PSCTOOLKIT: Parallel Sparse Computation Toolkit

Math 2 Product (M2P): Emerging Technologies in Computational Science for Industry, Sustainability and Innovation

F. Durastante[♠,♡]

([♠]Università d Pisa

[♡]Consiglio Nazionale delle Ricerche)

ISo8b – Mathematical software for Computational and Data Science at Extreme scales

May 31, 2023

Dipartimento di Matematica
Università di Pisa

Collaborators & Funding

1 With a Little Help from My Friends

Pasqua D'Ambra,
Consiglio Nazionale delle Ricerche
Istituto per le Applicazioni del Calcolo
"M. Picone"

Salvatore Filippone,
Università degli Studi di Roma "Tor Vergata"
Dipartimento di Ingegneria Civile e
Ingegneria Informatica
IAC-CNR

Leonardo **E**arly **A**ccess **P**rogram (LEAP). Computing time on the *Leonardo HPC System* for the project "PSCToolkit for Sparse Matrix Computations at Extreme Scales."

This work has been supported by the **Spoke 1** "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 Componente 2 Investimento 1.4: Potenziamento strutture di ricerca e creazione di "campioni nazionali di R&S (M4C2-19)", and the **Spoke 6** "Multiscale modelling & Engineering applications" - Next Generation EU (NGEU).

Table of Contents

2 The Model Problem

What we want to solve

$$\text{Solve : } A\mathbf{x} = \mathbf{b},$$

where

- $A \in \mathbb{R}^{n \times n}$ is a very large and sparse matrix $\mathrm{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$,

is often the most time consuming computational kernel in many areas of computational science and engineering problems,

What we want to solve

$$\text{Solve : } A\mathbf{x} = \mathbf{b},$$

where

- $A \in \mathbb{R}^{n \times n}$ is a very large and sparse matrix $\mathrm{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.

is often the most time consuming computational kernel in many areas of computational science and engineering problems,

the exascale challenge, using computer that perform 10^{15} Flops, targeting next-gen systems performing 10^{18} Flops to solve problems with tens of billions of unknowns.

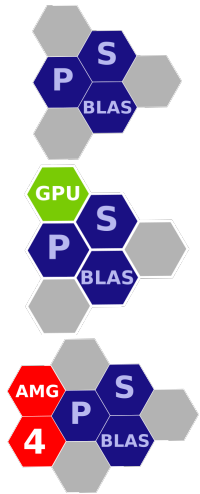
Table of Contents

3 The Parallel Sparse Computation Toolkit

Parallel Sparse Computation Toolkit – psctoolkit.github.io

Two central libraries PSBLAS and AMG4PSBLAS:

- Existing software standards:
  - MPI, OpenMP, CUDA
  - Serial sparse BLAS,
  - (Par)Metis,
  - AMD
- Attention to performance using modern Fortran;
- Research on new preconditioners;
- No need to delve in the data structures for the user;
- Tools for error and mesh handling beyond simple algebraic operations;
- Standard Krylov solvers



Parallel Sparse Computation Toolkit – psctoolkit.github.io

Two central libraries PSBLAS and AMG4PSBLAS:

- Domain decomposition preconditioners
- Algebraic multigrid with aggregation schemes
  - Parallel coupled weighted **matching based aggregation**[1,2]
  - Parallel decoupled smoothed aggregation (**V**aněk, **B**rezina, **M**andel)
- Parallel Smoothers (Block-Jacobi, DD-Schwartz, Hybrid-GS/SGS/FBGS, ℓ_1 variants) that can be coupled with specialized block (approximate) solvers MUMPS, SuperLU, incomplete factorizations (AINV, INVK/L, ILU-type)
- V-Cycle, W-Cycle, K-Cycle

1 P. D'Ambra, S. Filippone and P. S. Vassilevski, BootCMatch: a software package for bootstrap AMG based on graph weighted matching, ACM Trans. Math. Software **44** (2018), no. 4, Art. 39, 25 pp.

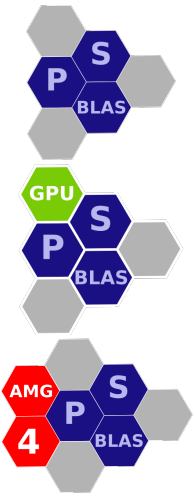
2 P. D'Ambra, F. D. and S. Filippone, AMG preconditioners for linear solvers towards extreme scale, SIAM J. Sci. Comput. **43** (2021), no. 5, S679–S703.

Parallel Sparse Computation Toolkit – psctoolkit.github.io

Two central libraries PSBLAS and AMG4PSBLAS.

- Freely available from: `https://psctoolkit.github.io`,
- Open Source with BSD 3 Clause License.
- P. D'Ambra, F. D., and S. Filippone, Parallel Sparse Computation Toolkit, Software Impacts (2023): 100463.

```
git clone git@github.com:psctoolkit/psctoolkit.git
(cd psblas3; ./configure; make -j; make install)
(cd psblas3-ext; ./configure; make -j; make install)
(cd amg4psblas; ./configure; make -j; make install)
```



Algebraic Multigrid Preconditioners

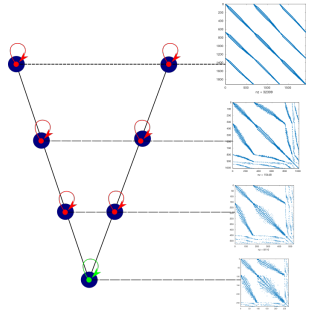
3 The Parallel Sparse Computation Toolkit

Given Matrix $A \in \mathbb{R}^{n \times n}$ SPD

Wanted Iterative method B to precondition the CG method:

- Hierarchy of systems
$$A_l \mathbf{x}_= \mathbf{b}_l, l = 0, \ldots, n_{\text{lev}}$$
- Transfer operators:
$$P_{l+1}^l : \mathbb{R}^{n_{l+1}} \to \mathbb{R}^{n_l}$$

Missing Structural/geometric infos



Smoother: “High frequencies”

$$M_l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$$

Prolongator: “Low frequencies”

$$P_{l+1}^l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_{l+1}}$$

Complementarity of Smoother and Prolongator

What are we looking for?

Solve the preconditioned system:

$$B^{-1}Ax = B^{-1}b,$$

with matrix $B^{-1} \approx A^{-1}$ (left preconditioner) such that:

Algorithmic scalability $\max_i \lambda_i(B^{-1}A) \approx 1$ being independent of n,

Linear complexity the action of B^{-1} costs as little as possible, the best being $\mathcal{O}(n)$ flops,

Implementation scalability in a massively parallel computer, B^{-1} should be **composed of local actions**, performance should depend linearly on the number of computing units employed (MPI Tasks, OpenMP Threads, GPUs).

What is our *recipe*?

- The smoother M is a standard iterative solver with good parallel properties, e.g., ℓ_1–Jacobi, Hybrid-FBGS, Hybrid-SGS, CG method, etc.
- The prolongator P is built by dofs *aggregation based on matching* in the weighted (adjacency) graph of A.
- The coarse solver can be (again) a preconditioned CG method.

What is our *recipe*?

3 The Parallel Sparse Computation Toolkit

- The smoother M is an iterative solver with good parallel properties:

  GS $A = M - N$, with $M = L + D$ and $N = -L^T$, where $D = \mathrm{diag}(A)$ and $L = \mathrm{tril}(A)$ is intrinsically sequential!

  HGS Inexact block-Jacobi version of GS, in the portion of the row-block local to each process the method acts as the GS method.

  ℓ_1-HGS On process $p = 1, \ldots, np$ relative to the index set Ω_p we factorize $A_{pp} = L_{pp} + D_{pp} + L_{pp}^T$ for $D_{pp} = \mathrm{diag}(A_{pp})$ and $L_{pp} = \mathrm{tril}(A_{pp})$ then:

$$
\begin{aligned}
M_{\ell_1-HGS} &= \mathrm{diag}((M_{\ell_1-HGS})_p)_{p=1,\ldots np}, \\
(M_{\ell_1-HGS})_p &= L_{pp} + D_{pp} + D_{\ell_1 p}, \\
(d_{\ell_1})_{i=1}^{nb} &= \sum_{j \in \Omega_p^{nb}} |a_{ij}|.
\end{aligned}
\qquad
M_{\ell_1-HGS} = \mathrm{diag}((M_{\ell_1-HGS})_p)_{p=1,\ldots np},
$$

  AINV Block-Jacobi with an approximate inverse factorization on the block $\Rightarrow$ suitable for GPUs

What is our *recipe*?

- The prolongator P is built by dofs *aggregation based on matching* in the weighted (adjacency) graph of A.

Given $\mathbf{w} \in \mathbb{R}^n$, let $P \in \mathbb{R}^{n \times n_c}$ and $P_f \in \mathbb{R}^{n \times n_f}$ be a prolongator and a complementary prolongator, such that:

$$\mathbb{R}^n = \mathrm{Range}(P) \oplus^{\perp} \mathrm{Range}(P_f), \quad n = n_c + n_f$$

$\mathbf{w} \in \mathrm{Range}(P)$: coarse space $\qquad$ $\mathrm{Range}(P_f)$: complementary space

$$[P, P_f]^T A [P, P_f] = \begin{pmatrix} P^T A P & P^T A P_f \\ P_f^T A P & P_f^T A P_f \end{pmatrix} = \begin{pmatrix} A_c & A_{cf} \\ A_{fc} & A_f \end{pmatrix}$$

A_c: coarse matrix $\qquad$ A_f: hierarchical complement

Sufficient condition for efficient coarsening

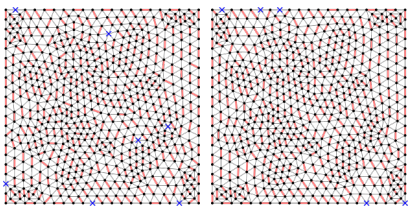
$A_f = P_f^T A P_f$ as well conditioned as possible, i.e.,

Convergence rate of *compatible relaxation*: $\rho_f = \|I - M_f^{-1} A_f\|_{A_f} \ll 1$

Parallel Matching Algorithms

3 The Parallel Sparse Computation Toolkit

1. What is the best matching algorithm from a computational point of view?
2. Can we do an **approximate global matching** over the whole graph for better aggregation quality?



Algorithm: Locally Dominant Edge

Input: Graph $G = (\mathcal{V}, \mathcal{E})$, Weights $\hat{A}$

$\mathcal{M} \leftarrow \emptyset$;

while $\mathcal{E} \neq \emptyset$ **do**

  Take a locally dominant edge $(i,j) \in \mathcal{E}$, i.e., such that

$$\arg\max_k \hat{a}_{ik} = \arg\max_k \hat{a}_{jk} = \hat{a}_{ij}$$

  Add $(i,j) \in \mathcal{M}$;

  Remove all edges incident to i and j from $\mathcal{E}$;

end

Output: Matching $\mathcal{M}$

Ü. V. Çatalyürek, F. Dobrian, A. Gebremedhin, M. Halappanavar and A. Pothen, Distributed-Memory Parallel Algorithms for Matching and Coloring, 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, Anchorage, AK, USA, 2011, pp. 1971-1980, doi: 10.1109/IPDPS.2011.360.

Table of Contents

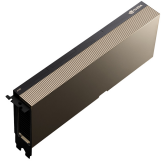
4 Pre-Exascale Results

The supercomputer of the TOP500: www.top500.org

4 Pre-Exascale Results

	System[1]	Cores	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	22,703
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	7,404
5	**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	10,096

Leonardo



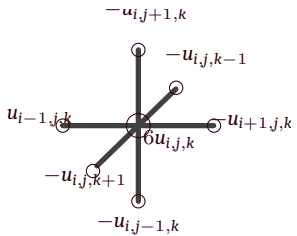
NVIDIA A100

[1] Top500: June 2023 List.

3D Poisson Problem

Finite **D**ifferences discretization of

$$\begin{cases} -\nabla^2 u = 1, & \mathbf{x} \in [0,1]^3 \\ u(\mathbf{x}) = 0, & \mathbf{x} \in \partial[0,1]^3. \end{cases}$$



Data distribution:

- For PSCToolkit we use a block 3D Distribution,
- For AMGX we use the `amgx_mpi_poisson7` tester.

Solver is **F**lexible **C**onjugate **G**radient and **CG** for PSCToolkit and AMGX respectively, tolerance 10^{-6}.

Weak Scaling

4 Pre-Exascale Results

- In **weak scaling**, both the **number of computing units** and the **problem size** are **increased**: *constant workload per computing unit*.
- We use 8×10^6 unknowns per GPU, *i.e.*, 3.2×10^7 unknowns per node.

We use the following resources:

- Number of GPUs from 1 to 8192,
- GPUs x Node 4 (1 MPI Task x GPU, 8 CPUs per Task)
- Pure MPI: 32 MPI Tasks per Node

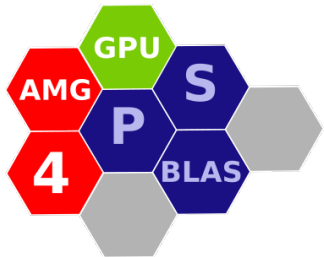
Within the software framework:

- Compilers: `gcc/11.3.0`
- MPI: `openmpi/4.1.4`
- CUDA `compilation tools, release 11.8, V11.8.89`

Algorithms

4 Pre-Exascale Results

- **Aggregation:** VBM, **Cycle:** V, **Smoother:** ℓ_1-Jacobi, **Coarse Solver:** PCG + ℓ_1-Jacobi,
- **Aggregation:** Smoothed Matching, **Cycle:** V, **Smoother:** ℓ_1-Jacobi, **Coarse Solver:** PCG + ℓ_1-Jacobi,
- **Aggregation:** Matching, **Cycle:** Variable V, **Smoother:** ℓ_1-Jacobi, **Coarse Solver:** PCG + ℓ_1-Jacobi,
- **Coarsening:** Classical Algebraic Multigrid, **Cycle:** V, **Smoother:** ℓ_1-Jacobi, **Coarse Solver**: ℓ_1-Jacobi, 40 sweeps
- **Aggregation:** (Iterative) Parallel Graph Matching, **Cycle:** V, **Smoother:** ℓ_1-Jacobi, **Coarse Solver**: ℓ_1-Jacobi, 40 sweeps



NVIDIA/**AMGX**

Distributed multigrid linear solver library on GPU

NVIDIA

Operator Complexity

A first measure of the **theoretical computational cost** and of the **memory footprint** of the different algorithms is given by the **operator complexity**:

$$\text{opc} = \frac{\sum_{l=0}^{n_{\text{lev}}} \text{nnz}(A_l)}{\text{nnz}(A)} =$$

"the total number of nonzeroes in the linear operators on all grids divided by the number of nonzeroes in the fine grid operator"

Computing Units	VBM	Matching Smoothed	Matching Unsmoothed	AMGX Classical	AMGX Matching
1	1,575	1,894	1,142	4,45456	1,27979
2	1,578	1,905	1,142	4,43576	1,31187
4	1,58	1,915	1,143	4,51377	1,33117
8	1,583	1,917	1,142	4,52376	1,33162
16	1,584	1,925	1,143	4,51239	1,32133

Operator Complexity

A first measure of the **theoretical computational cost** and of the **memory footprint** of the different algorithms is given by the **operator complexity**:

$$\text{opc} = \frac{\sum_{l=0}^{n_{\text{lev}}} \text{nnz}(A_l)}{\text{nnz}(A)} =$$

"the total number of nonzeroes in the linear operators on all grids divided by the number of nonzeroes in the fine grid operator"

Computing Units	VBM	Matching Smoothed	Matching Unsmoothed	AMGX Classical	AMGX Matching
32	1,584	1,93	1,143	4,49595	1,31887
64	1,587	1,93	1,143	4,50135	1,31914
128	1,588	1,936	1,143	4,49925	1,31421
256	1,587	1,905	1,144	4,49252	1,31314
512	1,589	1,937	1,143	4,4952	1,31329
1024	1,588	1,942	1,144	4,49503	1,31091

Operator Complexity

A first measure of the **theoretical computational cost** and of the **memory footprint** of the different algorithms is given by the **operator complexity**:

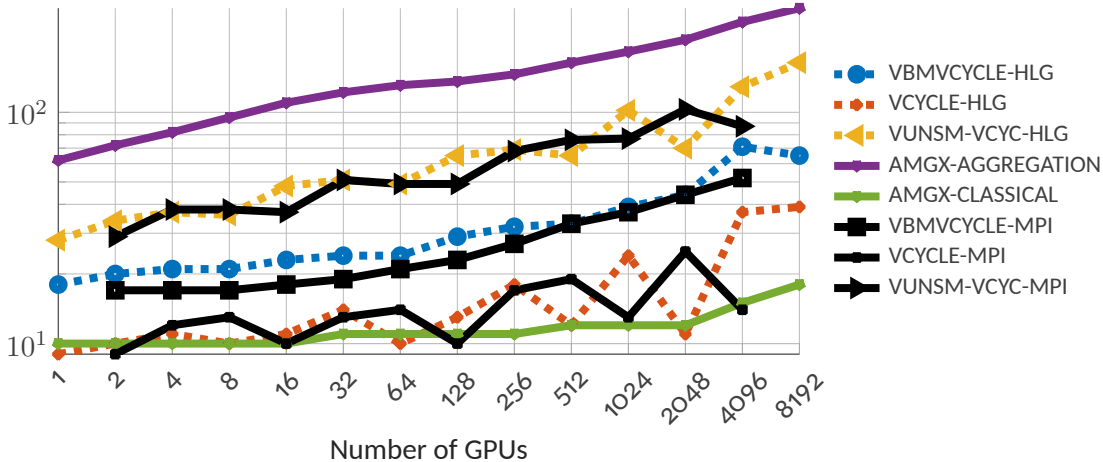
$$\mathrm{opc} = \frac{\sum_{l=0}^{n_{\mathrm{lev}}} \mathrm{nnz}(A_l)}{\mathrm{nnz}(A)} =$$

"the total number of nonzeroes in the linear operators on all grids divided by the number of nonzeroes in the fine grid operator"

Computing Units	VBM	Matching Smoothed	Matching Unsmoothed	AMGX Classical	AMGX Matching
2048	1,59	1,939	1,143	4,4921	1,31041
4096	1,588	1,906	1,144	4,49354	1,31049
8192			1,144	4,49371	1,30932

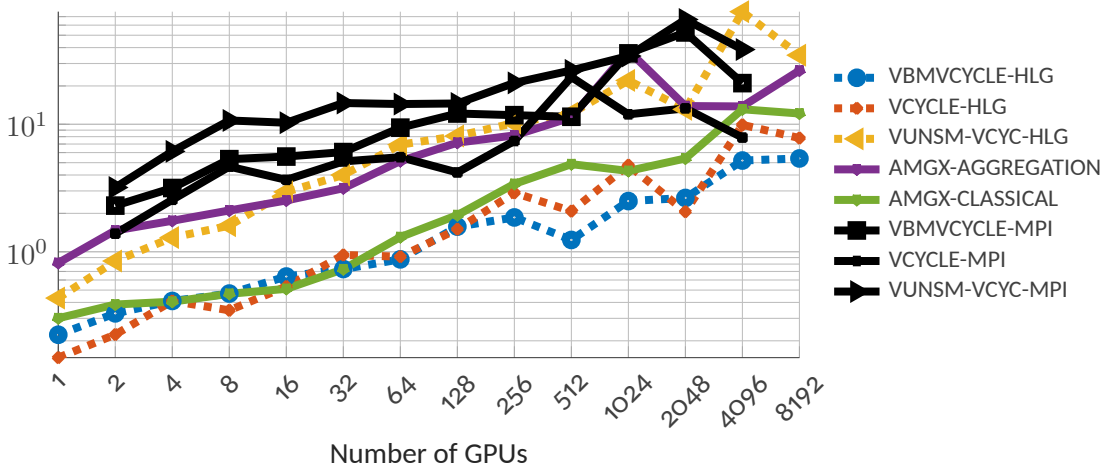
Algorithmic Scalability: Iteration Count

4 Pre-Exascale Results



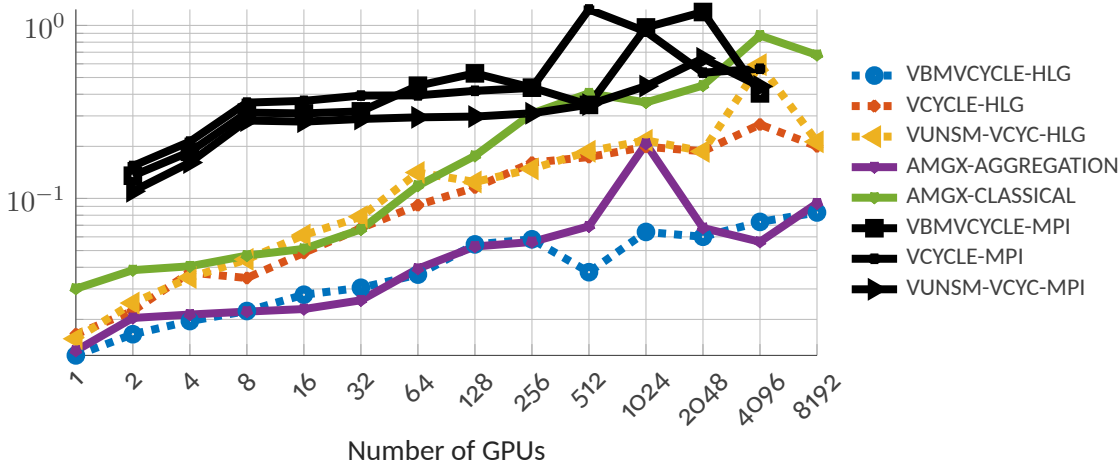
Implementation Scalability: Solve Time (s)

4 Pre-Exascale Results



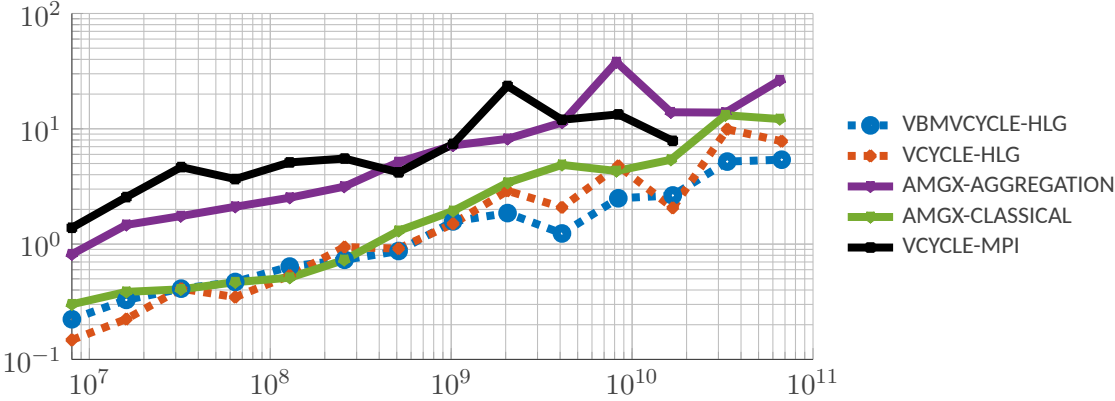
Implementation Scalability: Time × Iteration (s)

4 Pre-Exascale Results



Best Solve Time vs Global System Size

4 Pre-Exascale Results



Largest System Size is: $67121414144 \approx 7 \times 10^{10}$.

Plans for the Future

☹ Many **graph algorithms are inherently serial in nature**, and therefore require nontrivial algorithmic techniques for **creating concurrency**:

$$\textbf{Asymptotic cost for } G = (V, E) \text{ is } O(|E|\Delta), \qquad \Delta = \max_{v_i \in V} \deg(v_i).$$

Plans for the Future

4 Pre-Exascale Results

☹ Many **graph algorithms are inherently serial in nature**, and therefore require nontrivial algorithmic techniques for **creating concurrency**:

$$\textbf{Asymptotic cost} \text{ for } G = (V, E) \text{ is } O(|E|\Delta), \qquad \Delta = \max_{v_i \in V} \deg(v_i).$$

— If we **build the matching using MPI** we have 4 tasks per node and no OpenMP acceleration, weak scaling building times for the hierarchy goes from 18s (1 GPU) to 293s (8192 GPUs): too much!

Plans for the Future

4 Pre-Exascale Results

☹ Many **graph algorithms are inherently serial in nature**, and therefore require nontrivial algorithmic techniques for **creating concurrency**:

$$\textbf{Asymptotic cost for } G = (V, E) \text{ is } O(|E|\Delta), \qquad \Delta = \max_{v_i \in V} \deg(v_i).$$

- If we **build the matching using MPI** we have 4 tasks per node and no OpenMP acceleration, weak scaling building times for the hierarchy goes from 18s (1 GPU) to 293s (8192 GPUs): too much!
- There is an **undergoing effort** for moving it to the **GPUs**[1] and we are planning to *include it in PSCToolkit*,

[1] Bernaschi, M., P. D'Ambra, and D. Pasquini. "BootCMatchG: An adaptive algebraic multigrid linear solver for GPUs." Software Impacts 6 (2020): 100041.

Plans for the Future

4 Pre-Exascale Results

☹ Many **graph algorithms are inherently serial in nature**, and therefore require nontrivial algorithmic techniques for **creating concurrency**:

$$\textbf{Asymptotic cost for } G = (V, E) \text{ is } O(|E|\Delta), \qquad \Delta = \max_{v_i \in V} \deg(v_i).$$

- If we **build the matching using MPI** we have 4 tasks per node and no OpenMP acceleration, weak scaling building times for the hierarchy goes from 18s (1 GPU) to 293s (8192 GPUs): too much!
- There is an **undergoing effort** for moving it to the **GPUs**[1] and we are planning to *include it in PSCToolkit*,
- We are also investigating **alternative algorithmic approaches** and the possibility of a **multithreaded version**.

Plans for the Future

4 Pre-Exascale Results

- ☹ Many **graph algorithms are inherently serial in nature**, and therefore require nontrivial algorithmic techniques for **creating concurrency**:

$$\textbf{Asymptotic cost for } G = (V, E) \text{ is } O(|E|\Delta), \qquad \Delta = \max_{v_i \in V} \deg(v_i).$$

  - If we **build the matching using MPI** we have 4 tasks per node and no OpenMP acceleration, weak scaling building times for the hierarchy goes from 18s (1 GPU) to 293s (8192 GPUs): too much!
  - There is an **undergoing effort** for moving it to the **GPUs**[1] and we are planning to *include it in PSCToolkit*,
  - We are also investigating **alternative algorithmic approaches** and the possibility of a **multithreaded version**.

- 👍 Improved **OpenMP support**, for now we have added **matrix assembly routines** and **some BLAS**.

PSCTOOLKIT: Parallel Sparse Computation Toolkit

Thank you for listening!
Any questions?