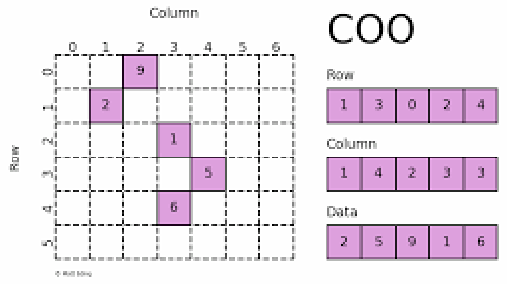
Why would you develop batched routines for operations on sparse matrices?

- *Sparse matrices are rarely of small size.*
- *Sparse matrices often make memory coalescing a challenge.*
- *All the same sparsity pattern? Same Format?*
- *Are there any application for data-parallel processing of sparse systems?*

Sparse Linear Algebra

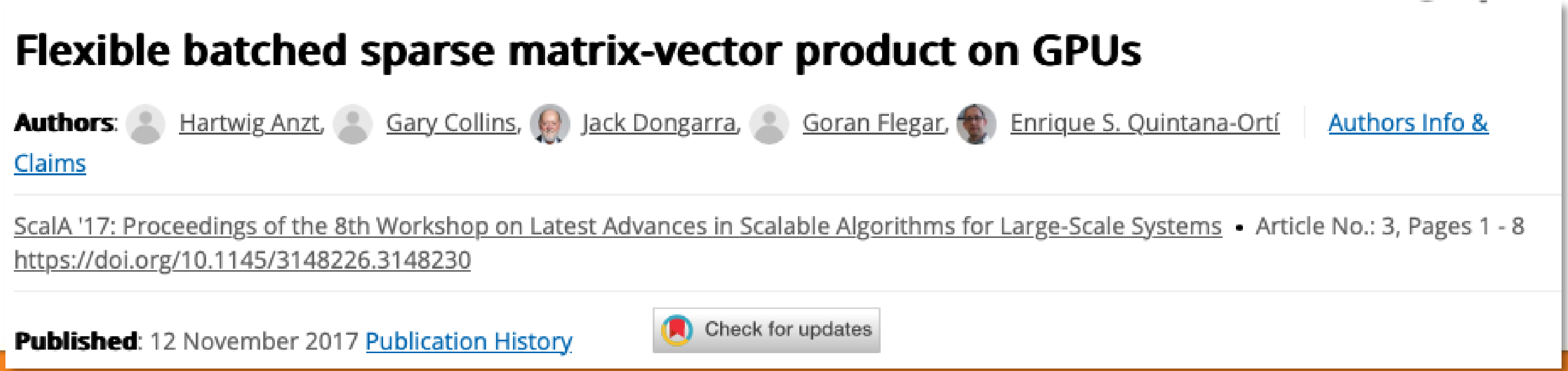


- In **sparse matrices** (and vectors) **most** of the matrix **values are zero.**
- Data structures for sparse matrices **ignore most or all explicit zeros.**
- **Sparse matrices** stored in sparse data structures are typically **large in size**.



Why would you develop batched routines for operations on sparse matrices?

- *Sparse matrices are rarely of small size.*
- *Sparse matrices often make memory coalescing a challenge.*
- *All the same sparsity pattern? Same Format?*
- *Are there any application for data-parallel processing of sparse systems?*

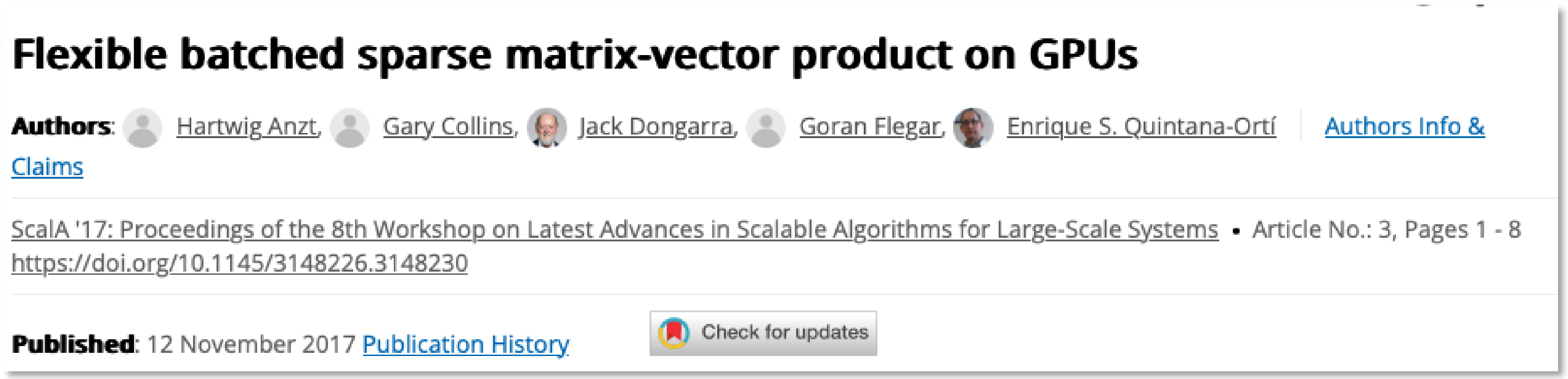


Flexible batched sparse matrix-vector product on GPUs

Authors: Hartwig Anzt, Gary Collins, Jack Dongarra, Goran Flegar, Enrique S. Quintana-Ortí | Authors Info & Claims

ScalA '17: Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems • Article No.: 3, Pages 1 - 8
https://doi.org/10.1145/3148226.3148230

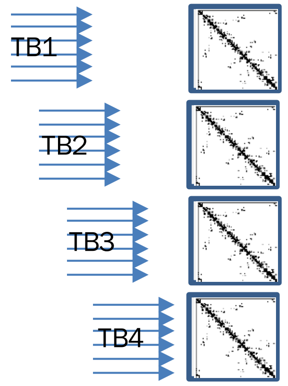
Published: 12 November 2017 Publication History


Flexible batched sparse matrix-vector product on GPUs

Authors: Hartwig Anzt, Gary Collins, Jack Dongarra, Goran Flegar, Enrique S. Quintana-Ortí Authors Info & Claims

ScalA '17: Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems • Article No.: 3, Pages 1 - 8
https://doi.org/10.1145/3148226.3148230

Published: 12 November 2017 Publication History Check for updates



Different scenarios:

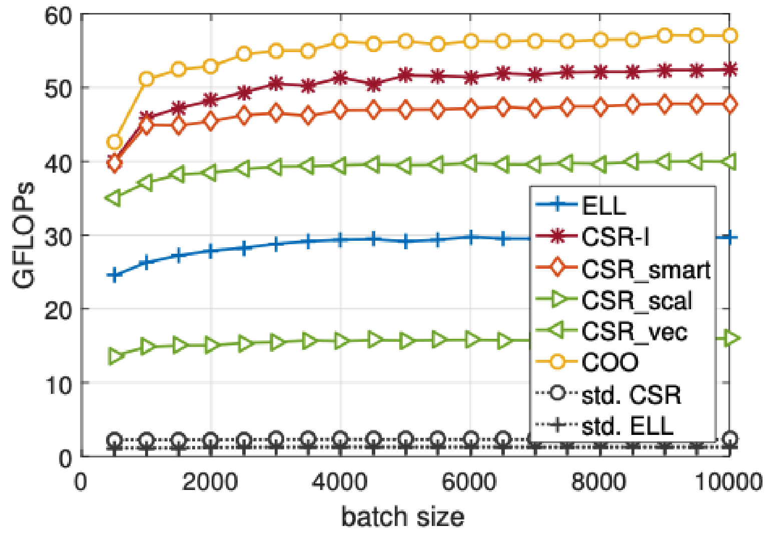
All SpMV operations of the batch have:

1. the same system size
(explicit zero padding to fix the sparsity pattern);

2. the same nonzero-per-row distribution
(allows reuse of row pointers/row indices);

3. the same nonzero locations
(reuse of row pointers/row indices and column indices);

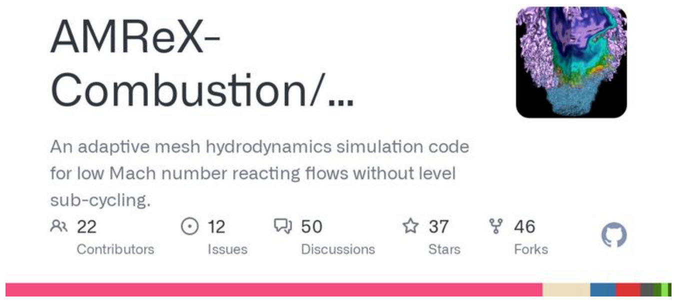
4. the same values but distinct sparsity patterns
(allows reuse of the values);



Performance of batched SpMV on NVIDIA P100 GPU for inhomogeneous batch of 32 matrices from the SuiteSparse matrix collection with n ∈ [11, 1015], nnz ∈ [76, 38352], nnz/n ∈ [3.0, 66.0].

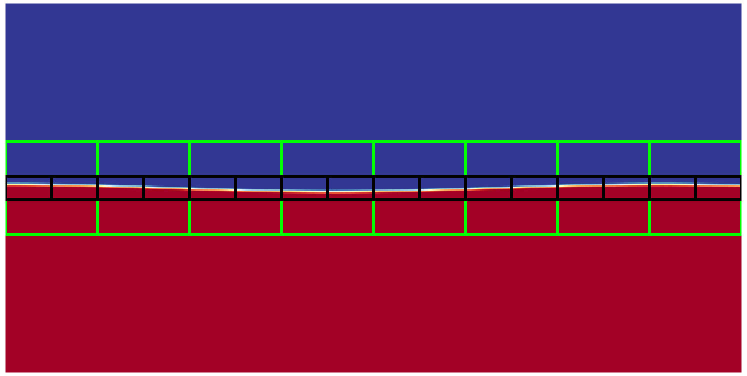
Combustion Simulations

PeleLM is a parallel, adaptive mesh refinement (AMR) code that solves the reacting Navier-Stokes equations in the low Mach number regime. The core libraries for managing the subcycling AMR grids and communication are found in the AMReX source code.


AMReX-Combustion/...

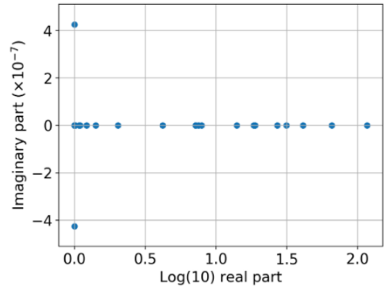
An adaptive mesh hydrodynamics simulation code for low Mach number reacting flows without level sub-cycling.

22 Contributors · 12 Issues · 50 Discussions · 37 Stars · 46 Forks

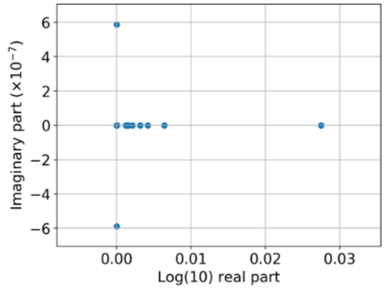


https://amrex-combustion.github.io/PeleLM/overview.html

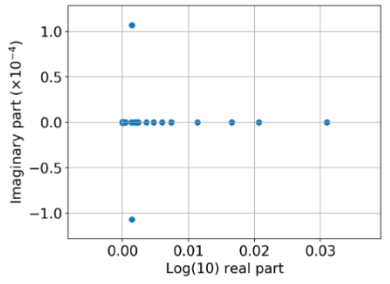
Problem	Size	Non-zeros (A)	Non-zeros (L+U)
dodecane_lu	54	2,332 (80%)	2,754 (94%)
drm19	22	438 (90%)	442 (91%)
gri12	33	978 (90%)	1,018 (93%)
gri30	54	2,560 (88%)	2,860 (98%)
isooctane	144	6,135 (30%)	20,307 (98%)
lidryer	10	91 (91%)	91 (91%)



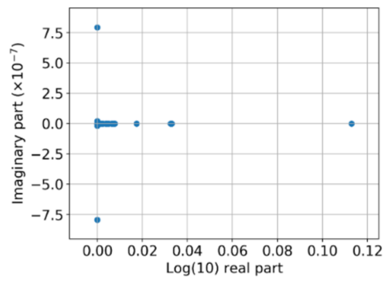
(a) dodecane_lu



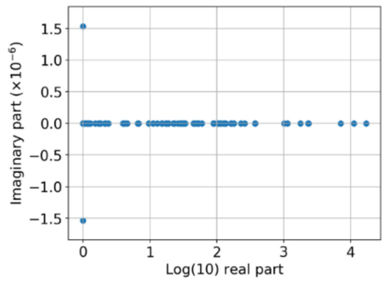
(b) drm19



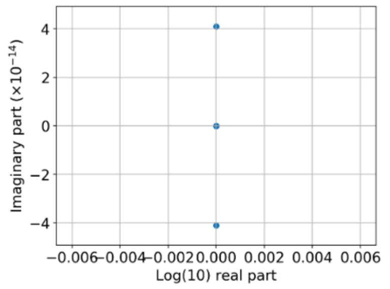
(c) gri12



(d) gri30



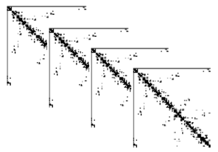
(e) isooctane



(f) lidryer

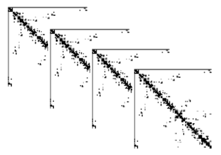
Batched Iterative Solvers

- Many sparse problems of medium size have to be solved concurrently.
  - ~ 50 – 2,000 unknowns, < 50% dense;
  - All sparse systems may share the same sparsity pattern;
  - An approximate solution may be acceptable (e.g., inside a non-linear solver);

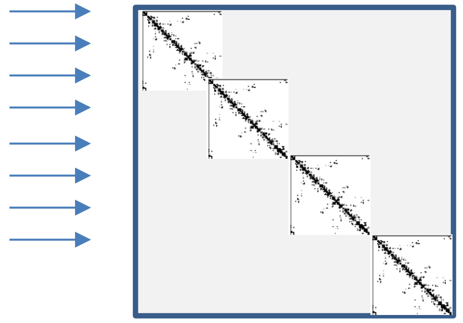


Batched Iterative Solvers

- Many sparse problems of medium size have to be solved concurrently.
  - ~ 50 – 2,000 unknowns, < 50% dense;
  - All sparse systems may share the same sparsity pattern;
  - An approximate solution may be acceptable (e.g., inside a non-linear solver);

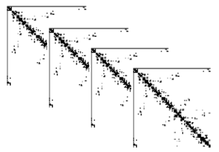


- One strategy is to arrange the individual systems on the main diagonal of one large system.
  - Convergence determined by the "hardest" problem;
  - No reuse of sparsity pattern information;
  - Global synchronization points;

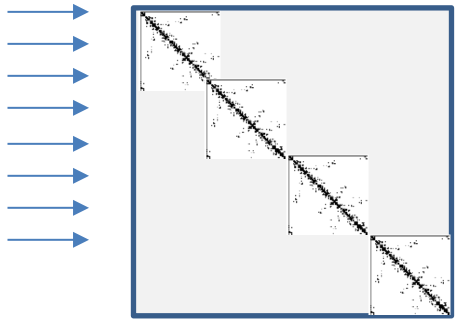


Batched Iterative Solvers

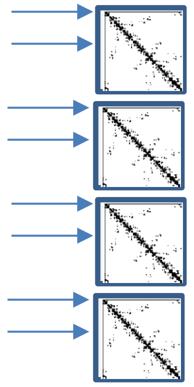
- Many sparse problems of medium size have to be solved concurrently.
    - ~ 50 – 2,000 unknowns, < 50% dense;
    - All sparse systems may share the same sparsity pattern;
    - An approximate solution may be acceptable (e.g., inside a non-linear solver);



- One strategy is to arrange the individual systems on the main diagonal of one large system.
    - Convergence determined by the "hardest" problem;
    - No reuse of sparsity pattern information;
    - Global synchronization points;



- Better approach: design batched iterative solve functionality that solves all problems concurrently.
    - Problem-dependent convergence accounted for;
    - No global synchronization;
    - Reuse of sparsity pattern information;
    - Parallelize across individual problems;



Kernel Execution Considerations

1. Batched functionality is generally memory-bound;
   -> Urgent need to minimize main memory access;

Implication for (dense) direct methods

- Algorithm steps need to be merged into one kernel (e.g. LU or inversion);

Implication for (sparse) iterative methods

- Interfacing external components via main memory impacts performance;
- All algorithm components have to be in-lined in the kernel code;

Kernel Execution Considerations

	Implication for (dense) direct methods	Implication for (sparse) iterative methods
1. Batched functionality is generally memory-bound; *-> Urgent need to minimize main memory access;*	• Algorithm steps need to be merged into one kernel (e.g. LU or inversion);	• Interfacing external components via main memory impacts performance; • All algorithm components have to be in-lined in the kernel code;
2. Different problems have different resource requirements; *-> hard to predict the register/shared memory requirement;*	• Different kernels for different problem sizes;	• Sparse matrix memory needs unknown; • Caching only for const data; • Shared memory space for intermediate vectors unknown;

Kernel Execution Considerations

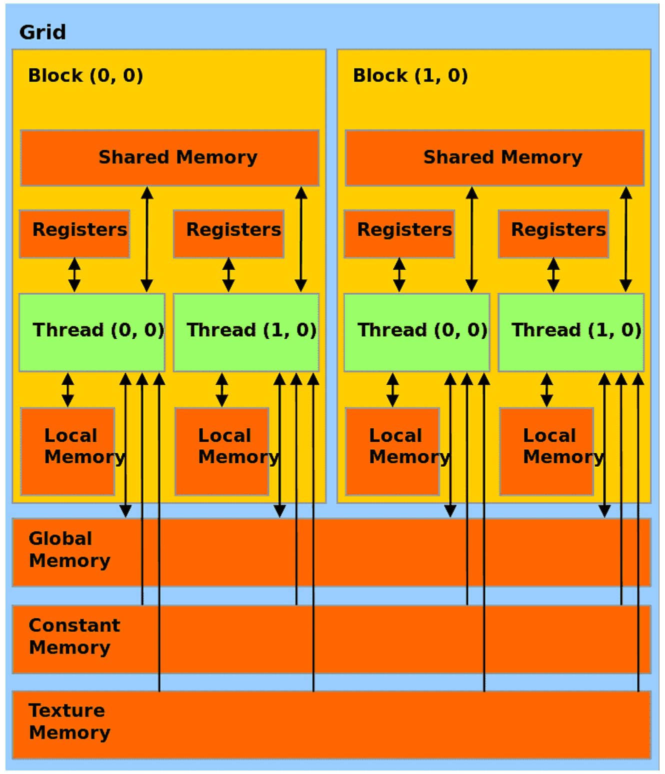
	Implication for (dense) direct methods	Implication for (sparse) iterative methods
1. Batched functionality is generally memory-bound; *-> Urgent need to minimize main memory access;*	• Algorithm steps need to be merged into one kernel (e.g. LU or inversion);	• Interfacing external components via main memory impacts performance; • All algorithm components have to be in-lined in the kernel code;
2. Different problems have different resource requirements; *-> hard to predict the register/shared memory requirement;*	• Different kernels for different problem sizes;	• Sparse matrix memory needs unknown; • Caching only for const data; • Shared memory space for intermediate vectors unknown;
3. Different problems may result in different algorithm behavior; -> unpredictable algorithm execution;	• Pivoting can result in some branching in the kernel execution;	• Need to monitor iterative solver convergence for each problem individually and complete early; • Need to schedule problems "appropriately";

Local / Global memory

```
$r \leftarrow b - Ax, \hat{r} \leftarrow r, p \leftarrow 0, v \leftarrow 0$
$\rho' \leftarrow 1, \omega \leftarrow 1, \alpha \leftarrow 1$
for $i < N_{iter}$ do
    if $\|r\| < \tau$ then
        Break
    end if
    $\rho \leftarrow r \cdot r'$
    $\beta \leftarrow \frac{\rho'\alpha}{\rho\omega}$
    $p \leftarrow r + \beta(p - \omega v)$
    $\hat{p} \leftarrow \text{PRECOND}(p)$
    $v \leftarrow A\hat{p}$
    $\alpha \leftarrow \frac{\rho}{\hat{r} \cdot v}$
    $s \leftarrow r - \alpha v$
    if $\|s\| < \tau$ then
        $x \leftarrow x + \alpha\hat{p}$
        Break
    end if
    $\hat{s} \leftarrow \text{PRECOND}(s)$
    $t \leftarrow A\hat{s}$
    $\omega \leftarrow \frac{t \cdot s}{t \cdot t}$
    $x \leftarrow x + \alpha\hat{p} + \omega\hat{s}$
    $r \leftarrow s - \omega t$
    $\rho' \leftarrow \rho$
end for
```

- Red objects: Intermediate vectors in SpMV: High priority to locate in shared memory
- Blue objects: Low priority
- Green objects: Constant matrices or vectors (cache)

Ginkgo



Choice dependent on hardware resources and target problem

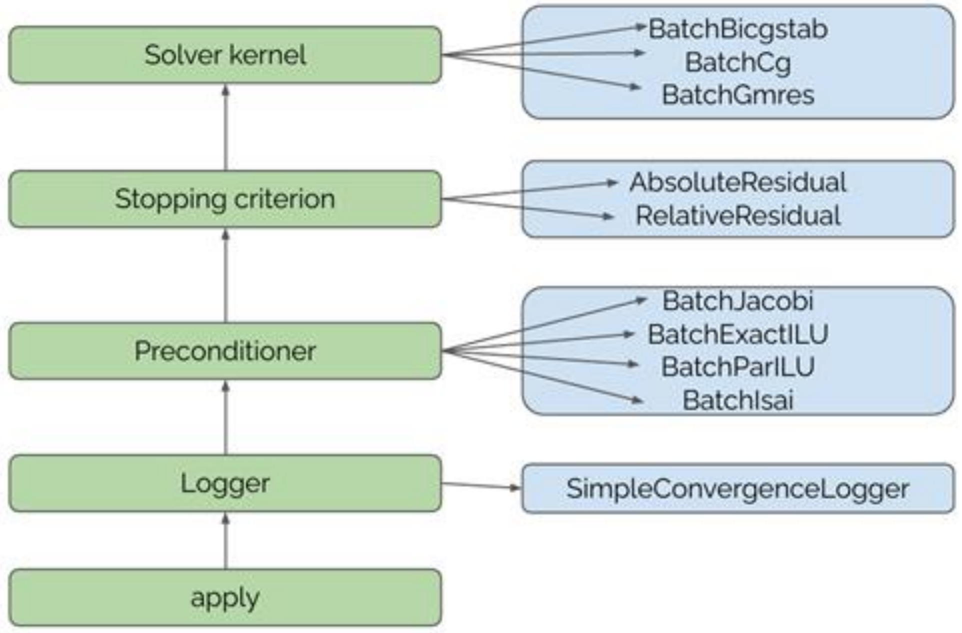
All operations must be in-lined to avoid main memory access

Inlining operations: SpMV, preconditioner…

```
$r \leftarrow b - Ax, \hat{r} \leftarrow r, p \leftarrow 0, v \leftarrow 0$
$\rho' \leftarrow 1, \omega \leftarrow 1, \alpha \leftarrow 1$
for $i < N_{iter}$ do
    if $\|r\| < \tau$ then
        Break
    end if
    $\rho \leftarrow r \cdot r'$
    $\beta \leftarrow \frac{\rho' \alpha}{\rho \omega}$
    $p \leftarrow r + \beta(p - \omega v)$
    $\hat{p} \leftarrow \text{PRECOND}(p)$
    $v \leftarrow A\hat{p}$
    $\alpha \leftarrow \frac{\rho}{\hat{r} \cdot v}$
    $s \leftarrow r - \alpha v$
    if $\|s\| < \tau$ then
        $x \leftarrow x + \alpha \hat{p}$
        Break
    end if
    $\hat{s} \leftarrow \text{PRECOND}(s)$
    $t \leftarrow A\hat{s}$
    $\omega \leftarrow \frac{t \cdot s}{t \cdot t}$
    $x \leftarrow x + \alpha \hat{p} + \omega \hat{s}$
    $r \leftarrow s - \omega t$
    $\rho' \leftarrow \rho$
end for
```

- Host side dispatch and the solver kernel is templated.
- Matrix format is templated.
- Preconditioner is templated.

Ginkgo



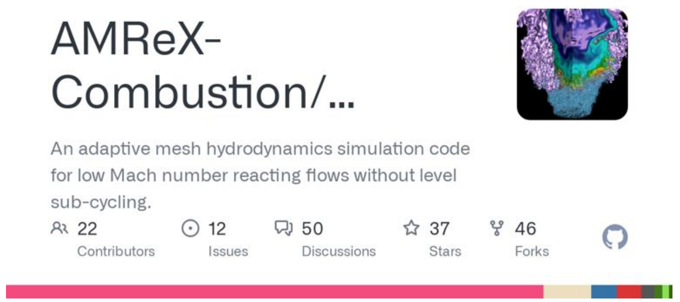
```
template <typename StopType, typename PrecType,
    typename LogType, typename BatchMatrixType,
    typename ValueType>
__global__ void apply_kernel(int padded_length,
    const StorageConf config, int max_iter,
    remove_complex<ValueType> tol,
    LogType logger, PrecType preconditioner,
    const BatchMatrixType a,
    const ValueType *__restrict__ b,
    ValueType *__restrict__ x,
    ValueType *__restrict__ workspace)
```

ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

Combustion Simulations

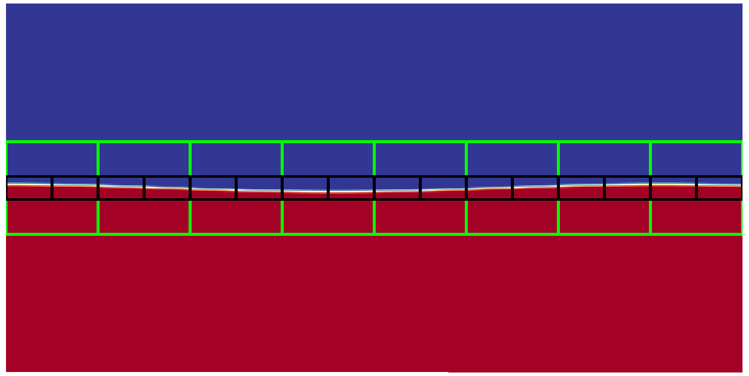
PeleLM is a parallel, adaptive mesh refinement (AMR) code that solves the reacting Navier-Stokes equations in the low Mach number regime. The core libraries for managing the subcycling AMR grids and communication are found in the AMReX source code.

https://amrex-combustion.github.io/PeleLM/overview.html

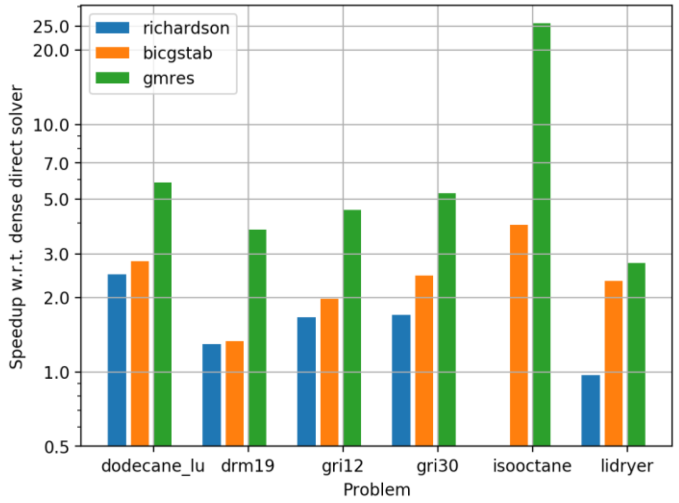

AMReX-Combustion/...

An adaptive mesh hydrodynamics simulation code for low Mach number reacting flows without level sub-cycling.

22 Contributors · 12 Issues · 50 Discussions · 37 Stars · 46 Forks



Problem	Size	Non-zeros (A)	Non-zeros (L+U)
dodecane_lu	54	2,332 (80%)	2,754 (94%)
drm19	22	438 (90%)	442 (91%)
gri12	33	978 (90%)	1,018 (93%)
gri30	54	2,560 (88%)	2,860 (98%)
isooctane	144	6,135 (30%)	20,307 (98%)
lidryer	10	91 (91%)	91 (91%)



Batched Sparse Iterative Solvers for Computational Chemistry Simulations on GPUs

Publisher: IEEE

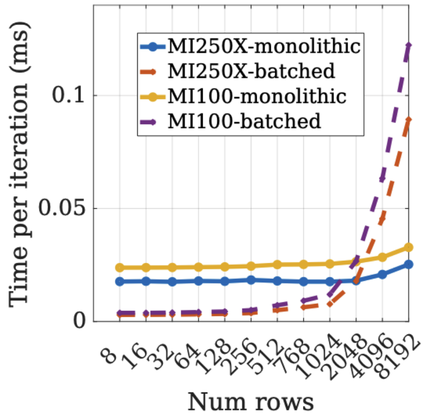
Cite This · PDF

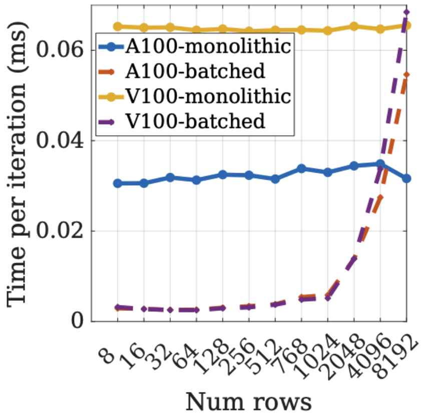
Isha Aggarwal ; Aditya Kashi ; Pratik Nayak ; Cody J. Balos ; Carol S. Woodward ; Hartwig Anzt All Authors

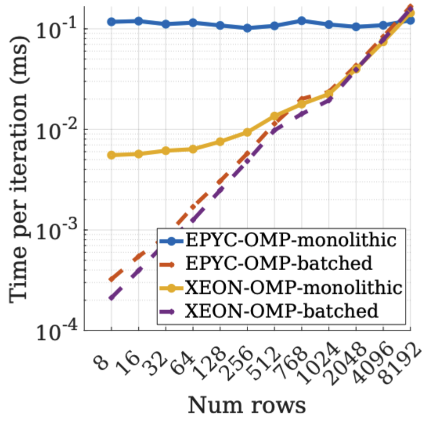
Batched solvers for monolithic problems

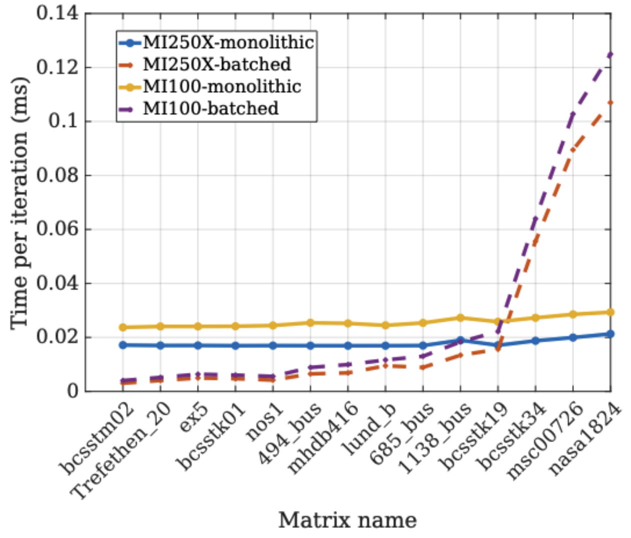
Architecture	*FLOP/s FP64* [TFlops]	*BW* (GB/s)	*L1 per CU* [KB]	*L2 per CU* [MB]	*# of SMs*	*Compiler Environment*
NVIDIA A100-40GB (Ampere)	9.7	1555	192	40	108	gcc-8.5 + CUDA-11.4
NVIDIA V100-16GB (Volta)	7.8	990	128	6	80	gcc-7.5 + CUDA-11.3
AMD MI250X-64GB (1 GCD)	25.9	1600	16+64	8	112	Clang-14 + ROCM-5.1
AMD MI100-32GB (CDNA)	11.5	1230	16+64	8	120	gcc-8.5 + ROCM-4.5
AMD EPYC-7032 (Rome)	1.5	208	64	16	32	gcc-8.5 + OpenMP 4.5
Intel Xeon Platinum (Ice Lake)	2.9	1000	64	38	38	gcc-8.5 + OpenMP 4.5

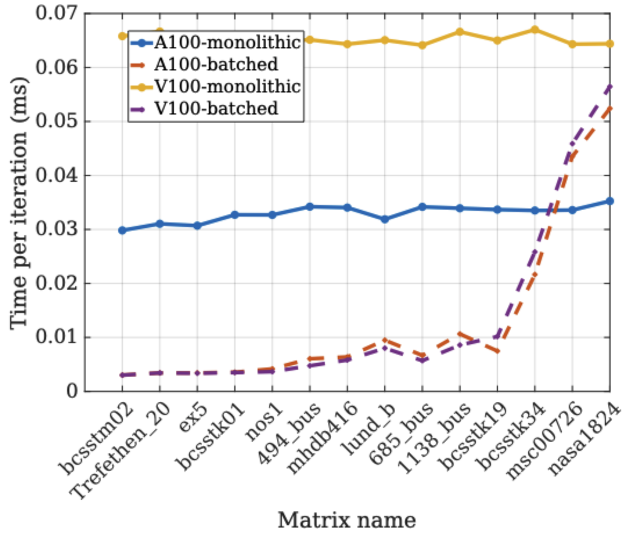
Laplace 1D

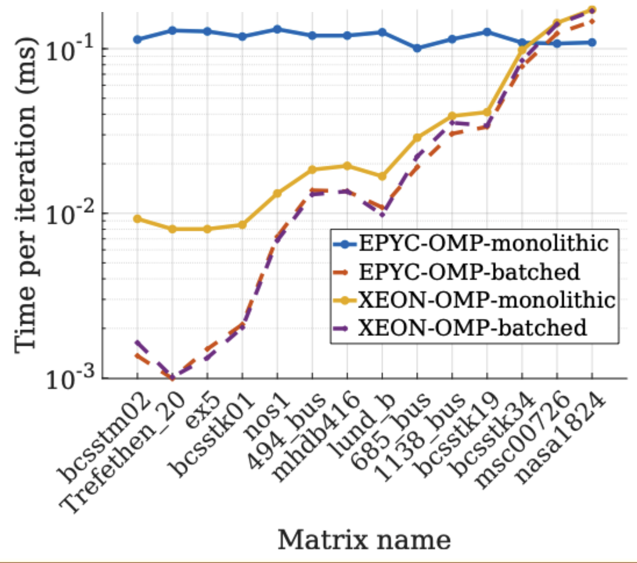








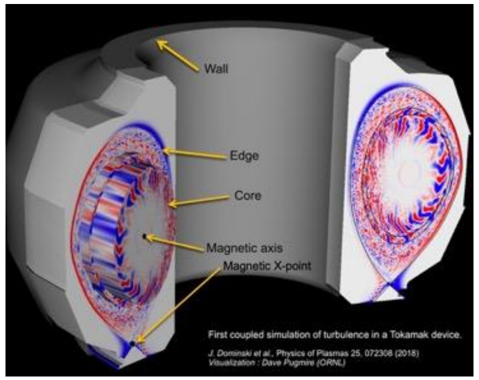


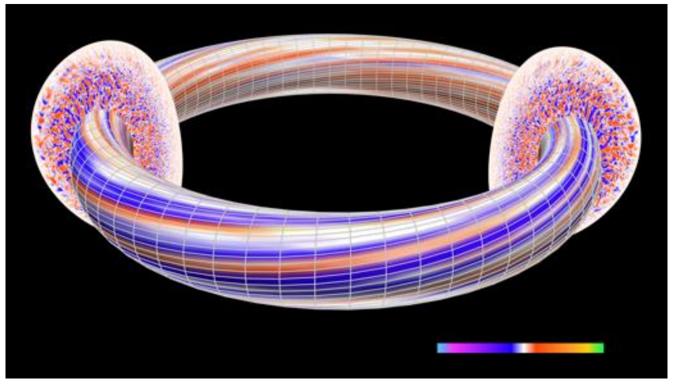


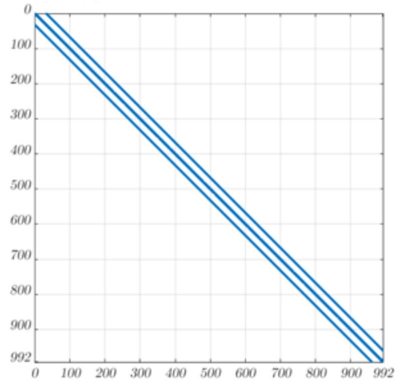
XGC DIII-D National Fusion Facility tokamak electromagnetic (EM) test case

XGC is a gyrokinetic particle-in-cell code, which specializes in the simulation of the edge region of magnetically confined thermonuclear fusion plasma. The simulation domain can include the magnetic separatrix, magnetic axis and the biased material wall. XGC can run in total-delta-f, and conventional delta-f mode. The ion species are always gyrokinetic except for ETG simulation. Electrons can be adiabatic, massless fluid, driftkinetic, or gyrokinetic.

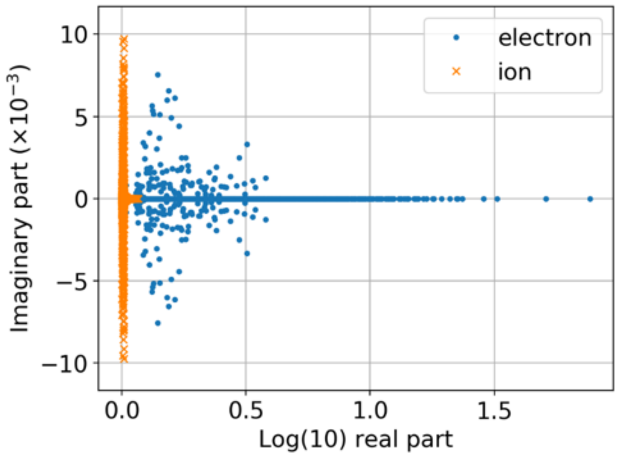
Source: https://xgc.pppl.gov/html/general_info.html

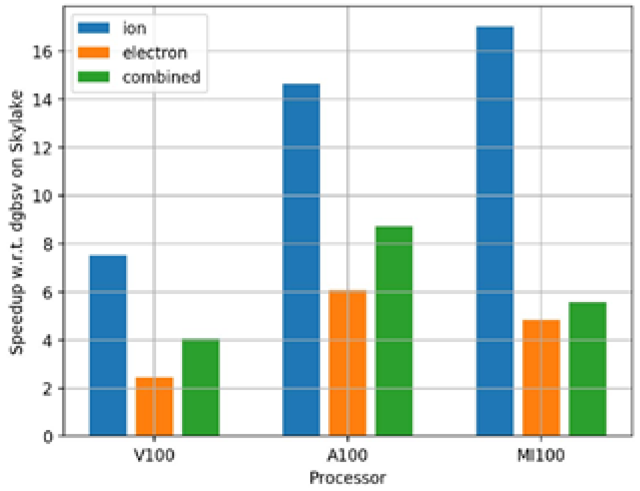

Wall
Edge
Core
Magnetic axis
Magnetic X-point
First coupled simulation of turbulence in a Tokamak device.
J. Dominski et al., Physics of Plasmas 25, 072308 (2018)
Visualization : Dave Pugmire (ORNL)



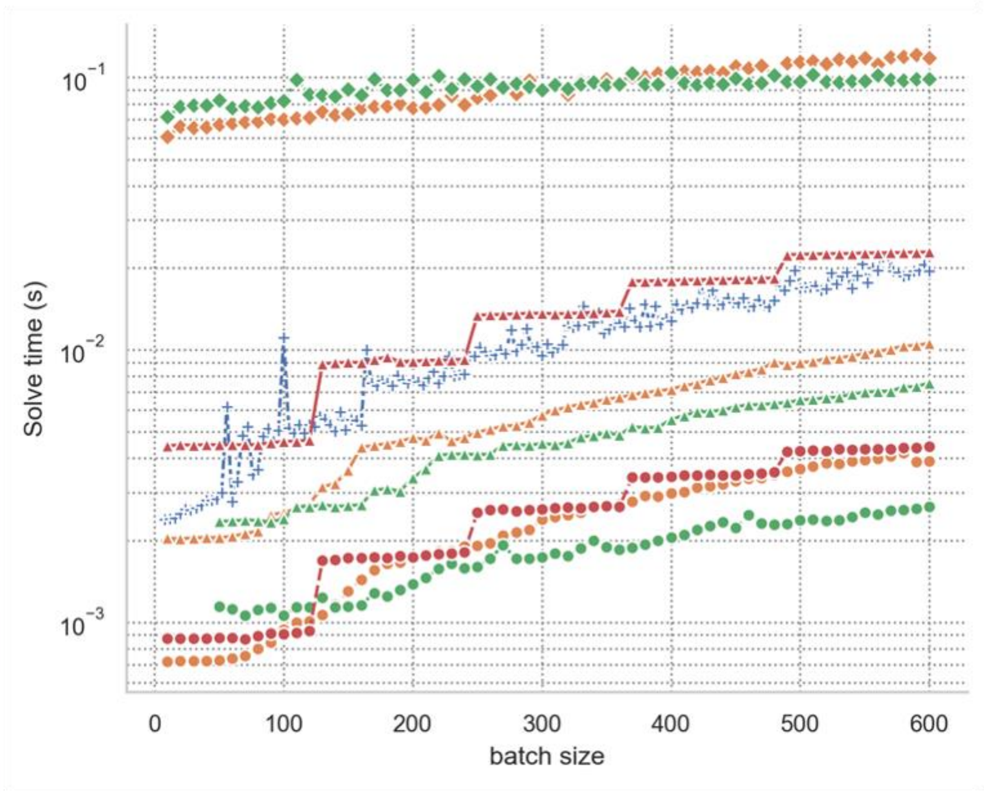


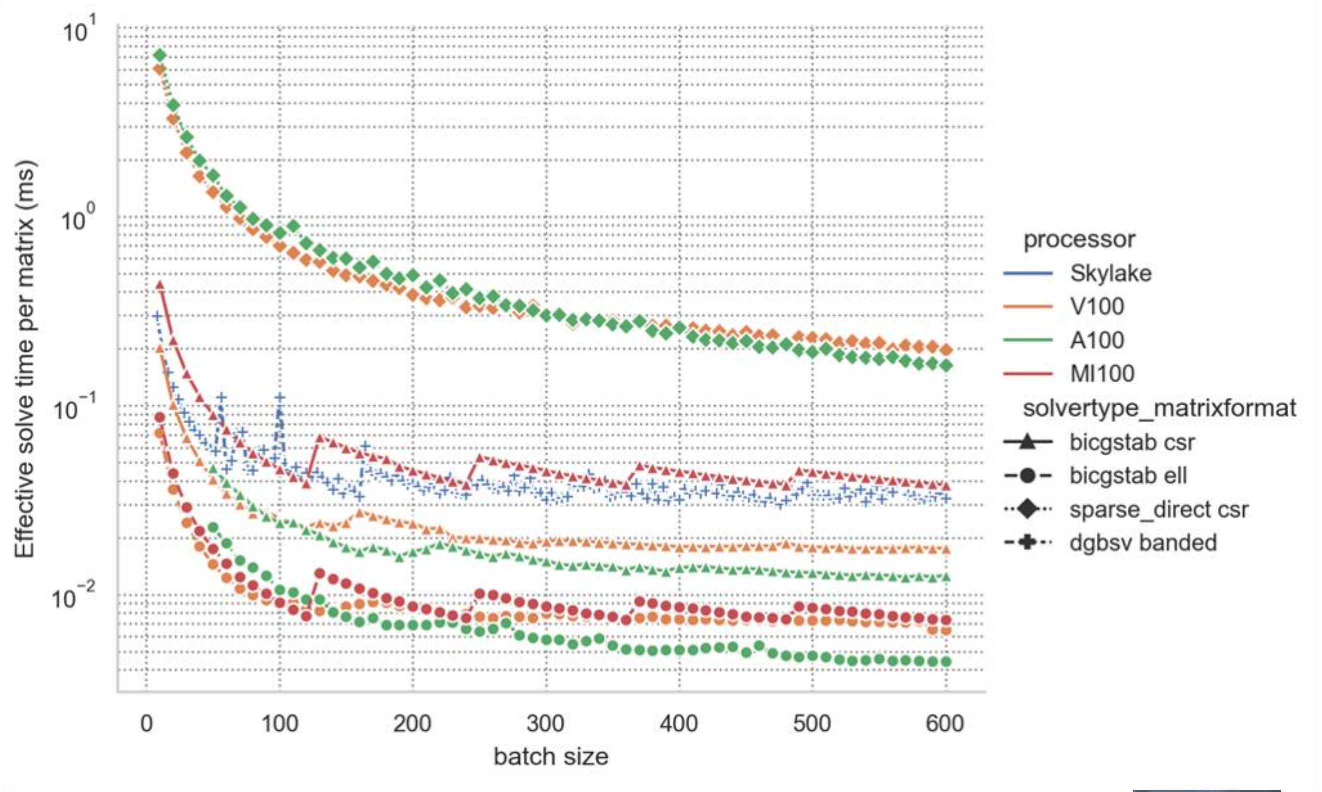
- Two species
- Ions easy to solve
- Electrons hard to solve
- Banded matrix structure
- Non-symmetric, BiCGSTAB
- n = ~1,000
- nz = ~9,000





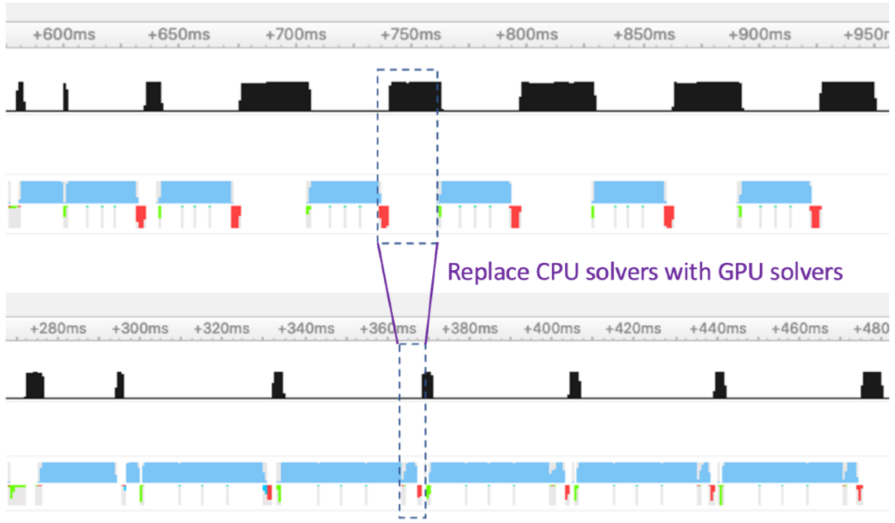
XGC DIII-D National Fusion Facility tokamak electromagnetic (EM) test case





Aditya Kashi, Pratik Nayak, Dhruva Kulkarni, Aaron Scheinberg, Paul Lin, and Hartwig Anzt. **Batched sparse iterative solvers on gpu for the collision operator for fusion plasma simulations**. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 157–167. IEEE, 2022.

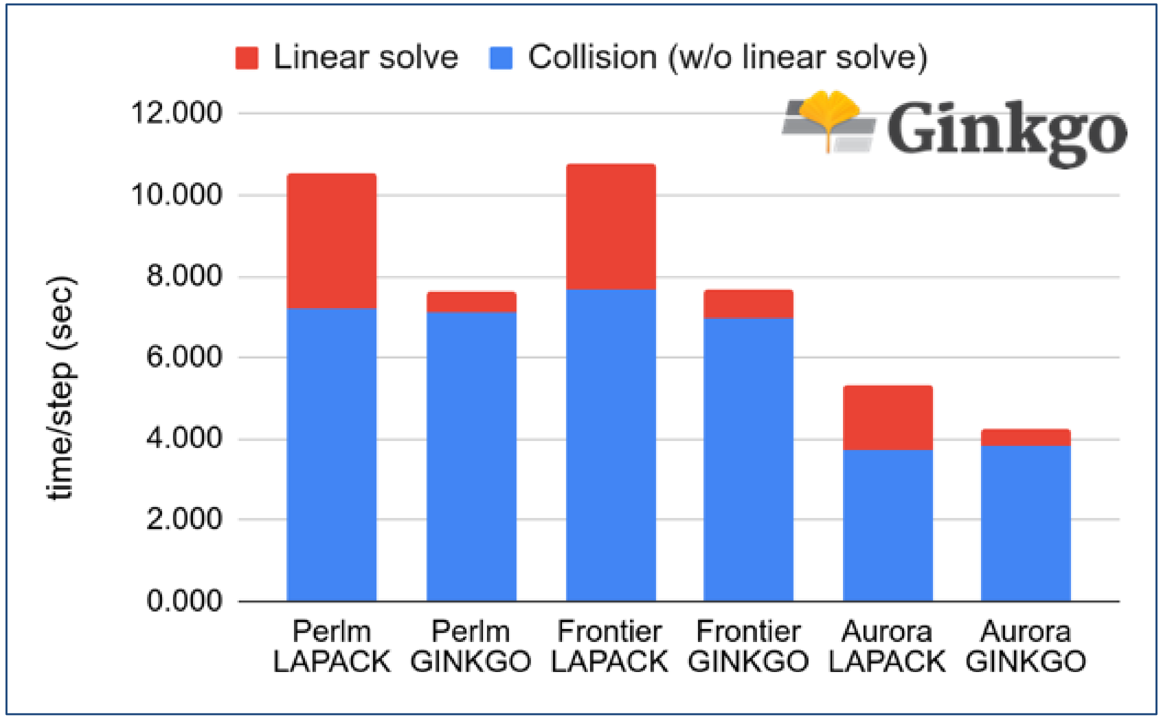
XGC DIII-D National Fusion Facility tokamak electromagnetic (EM) test case



- 8 nodes of NERSC Perlmutter: 32 A100s, 1 MPI per GPU; single socket 64-core AMD EPYC

- 8 nodes OLCF Frontier: 32 MI250X, 64 GCDs, 1 MPI per GCD; single socket 64-core AMD EPYC

- 8 nodes ALCF Aurora: 48 Intel Data Center Max 1550, 96 tiles, 1 MPI per tile; dual socket 52-core Intel CPU Max 9470C SPR



Findings from working with PeleLM and XGC production simulations

- **Applications need to solve many small problems in parallel – these are not always dense.**
  - Iterative solvers can be much faster if problems are well conditioned.
  - Performance heavily depends on problem characteristics and implementation.
- **Implementing batched sparse solvers is challenging.**
  - Convergence is problem-dependent.
  - One-kernel design necessary to achieve good performance. This includes the choice of matrix format and preconditioning.
- **Hardware characteristics and problem characteristics need to be met.**
  - Which data to locate in shared memory? (changing vectors).
  - Which data to locate in main memory, hope for caching? (constant data, matrix)
- **Very application dependent**
  - Need to implement kernel and run on hardware to see benefits for a given problem.
  - Much harder to develop off-the-shelf solutions.