Utilizing batched solver ideas for efficient solution of non-batched linear systems

Pratik Nayak, Hartwig Anzt and Fritz Göbel

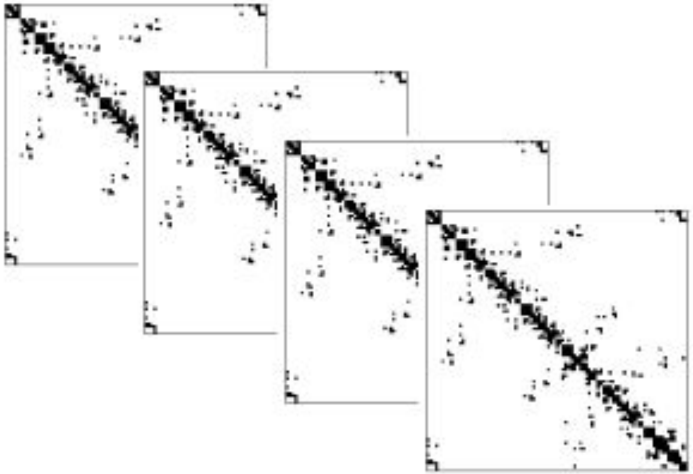
Parallel and Distributed Scientific and Engineering Computing Workshop, IPDPS 2023, St. Petersburg, Florida, USA

Outline

- Motivation
- Design philosophy and choices
- Implementation
- Applications and performance analysis

What are batched methods ?

- Batching: <u>Independent</u> computations that can be <u>scheduled in parallel</u>.
- Are highly suitable for GPUs and processors with many parallel computing units.
- Can maximize utilization of the GPU, due to excellent scalability.



What are batched methods ?

- Related work:
  - Usage in block-Jacobi preconditioners (Anzt. et.al PMAM 17)
  - Dense triangular solves on GPUs, DGETRF (Dong et.al 2014)
  - Tri-/Penta- diagonal solvers on GPUs (Carroll et.al 2021, Gloster et.al 2019, Valero-Lara et.al 2018)
  - Batched BLAS interface (Dongarra et.al 2016)

Ginkgo's batched interface: Design

Design philosophy:

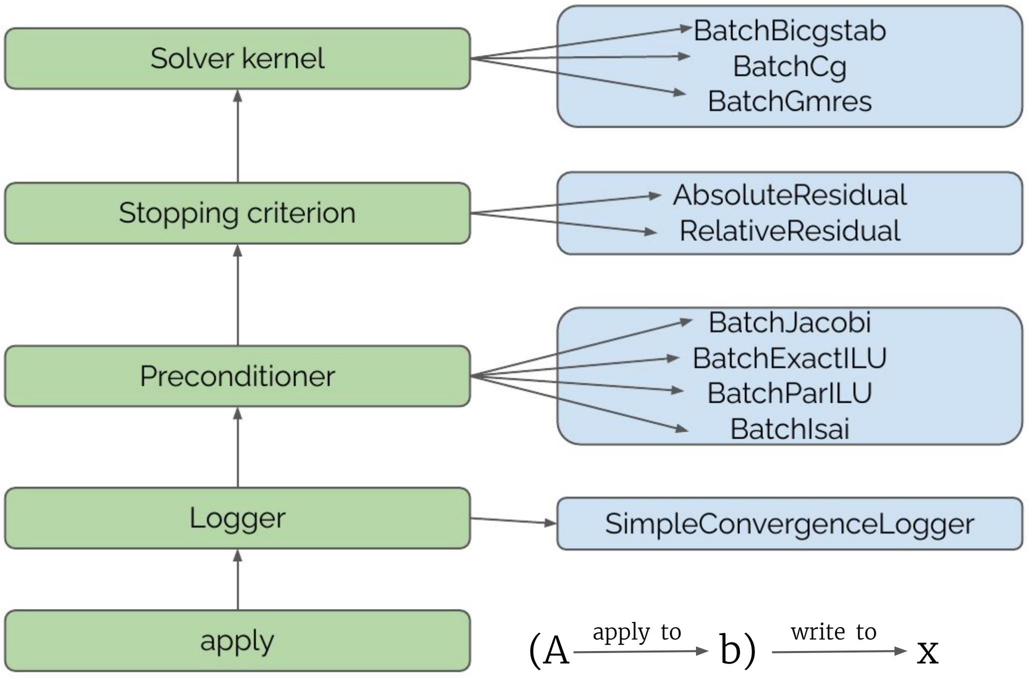
- Template the global solver apply kernel on logger, stopping criterion, matrix format and preconditioner type.
- Auto-configure shared memory based on problem size.
- Solve one linear system on one thread block.

Functionality:

- Sparse matrix formats: BatchCsr and BatchEll
- Iterative solvers: BatchBicgstab, BatchGmres, BatchCg, BatchIdr and BatchRichardson
- Preconditioners: BatchBlockJacobi, BatchILU, BatchISAI, BatchParILU

Multi-level dispatch mechanism

- Single device kernel call, but selection of different parameters through a multi-level dispatch.
- Allows for optimal use of caches and compute resources without launch overheads.



Automatic shared memory config

- Red objects: Intermediate vectors in SpMV: High priority
- Blue objects: Other vectors: Low priority
- Green objects: Constant matrices or vectors (In constant cache)

$$
\begin{array}{l}
\boldsymbol{r} \leftarrow \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}, \boldsymbol{z} \leftarrow \boldsymbol{M}\boldsymbol{r}, \boldsymbol{p} \leftarrow \boldsymbol{z}, \boldsymbol{t} \leftarrow \boldsymbol{0} \\
\rho \leftarrow \boldsymbol{r} \cdot \boldsymbol{z}, \alpha \leftarrow 1, \hat{\rho} \leftarrow 1 \\
\textbf{for } i < N_{iter} \textbf{ do} \\
\quad \textbf{if } |\rho| < \tau \textbf{ then} \\
\quad\quad \texttt{break} \\
\quad \textbf{end if} \\
\quad \boldsymbol{t} \leftarrow \boldsymbol{A}\boldsymbol{p} \\
\quad \alpha \leftarrow \frac{\rho}{\boldsymbol{p} \cdot \boldsymbol{t}} \\
\quad \boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha \boldsymbol{p} \\
\quad \boldsymbol{r} \leftarrow \boldsymbol{r} - \alpha \boldsymbol{t} \\
\quad \boldsymbol{z} \leftarrow \mathrm{PRECOND}(\boldsymbol{r}) \\
\quad \hat{\rho} \leftarrow \boldsymbol{r} \cdot \boldsymbol{z} \\
\quad \boldsymbol{p} \leftarrow \boldsymbol{z} + \frac{\hat{\rho}}{\rho} \cdot \boldsymbol{p} \\
\quad \rho \leftarrow \hat{\rho} \\
\textbf{end for}
\end{array}
$$

Ginkgo's batched v/s monolithic solvers

Batched

- Single kernel for solver apply
- Maximize utilization of shared memory across operations.
- Utilize one thread block for solve, judicious use of resources

Monolithic

- One kernel for each operation, SpMV, etc.
- Shared memory can be used within each operation only.
- Utilize full GPU

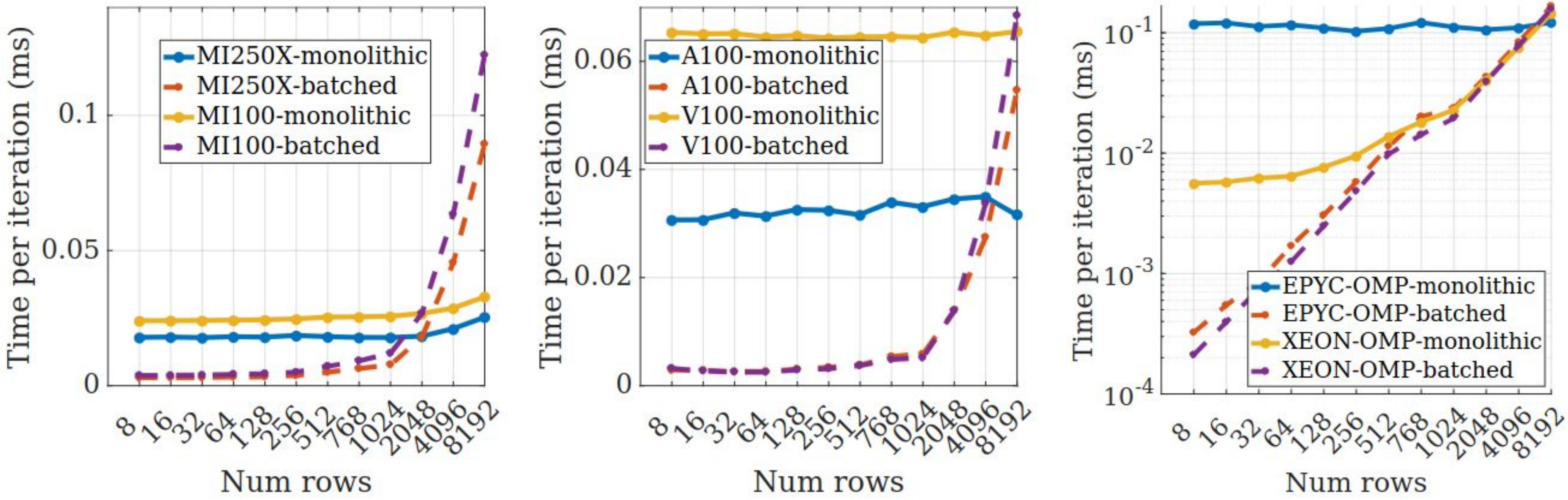
Hardware characteristics

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

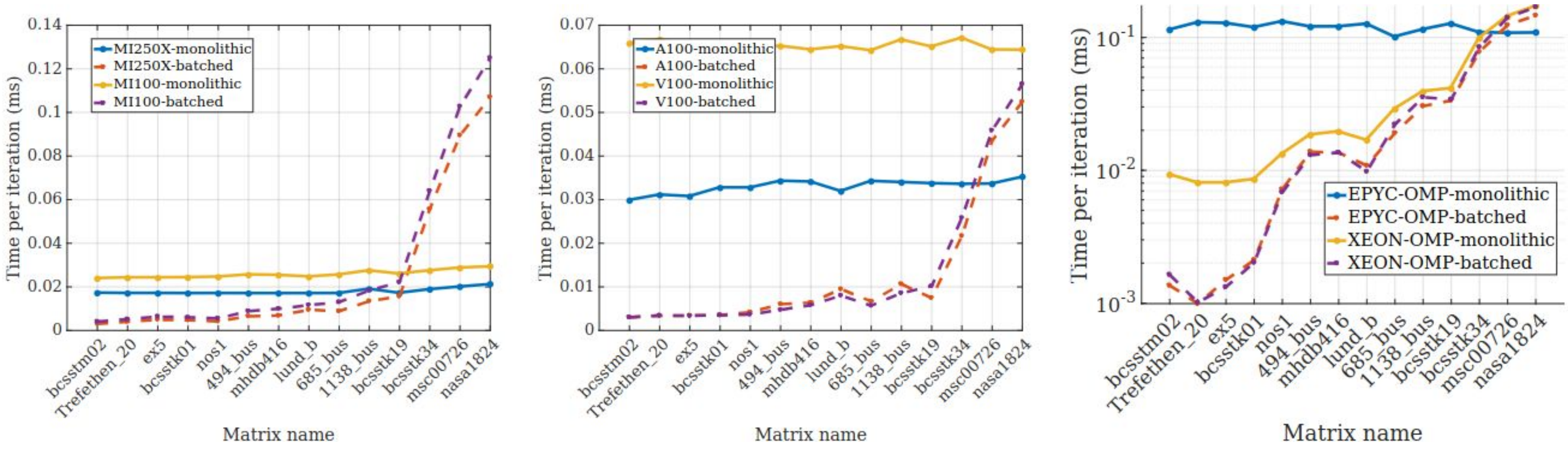
Experiments

- A 3-pt Laplacian problem for studying scaling behaviour.
- Wide variety of matrices from the Suitesparse matrix collection.
- On 2 different AMD architectures (with ROCm), 2 NVIDIA architectures(with CUDA), and 2 CPUs (with OpenMP).

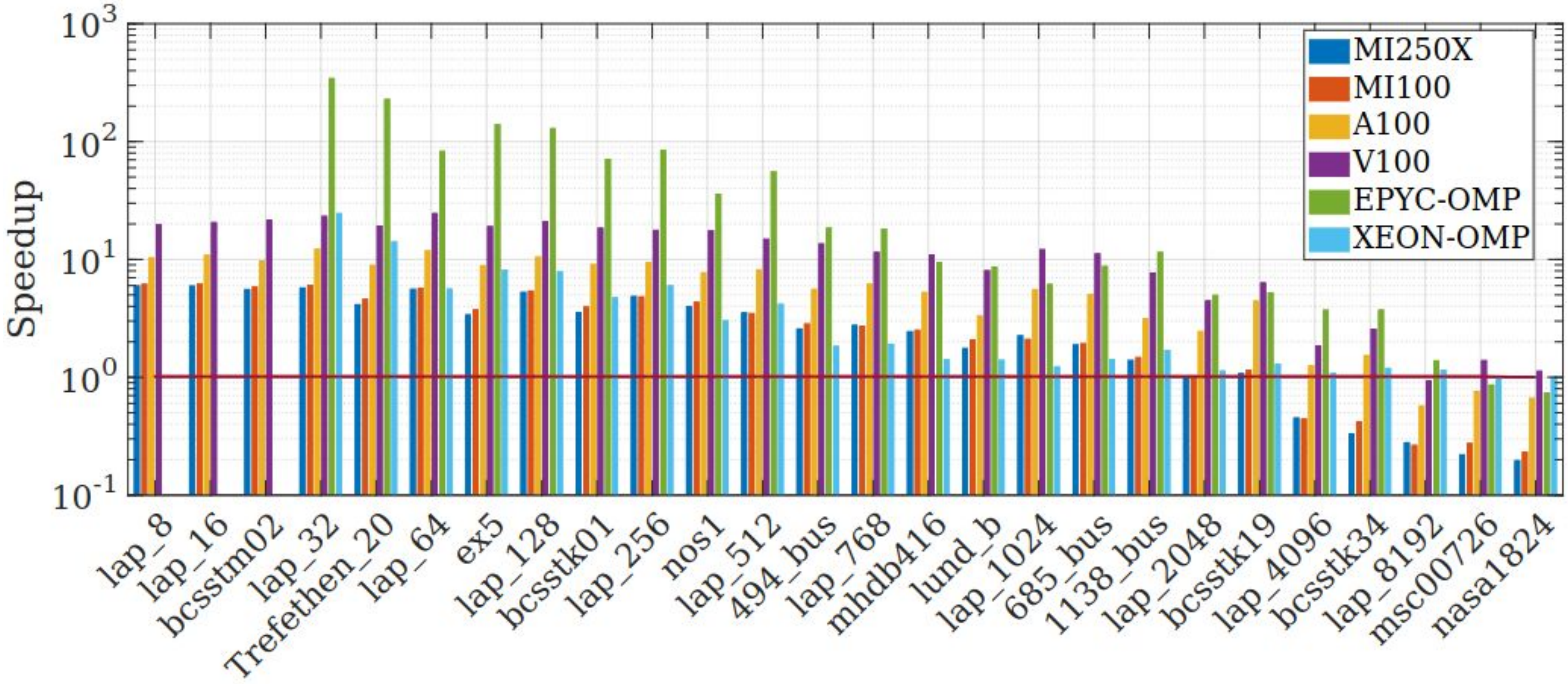
The Laplacian problem



Suitesparse matrices



Speedup



Speedup
10^3
10^2
10^1
10^0
10^{-1}
MI250X
MI100
A100
V100
EPYC-OMP
XEON-OMP
lap_8
lap_16
bcsstm02
lap_32
Trefethen_20
lap_64
ex5
lap_128
bcsstk01
lap_256
nos1
lap_512
494_bus
lap_768
mhdb416
lund_b
lap_1024
685_bus
1138_bus
lap_2048
bcsstk19
lap_4096
bcsstk34
lap_8192
msc00726
nasa1824

Speedup

- Batched solver with single level of parallelism can outperform monolithic solver for small problems.
- Speedups of around 10x, in particular very effective for large cache architectures such as CPUs.

Summary

- Batched solver ideas can greatly help improve efficiency of monolithic solvers.
    - Utilization of caches across distinct operations crucial.
- Judicious use of resources can benefit overall application, especially when combining batched solver ideas with streams and queues.

Acknowledgements

KIT | NHR
National High-Performance Computing Center

ECP
EXASCALE COMPUTING PROJECT

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES

Thank you!

Ginkgo

https://github.com/ginkgo-project/ginkgo

nayak@kit.edu