Two-stage Asynchronous Iterative Solvers for multi-GPU Clusters

Pratik Nayak
pratik.nayak@kit.edu

Terry Cojean
terry.cojean@kit.edu

Hartwig Anzt
hartwig.anzt@kit.edu

Steinbuch Center for Computing,
Karlsruhe Institute for Technology

12th November, 2020

HELMHOLTZ
RESEARCH FOR GRAND CHALLENGES

Karlsruhe Institute of Technology

EXASCALE COMPUTING PROJECT

Objectives

1. Our objectives:
   - To Study two-stage asynchronous iterative algorithms for Exascale.
   - In particular, study multi-GPU, multi-node problems.
   - And provide a framework to test these algorithms for a wide variety of problems and architectures.

Motivation

Begin of the Accelerator era

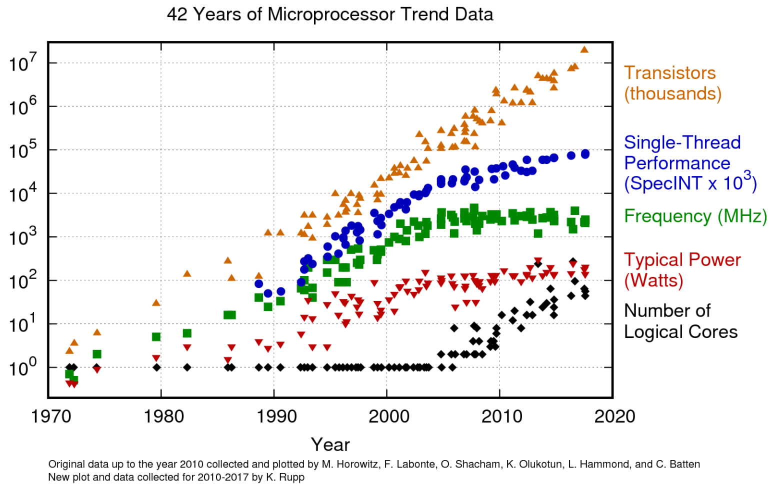
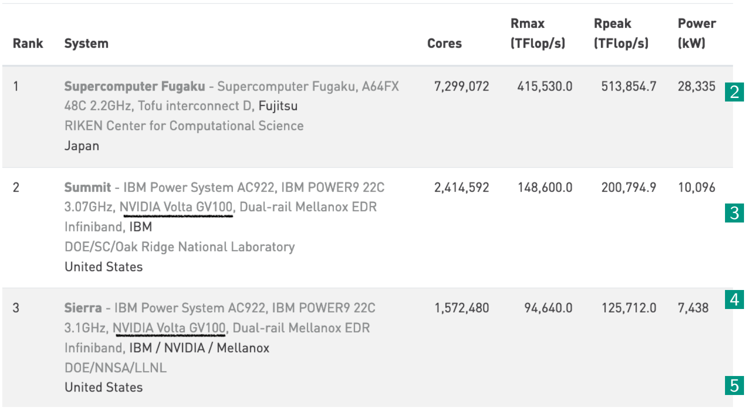


Figure: Computing trends

Current and future computing trends



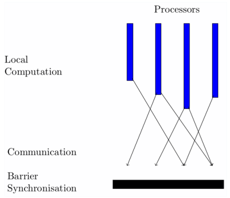
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,299,072	415,530.0	513,854.7	28,335
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438

Figure: Top 3 of the top 500

1. Frontier: AMD Epyc CPUs + AMD Instinct GPUs ($\gtrsim$1.5 EF, by 2021, First Exascale system)
2. El Capitan: AMD Epyc CPUs + AMD Instinct GPUs ($\gtrsim$2 EF, by 2023)
3. Aurora: Intel Xeon + Intel Xe GPUs (2021-2022)
4. Leonardo (EuroHPC): Intel CPU + NVIDIA A100 GPUs ($\approx$200 PF)
5. LUMI (EuroHPC): AMD Epyc CPUs + AMD Instinct GPUs ($\approx$550 PF, by Q4 2021)

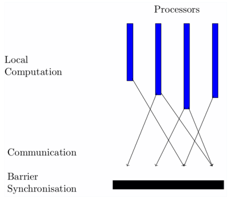
Synchronous v/s Asynchronous models

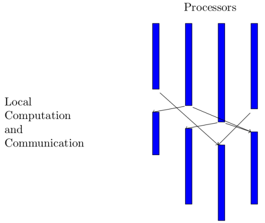
1. Bulk synchronous model of parallel execution (Most algorithms today).
   - A known task graph.
   - Needs regular synchronization between processes.
   - Not feasible for large number of processes.



Synchronous v/s Asynchronous models

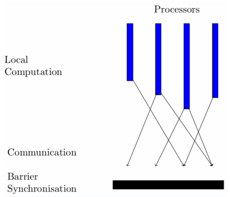
1. Bulk synchronous model of parallel execution (Most algorithms today).
   - A known task graph.
   - Needs regular synchronization between processes.
   - Not feasible for large number of processes.
2. Asynchronous model of execution (Where algorithms need to be).
   - Ideally, no synchronization.
   - Feasible and possibly necessary for large number of processes.
   - Possibly unknown task graph.

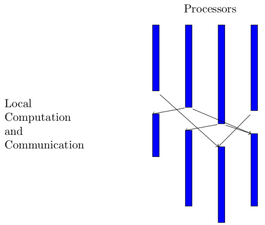




Synchronous v/s Asynchronous models

1. Bulk synchronous model of parallel execution (Most algorithms today).
   - A known task graph.
   - Needs regular synchronization between processes.
   - Not feasible for large number of processes.
2. Asynchronous model of execution (Where algorithms need to be).
   - Ideally, no synchronization.
   - Feasible and possibly necessary for large number of processes.
   - Possibly unknown task graph.
3. Partial Synchronization model (A compromise, future work).
   - Regular synchronization (relaxed) between processes.
   - Might work for large number of processes.
   - Partially known task graph.





Background

Problem Formulation.

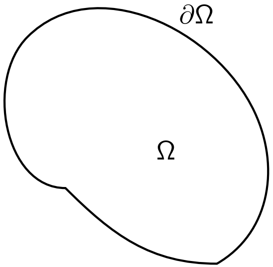


Figure: Generic Domain

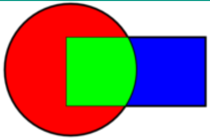
Problem:

$$\mathcal{L}x = f \ in \ \Omega; \qquad \mathcal{B}x = g \ on \ \partial\Omega$$

Linear system:

$$Ax = f$$

Schwarz methods



1. Initially used to prove convergence of the Poisson problem for general domains (Schwarz, 1870). Alternating method. Slow convergence.
2. Solve each subset(subdomain) independently and communicate between each "iteration".
3. Gained popularity with parallel computers.

Source: ddm.org

Restricted Additive Schwarz methods

An improvement of the parallel version of the Schwarz method for faster convergence.

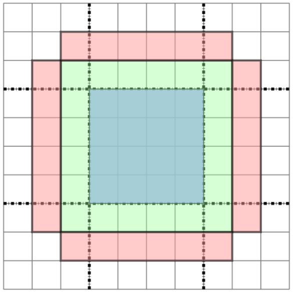
Group unknowns into subsets:

$$x_j = \tilde{R}_j x, \;\; j = 1, \dots, N$$

$\tilde{R}_j$ is the rectangular Restriction matrices which corresponds to a non-overlapping decomposition.

Used widely as a preconditioner:

$$M_{RAS}^{-1} = \sum_{j}^{N} \tilde{R}_j^T A_j^{-1} R_j$$



Restricted Additive Schwarz

Compute using the full overlapped sub-matrix, but update only your locally associated values.

Source: Schematic inspired by Yamazaki et.al, 2019, doi: 10.1016/j.parco.2019.05.004

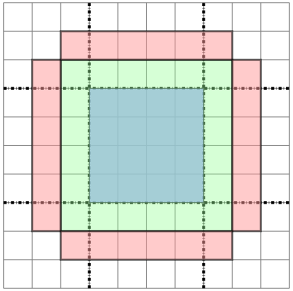
Restricted Additive Schwarz methods

RAS:

$$x_p^{k+1} = x_p^k + \sum_j^N \tilde{R}_p(R_j f - (R_j A R_j^T)^{-1} R_j x^k)$$

Advantages:

1. Saves communication compared to Additive Schwarz.
2. Reduced subdomain update count compared to Additive Schwarz.
3. Collision free parallel implementation, free from weightings.



Source: Efstathiou and Gander, 2003, doi: 10.1023/B:BITN.0000014563.33622.1d

Asynchronous iterative methods

Asynchronous iteration:

$$\begin{cases} x^{(s)}(0) = x_0^{(s)} \\ x^{(s)}(n+1) = \begin{cases} \mathcal{R}^{(s)}\left(x^{(1)}(\tau_1^{(s)}(n)), \dots, x^{(p)}(\tau_p^{(s)}(n))\right) & \text{if } s \in \sigma(n) \\ x^{(s)}(n) \text{ if } s \notin \sigma(n) \end{cases} \end{cases} \tag{1}$$

where, $s = 1, \dots, p$ and p is the number of subdomains.

$\tau_j^{(s)}(n)$ is the delay function that represents the subdomain update number of the data from subdomain j available at s at subdomain update number n.

$\sigma(n)$ is the set of subdomains which update at subdomain update number n.

Necessary conditions for convergence

Necessary conditions (not necessarily sufficient):

1. $$\forall s, j \in 1, ..., p, \forall n \in \mathbb{N}*, \tau_j^{(s)}(n) \leq n \tag{2}$$

   States that the delay function cannot return future iterations.

2. $$\forall s \in 1, ..., p, \#\{n \in \mathbb{N} | s \in \sigma(n)\} = +\infty \tag{3}$$

   States that no subdomain can stop updating its neighbors.

3. $$\forall s, j \in 1, ..., p, \lim_{n \to \infty} \tau_j^{(s)}(n)\} = +\infty \tag{4}$$

   States that new data will eventually always be provided to the subdomain.

Implementation and Experimentation

The RAS iterative solver.

Algorithm 1 RAS Iterative solver

```
1: procedure ITERATIVE SOLUTION($A, x, b$)
2:     procedure INITIALIZATION
3:         Partition matrix                                      ▷ objective based
4:         Distribute data
5:         Initialize data
6:     procedure SOLVE
7:         while ($subd\_update\_count < max\_subd\_update\_count$ or until convergence) do
8:             Locally solve the matrix                          ▷ Iterative / direct
9:             Exchange boundary information
10:            Update boundary information
11:            Check for Convergence                             ▷ Decentralized
12:        Gather the final solution vector and post-processing
```

Experimentation Parameters

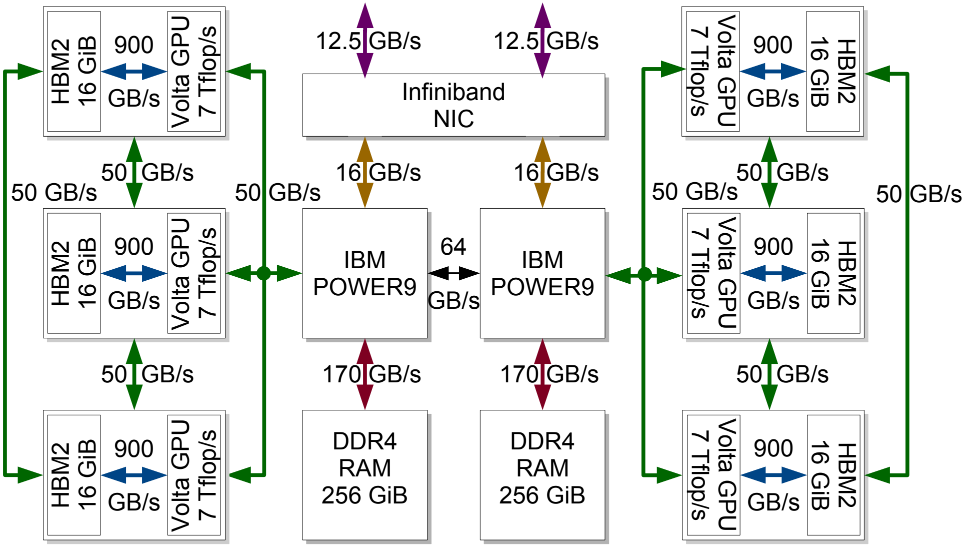
1. Everything implemented with Ginkgo.
2. Experiments performed on Summit, ORNL.
    1. 6 GPU's per node, NVIDIA Tesla V100's.
3. RDMA communication with MPI-onesided functions (IBM Spectrum MPI with CUDA Aware).
4. Partitioning with METIS.
5. Global convergence detection is decentralized (leader election based) (Bahi et.al, 2005).
6. Test problems:
    1. 3D Laplacian problem with 2nd order basis functions (deal.ii, example 6) (PCG)
    2. 2D Advection problem with 5th order basis functions (deal.ii, example 9) (PGMRES)

Source: Bahi et.al, doi: 10.1109/TPDS.2005.2 17 / 25

Experimentation Parameters

1. Global relative residual reduction goal: 1e-12
2. Local subdomain iteration count varies from 30 to default (def, a local relative residual reduction goal of 1e-6)
3. Experiments run on upto 36 GPUs.
4. We compute final true residual norm ($||r||_2 = ||b - Ax||_2$) to verify correctness along with comparing the solutions by visualization (for a token problem size)

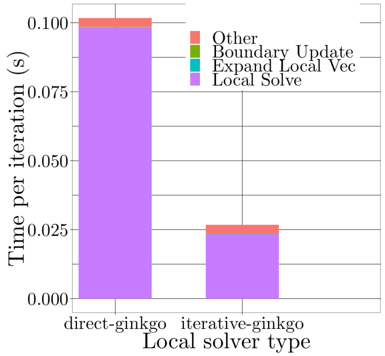
System configuration: Summit, ORNL



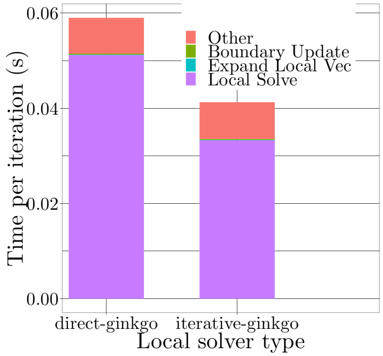
Source: Kahle et.al, doi: 10.1109/ ISSCC.2019.8662426

Results

Comparison with local direct solver - Time per subdomain update



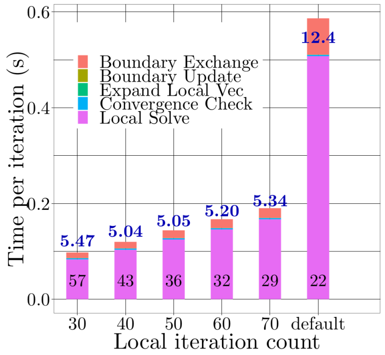
(a) Laplacian problem



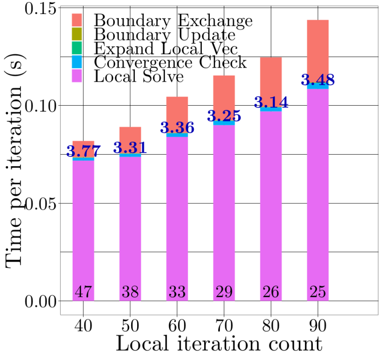
(b) Advection problem

1. Communication time similar between direct and iterative solvers.
2. Local solve time is significantly lesser.
3. Advection problem (with preconditioned GMRES) is harder to solve.

Controlling the local iteration criterion - Synchronous



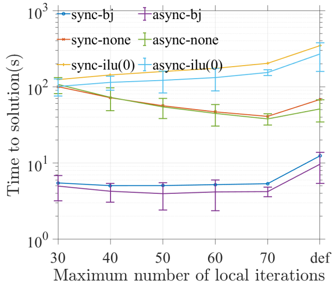
(a) Laplacian problem



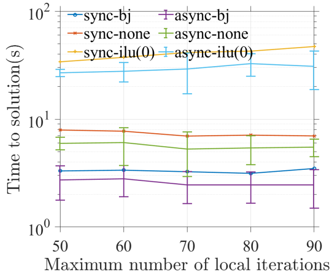
(b) Advection problem

1. Optimal local iteration count depends on problem size.
2. Number of subdomain updates decreases with increasing local max-iter counts.
3. Local solver cost increases with increasing local max-iter counts.

Preconditioner effects and comparisons with synchronous



(a) Laplacian problem



(b) Advection problem

1. Asynchronous is better than synchronous in almost all cases.
2. Block-jacobi preconditioner seems to be the most efficient.
3. ILU(0) seems to be too expensive compared to its effectiveness.

Outlook

Summary and Future work

Summary

- Asynchronous methods can improve the overall time to solution.
- Two stage methods can be efficient, particularly in conjunction with asynchronous communication.
- Balancing local max-iteration count and global subdomain update count is essential with multi-stage methods.

Future Work

- Estimating error and convergence bounds for these multi-stage asynchronous methods.
- Extension to multi-level Schwarz methods.