SparseLinSol: Library for Solving SLAEs Based on Multigrid Methods

B. Krasnopolsky[1]

krasnopolsky@imec.msu.ru

A. Medvedev[2]

alexey.v.medvedev@gmail.com

[1]*Institute of Mechanics, Moscow State University*

[2]*JSC T-Services*

CFD-Weekend, 28 November 2015, Moscow, Russia

Overview

1. Motivation
2. SparseLinSol library
3. OpenFOAM simulations speedup
4. Conclusion
5. Future plans

1. Motivation

OpenFOAM Package

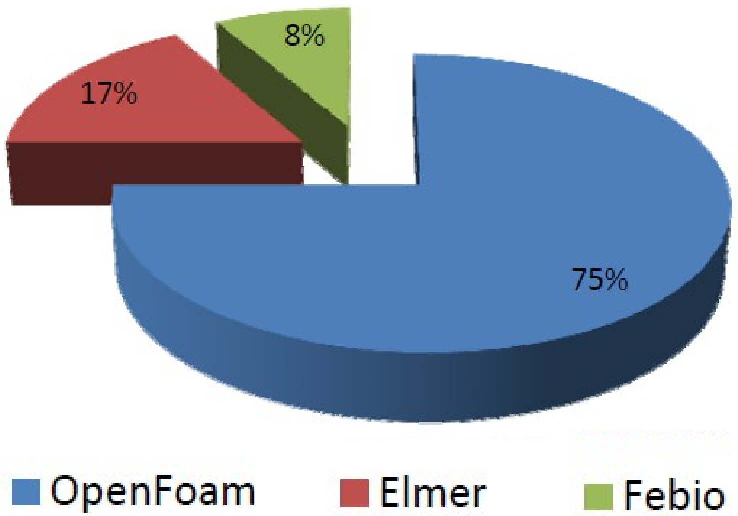
- ***One of the most popular open-source engineering packages***
- ***Main purpose — computational fluid dynamics and conjugate heat transfer problems modeling***
  - Finite volume method
  - Various computational grids and numerical schemes
  - A wide range of turbulence models
  - Moving meshes
  - Separate solver for every mathematical model
- ***Users can modify existing solvers or write a new one depending on the specific needs***

www.esi-group.com

PRACE Initiative

- **_Open-source software usage survey_**
  - Providing an access to HPC systems with pre-installed open-source software

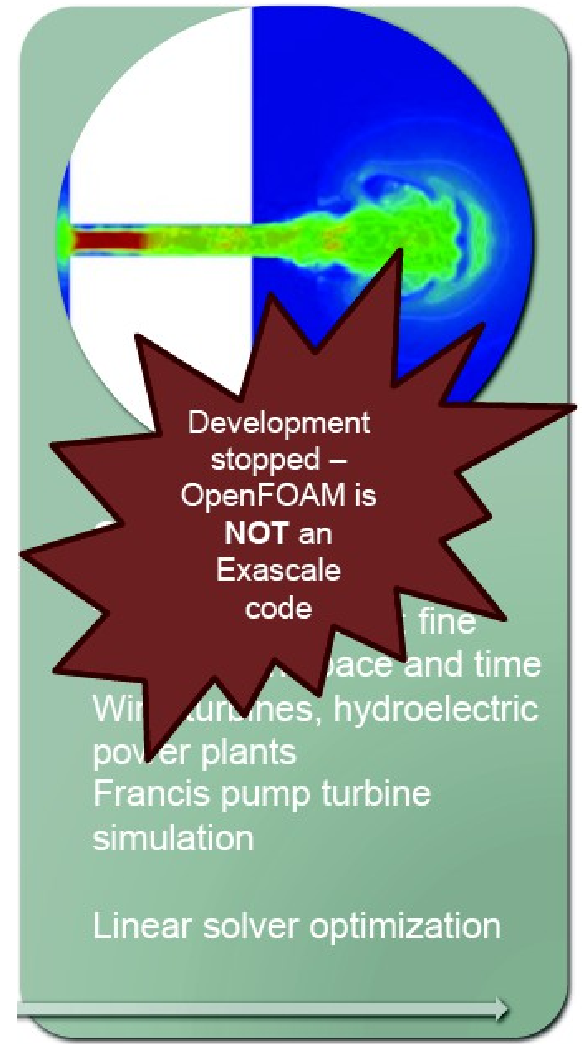
- **_Open-source applications tuning for perspective HPC systems_**
  - «…the main goal is to improve scalability of OpenFOAM for industrial relevant cases»
  - «As is typical of CFD applications the scalability bottleneck has been identified as being in the MPI communication pattern of the linear algebra core libraries.»



*PRACE Second Implementation Project, RI-283493. D9.1.1. Support for Industrial Applications Year 1, 2012

Verdict

- ***PRACE research projects:***
  - *P. Dagna, J. Hertzer.* Evaluation of Multi-threaded OpenFOAM Hybridization for Massively Parallel Architectures. ***http://www.prace-project.eu/IMG/pdf/wp98.pdf***
  - M. Manguoglu. A General Sparse Sparse Linear System Solver and Its Application in OpenFOAM. ***http://www.prace-ri.eu/IMG/pdf/A_General_Sparse_Sparse_Linear_System_Solver_and_Its_Application_in_OpenFOAM.pdf***



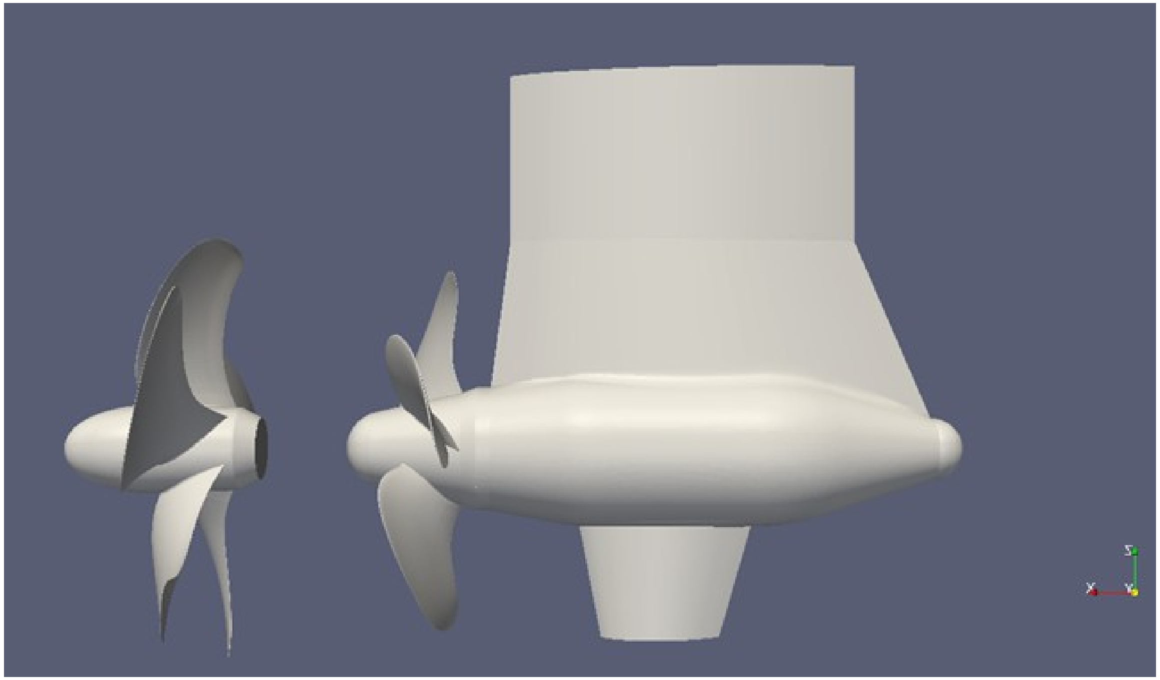
- ***Adoption of most useful applications for exascale computations***

****M. Parsons.* Software co-design for extreme scale computing // *Extreme Scale Scientific Computing Workshop*, MSU, Russia, 2014.**

Initial Problem Statement (1)

Hydrodynamic characteristics modeling for marine propellers:

- ***Incompressible flow***
- ***Unsteady (~10^4 time steps)***
- ***Two moving domains***
- ***k-ω SST turbulence model***
- ***Grids ~ 100M cells***

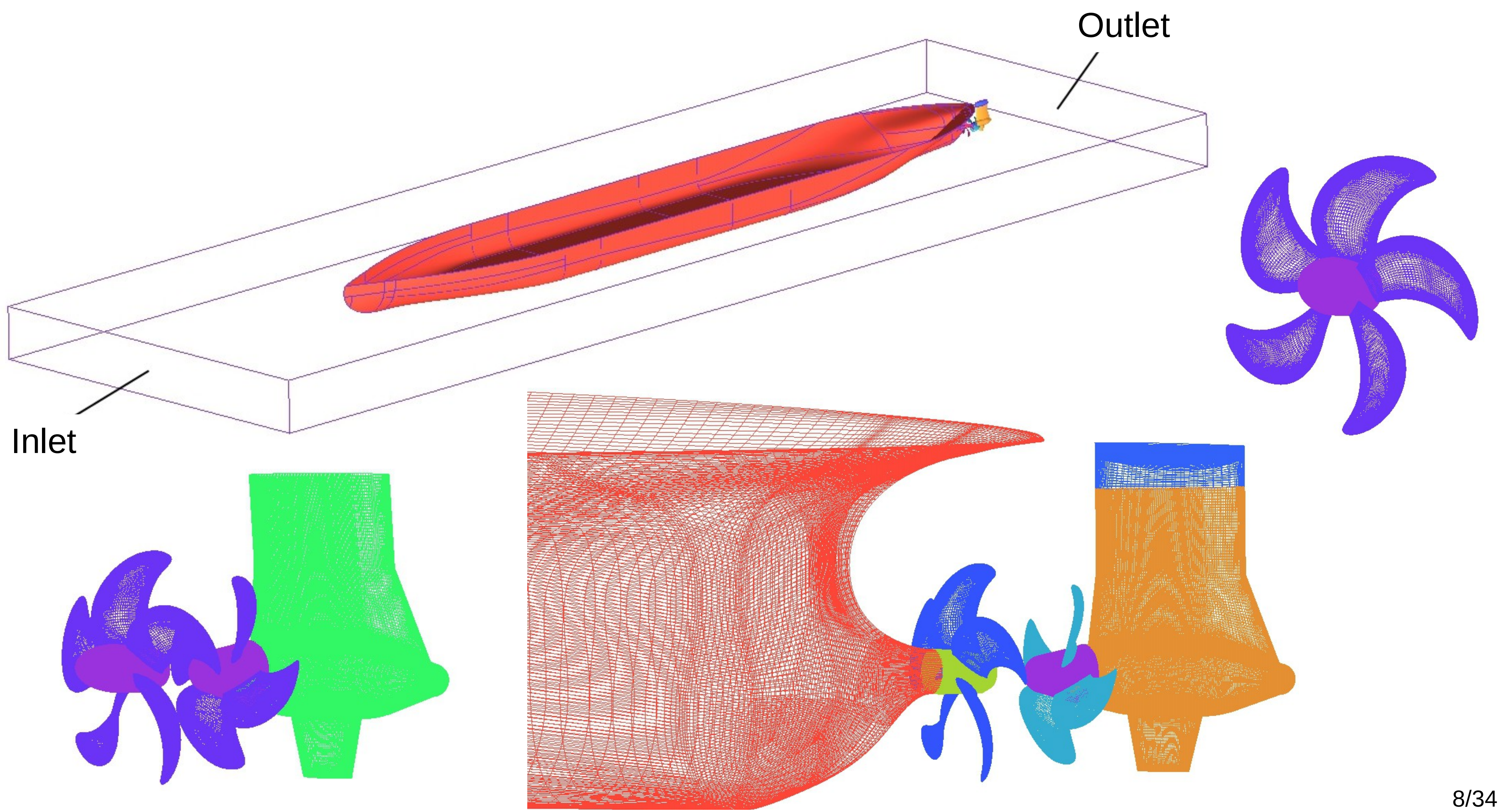


1. Is it possible, using OpenFOAM?

2. If so, how to speedup these simulations?

Initial Problem Statement (2)

Computational grids 41, 60 and 99M cells



Strategy Planned

- ***Solution of pressure Poisson equation takes 70-90% of overall simulation time: good candidate for revision***

- ***SLAE solver could be implemented as a dynamically loaded plug-in***

Started in 2012

- ***Possible directions of SLAE solution speedup:***
  - Another mathematical methods (vs OpenFOAM)?
  - Hybrid programming models?
  - Coprocessors/accelerators?

- ***Available alternatives for GPUs:***
  - Ofgpu
  - Cufflink
  - SpeedIT

Not for 100M grids...

2. SparseLinSol library

SparseLinSol Library Overview

<u>Library functionality:</u>

- ***A set of most popular methods for solving large sparse SLAEs:***
  - Krylov subspace iterative methods
  - Algebraic multigrid methods (*hypre* based)
  - Gauss-Seidel, Jacobi, Chebyshev polynomial methods
- ***Hybrid parallel programming model for multicore CPUs***
- ***CUDA-code extension to use GPU accelerators***
- ***OpenFOAM coupling plugin***
- ***<u>Setup part of multigrid methods is still on CPUs and with MPI...</u>***

Implementation Details for CPUs

<u>Hybrid programming model:</u>

- ***<u>NOT</u> MPI+OpenMP***
  - MPI and MPI+OpenMP applications coupling issues
  - NUMA-architecture
- ***MPI + Posix Shared Memory (MPI+ShM)***
  - Hybridization is logical: initially all the processes are equal
  - Totally hidden inside the code of the library
- ***Hierarchical data distribution in order to fit hardware memory hierarchy (co-design)***
- ***4 levels of abstractions: Node / Device / Numa-node / Core***

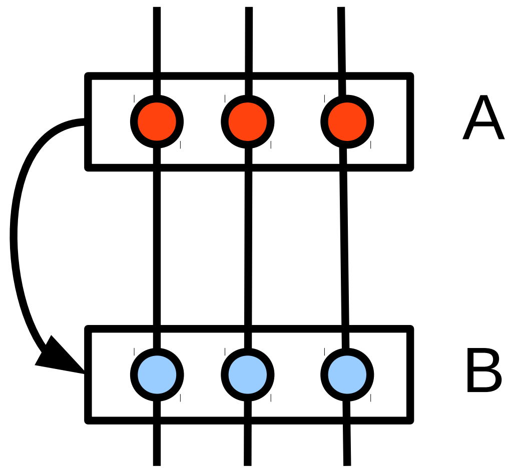
All the algorithms must be revisited...

MPIShM (1)

Auxiliary library with intra-node synchronization primitives:

- ***IBarrier***

```
void MPIShM_IBarrier_<...>_init (MPIShM_IBARRIER_type *Barrier, proc_id *id);
void MPIShM_IBarrier_<...>_wait (MPIShM_IBARRIER_type *Barrier, proc_id *id);
```



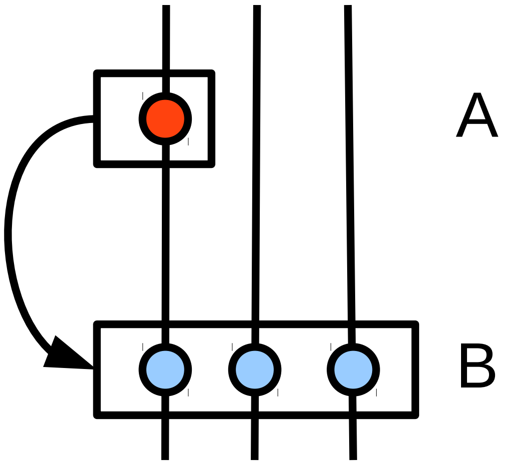
MPIShM (2)

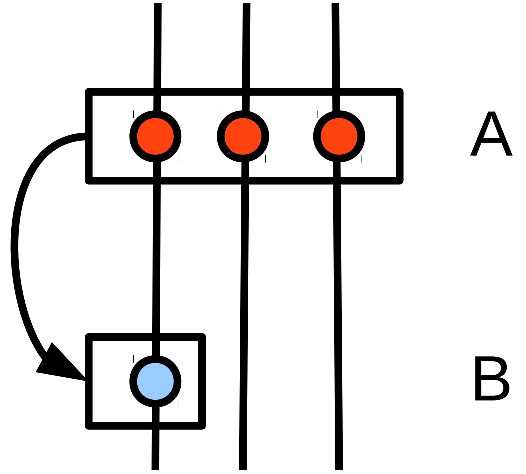
ISyncOne

```
void MPIShM2_ISyncOne_<...>_init (MPIShM_ISYNC_type *Sync, proc_id *id);
void MPIShM2_ISyncOne_<...>_wait (MPIShM_ISYNC_type *Sync, proc_id *id);
```

ISyncAll

```
void MPIShM2_ISyncAll_<...>_init (MPIShM_ISYNC_type *Sync, proc_id *id);
void MPIShM2_ISyncAll_<...>_wait (MPIShM_ISYNC_type *Sync, proc_id *id);
```



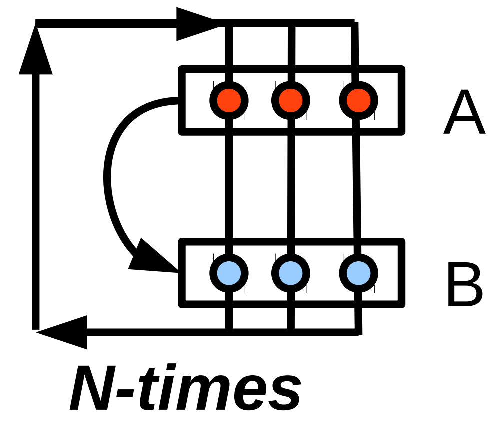


MPIShM (3)

Implementations:

- ***Semaphores***
- ***Atomics***

IBarrier implementation timings, usec:



	Core-i5, 4 threads	**T-Nano, 16 threads**	**Zilant, 24 threads**
Pthreads	32	133	207
Semaphores	21	97	156
Atomics	0.3	2.8	5.9

ShM model ~10% faster than MPI inside the single node

Lomonosov: 2x4 cores Intel X5570 (Nehalem), 2xNVIDIA X2070, IB QDR
T-Nano: 2x8 cores Intel E5-2670 (Sandy Bridge), IB QDR
Zilant: 2x12 cores AMD Opteron 6174 (Magny Cours), IB QDR

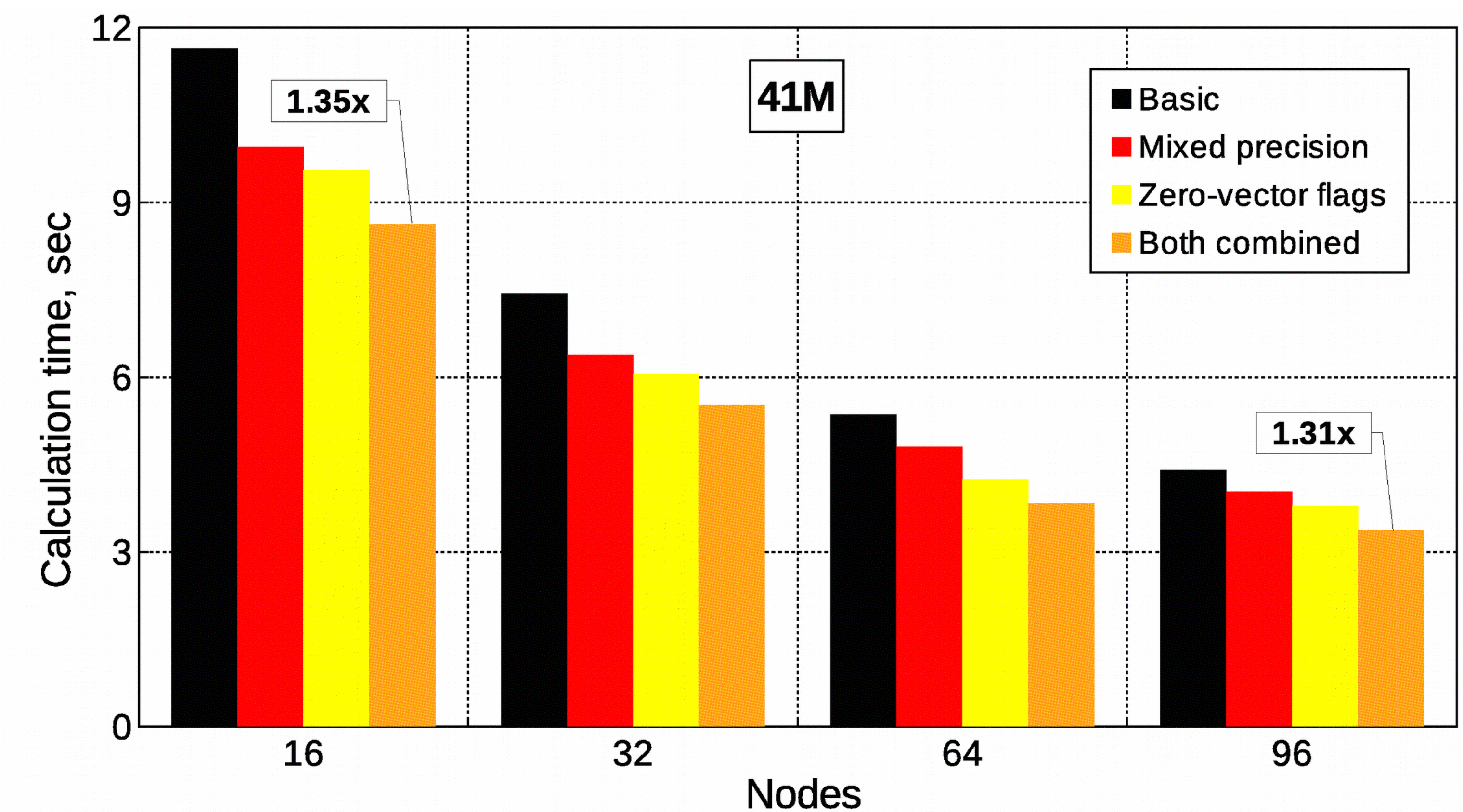
Implementation Details for GPUs

- ***GPU is quite different throughput-oriented architecture***
  - check all code if it is memory efficient on GPU
  - different matrix formats required
- ***Massively-parallel algorithms are required***
- ***Can't move the whole workflow on GPU:***
  - lots of separate kernels and CPU-GPU memcopies
- ***MultiGPU algorithms are even more difficult***
  - CPU-GPU copy for each MPI call, CPU-GPU synchronization
  - scalability is limited due to GPU overheads

Optimizations (1)

General optimizations:

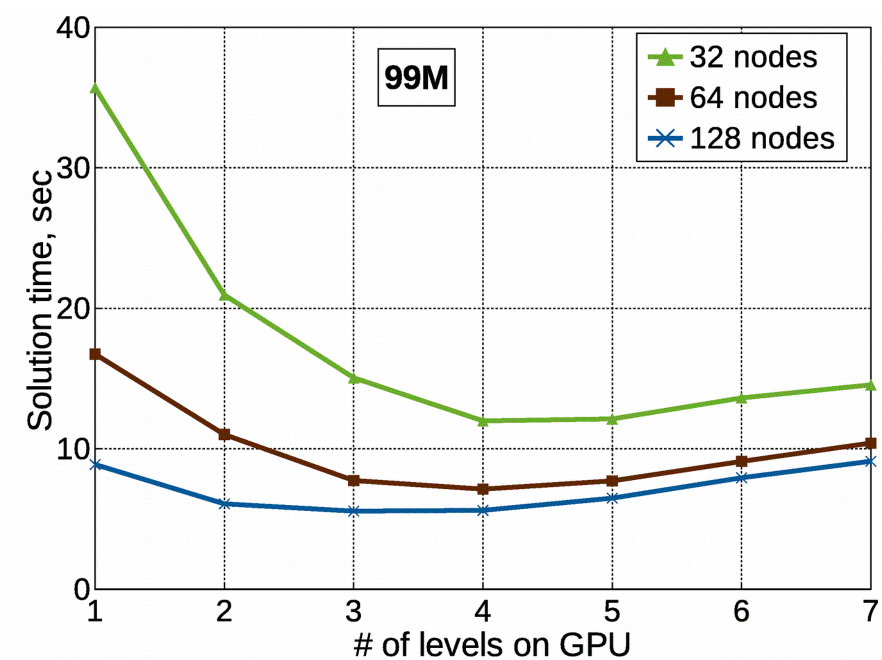
- *Zero vector flags*
- *Single precision for multigrid matrices hierarchy*

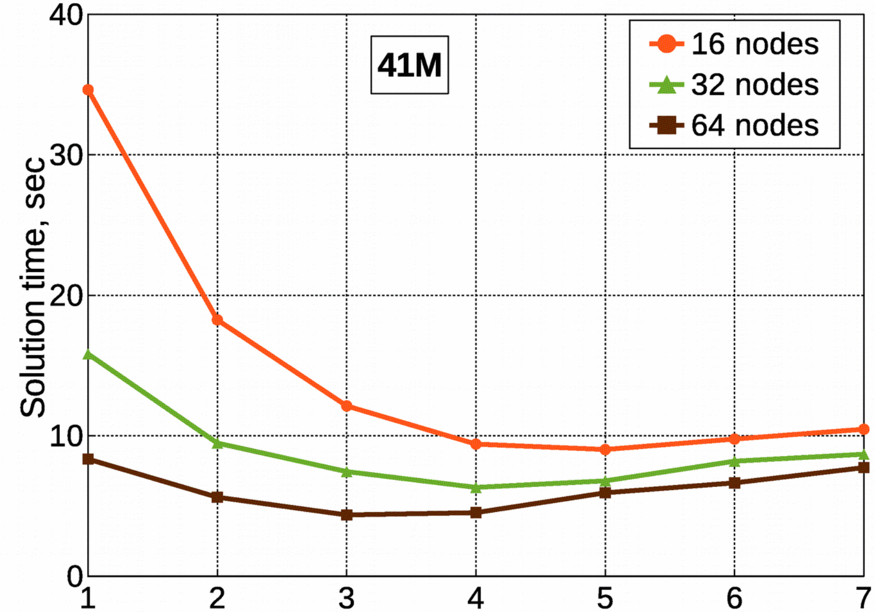


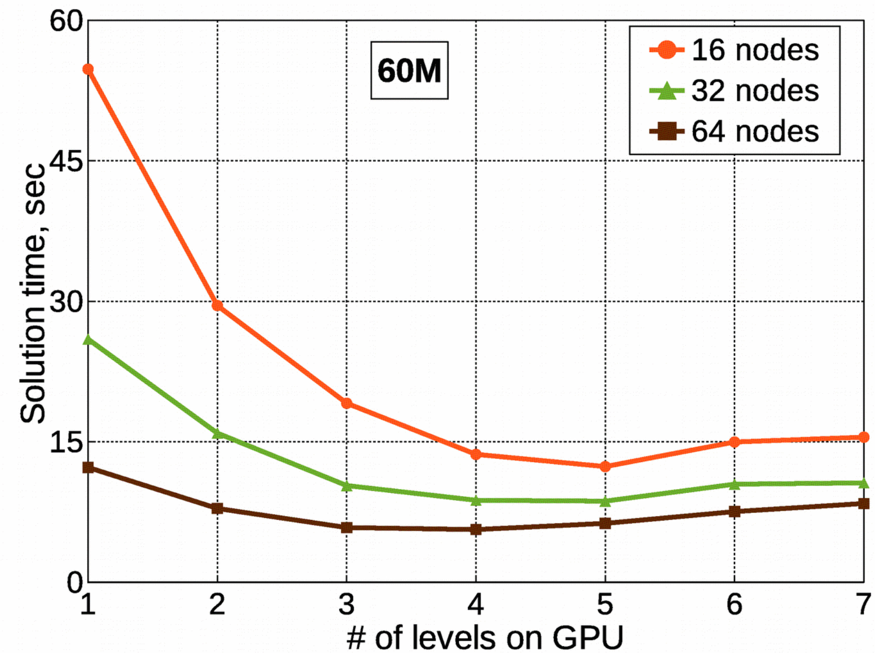
Optimizations (2)

GPU-specific optimization:

- ***Variable number of multigrid levels on GPUs***

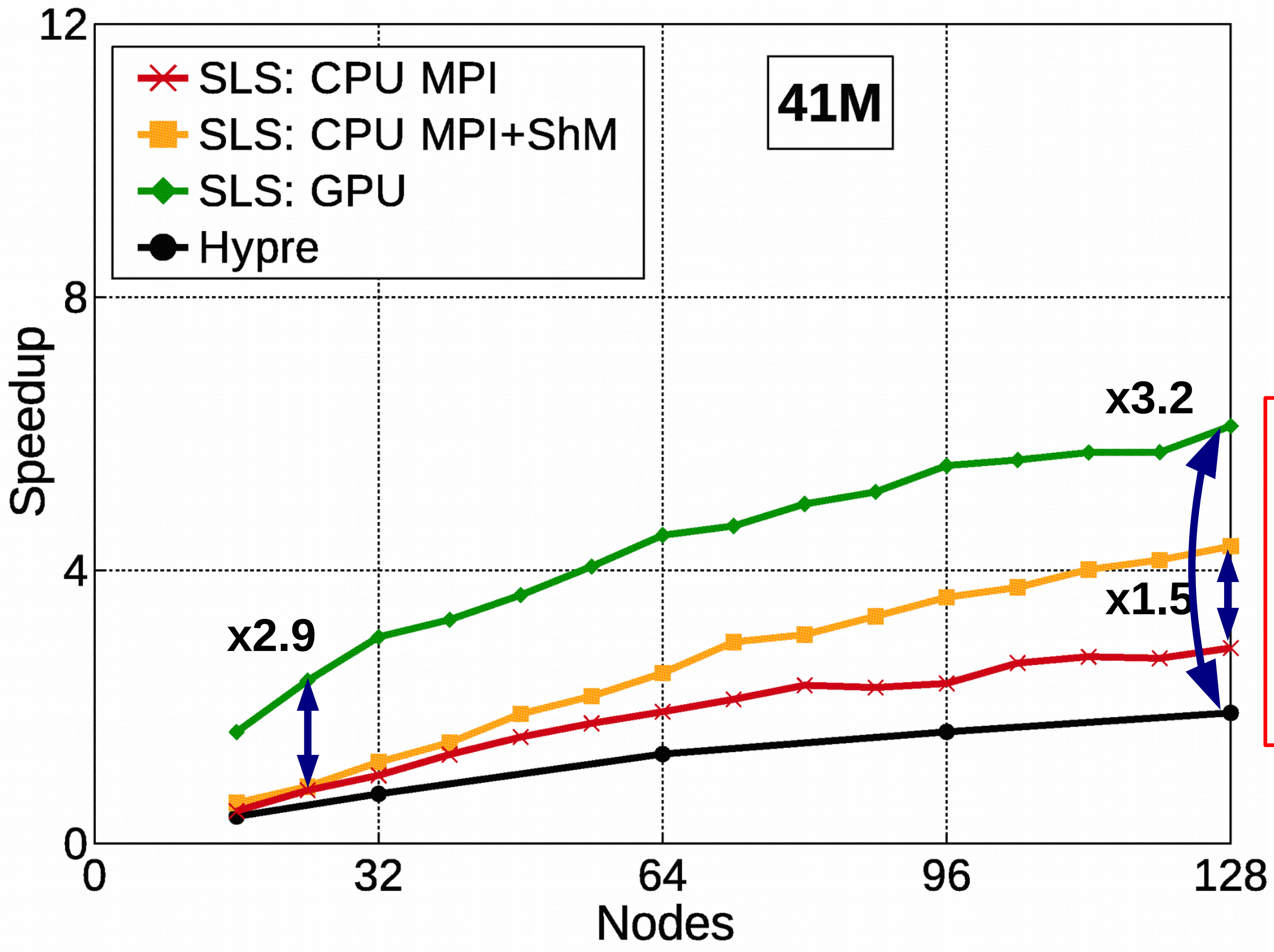






Estimate: $N_{nz} \sim 180K$ per GPU

Strong Scalability, 41M



Lomonosov

Parallel efficiency:

MPI: 72%

MPI+ShM: 91%

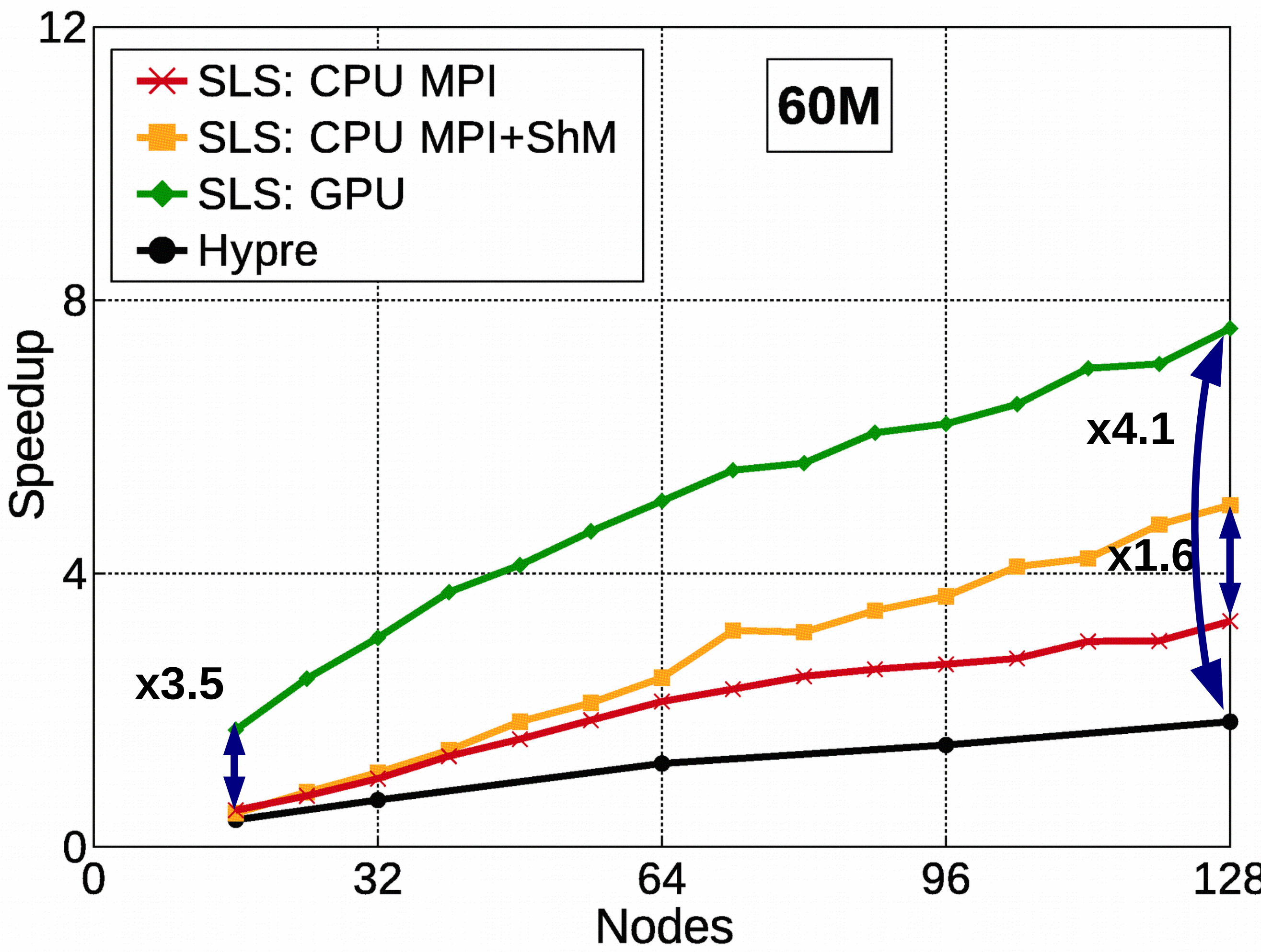
GPU: 51%

- **GPU vs CPU ShM: 1.4-2.9**
- **GPU vs CPU MPI: 2.1-3.4**
- **GPU vs Hypre: 3.2-4.1**

Numerical method: BiCGStab + CAMG, Chebyshev polynomial smoother

Scalability is normalized to 32 nodes “SLS CPU MPI” point

Strong Scalability, 60M

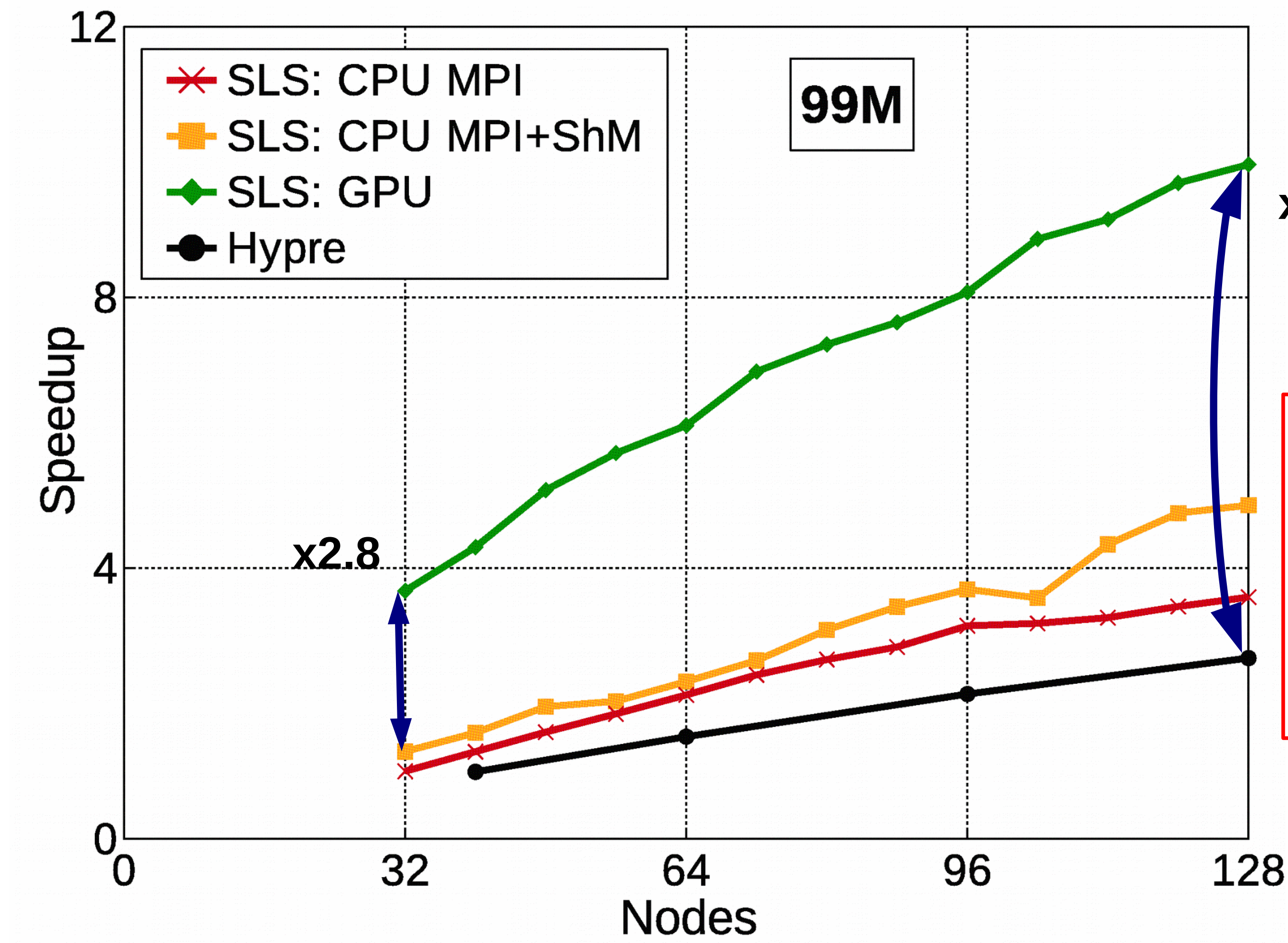


Lomonosov
Parallel efficiency:
MPI: 83%
MPI+ShM: 114%
GPU: 62%

- **GPU vs CPU ShM: 1.5-3.1**
- **GPU vs CPU MPI: 2.3-3.2**
- **GPU vs Hypre: 4.1-4.5**

Numerical method: BiCGStab + CAMG, Chebyshev polynomial smoother
Scalability is normalized to 32 nodes “SLS CPU MPI” point

Strong Scalability, 99M



Lomonosov

Parallel efficiency:

MPI: 89%

MPI+ShM: 95%

GPU: 68%

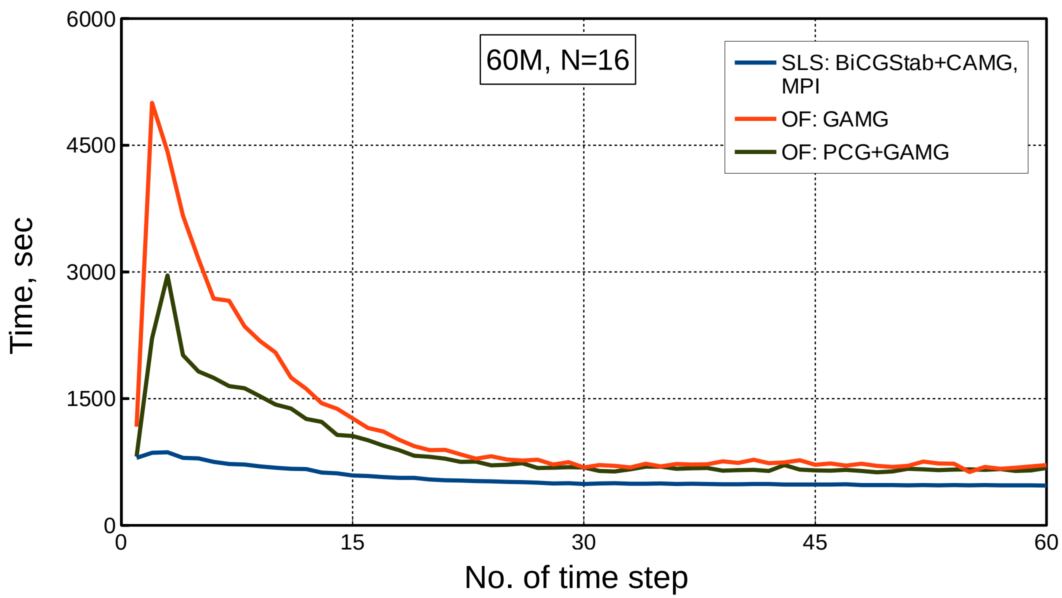
- **GPU vs CPU ShM: 2-2.8**
- **GPU vs CPU MPI: 2.6-3.7**
- **GPU vs Hypre: 3.7-4.3**

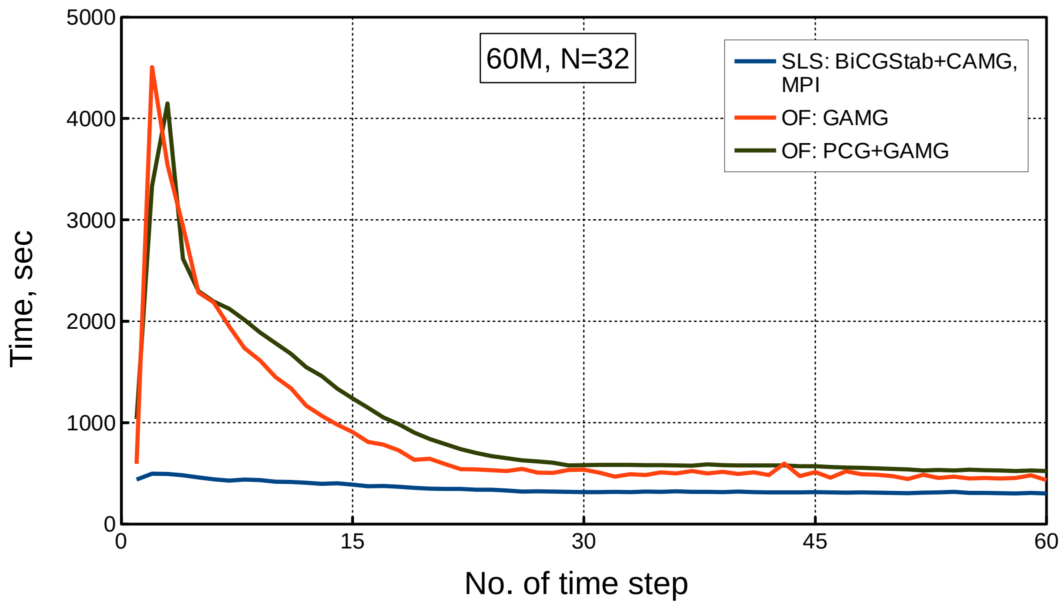
Numerical method: BiCGStab + CAMG, Chebyshev polynomial smoother

Scalability is normalized to 32 nodes “SLS CPU MPI” point

3. OpenFOAM simulations

Time Step Timings





"Lomonosov"

SLS: BiCGStab + CAMG, GS	***OF: GAMG***	***OF: PCG + GAMG***
Total time, hours		
9.2	19.8	15.3
Typical time step, seconds		
480	700	680
Total time, hours		
5.9	14.4	16.4
Typical time step, seconds		
310	460	530

MPI vs MPI+ShM

One time step in detail:

"T-Nano"

Cores	*Non-SLAE time*	*SLS: MPI*			*SLS: MPI+ShM*		
		SLS: Setup	*SLS: Solve*	*SLS: Total*	*SLS: Setup*	*SLS: Solve*	*SLS: Total*
128	120	361	1032	1393	431	872	1303
256	67	170	517	697	220	411	631
384	75	135	442	577	165	298	463
Speedup:			**2.33**	**2.41**		**2.93**	**2.81**

Linear scalability for the solve-part of SparseLinSol library

4. Conclusion

- ***SparseLinSol – new library for solving large sparse SLAEs***
- ***Implements MPI+ShM hybrid programming model for multicore HPC systems***
- ***Implements GPU acceleration***
- ***Up to 4x speedup against hypre***
- ***OpenFOAM coupling plug-in implemented***
- ***Preliminary results demonstrate OpenFOAM acceleration on large-scale hydrodynamics problems***

5. Future Plans

- ***Detailed benchmarking & testing with OpenFOAM***
- ***Comparison with AmgX library by NVIDIA***
- ***Setup part of multigrid methods...***

AmgX vs. SLS

- Compare similar methods
- Only solve part, SLS doesn't have GPU acceleration of setup part yet
- AmgX version: 1.2, trial (available on NVIDIA site)
- Hardware: Lomonosov-2
  - 1xCPU: E5-2697 v3, 14 cores
  - 1xGPU: K40s
  - FDR Infiniband

AmgX config file

```
{
    "config_version": 2,
    "determinism_flag": 1,
    "solver": "PBICGSTAB",
    "max_iters": 50,
    "monitor_residual": 1,
    "convergence": "RELATIVE_INI",
    "tolerance": 1e-10,
    "norm": "L2",
    "obtain_timings": 1,
    "store_res_history": 1,
    "print_grid_stats": 1,
    "print_solve_stats": 1,
    "preconditioner": {
        "scope": "amg_solver",
        "solver": "AMG",
        "max_levels": 24,
        "min_coarse_rows": 100,
        "max_iters": 1,
        "cycle": "V",
        "selector": "PMIS",
        "interpolator": "D2",
        "smoother": "BLOCK_JACOBI",
        "relaxation_factor": 0.99,
        "coarse_solver":
                 "DENSE_LU_SOLVER",
        "dense_lu_num_rows": 10,
        "presweeps": 2,
        "postsweeps": 2,
        "coarsest_sweeps": 0,
        "interp_max_elements": 4,
        "print_grid_stats": 1,
        "strength": "AHAT",
        "strength_threshold": 0.5,
        "interp_truncation_factor":
                                 0.25,
        "max_row_sum": 0.9
    }
}
```

SLS config file (1)

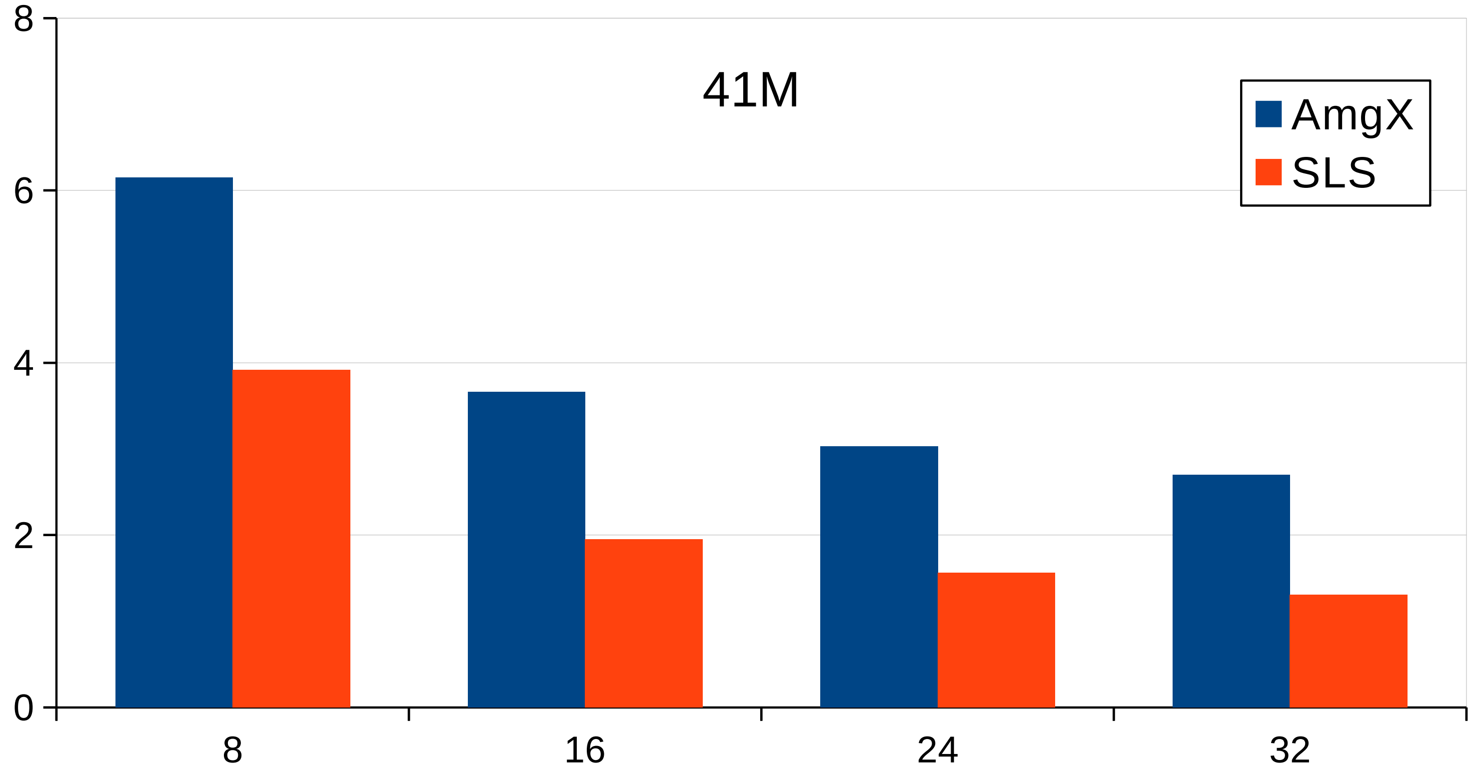
```
env.nthreads = 14;
env.nnumas = 1;
env.nsockets = 1;

env.param.solver  = SLS_CBiCGStab;
env.param.precond = SLS_CAMG;
env.param.pre_smoother       = SLS_Jacobi;
env.param.post_smoother      = SLS_Jacobi;
env.param.coarse_grid_solver = SLS_Direct;

env.param.MG_fp_type = SLS_Float32;

env.SLS_UPDATE_GLOBAL_PARAMS("accelerator", SLS_GPU);
env.SLS_UPDATE_GLOBAL_PARAMS("acc_levels", 2);

env.SLS_UPDATE_GLOBAL_PARAMS("matrix_reordering", 0);
env.SLS_UPDATE_GLOBAL_PARAMS("matrix_decomposition", 0);

env.SLS_UPDATE_SOLVER_PARAMS("abs_tolerance", 0.0, -1);
env.SLS_UPDATE_SOLVER_PARAMS("rel_tolerance", 1e-10, -1);
```

SLS config file (2)

```
    env.SLS_UPDATE_PRECOND_PARAMS("mg_num_paths", 2, -1);
    env.SLS_UPDATE_PRECOND_PARAMS("mg_coarse_matrix_size", 500, -1);
    env.SLS_UPDATE_PRECOND_PARAMS("mg_max_levels", 40, -1);

//  pmis
    env.SLS_UPDATE_PRECOND_PARAMS("mg_coarsening_type",    8, -1);
    env.SLS_UPDATE_PRECOND_PARAMS("mg_interpolation_type", 6, -1);
    env.SLS_UPDATE_PRECOND_PARAMS("mg_agg_num_levels",     2, -1);
    env.SLS_UPDATE_PRECOND_PARAMS("mg_agg_interpolation_type", 4, -1);

    env.SLS_UPDATE_PRE_SMOOTHER_PARAMS("max_iterations", 2, -1);

    env.SLS_UPDATE_POST_SMOOTHER_PARAMS("max_iterations", 2, -1);
```

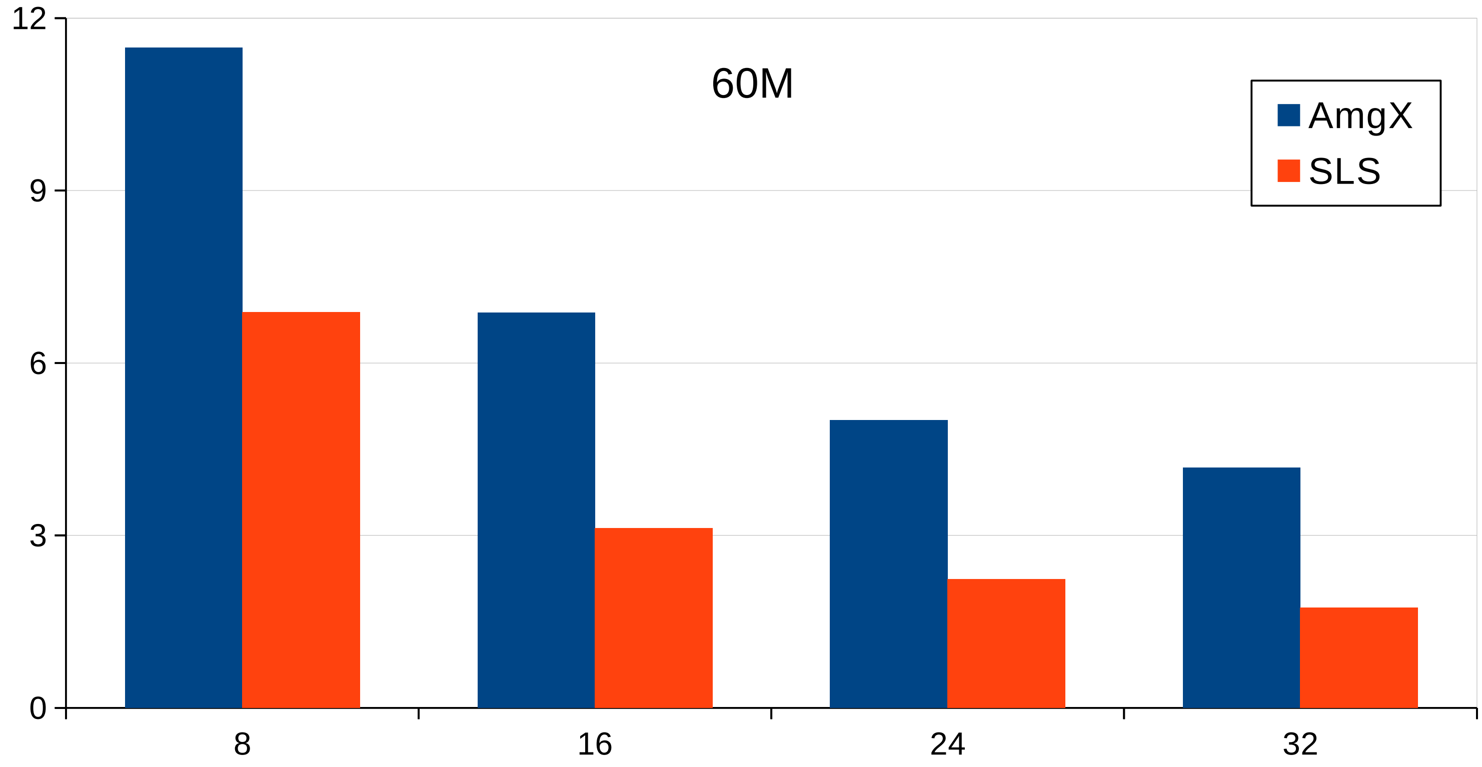
AmgX vs. SLS (1)

Solve part time comparison. 41M matrix, relative residual 1e-10



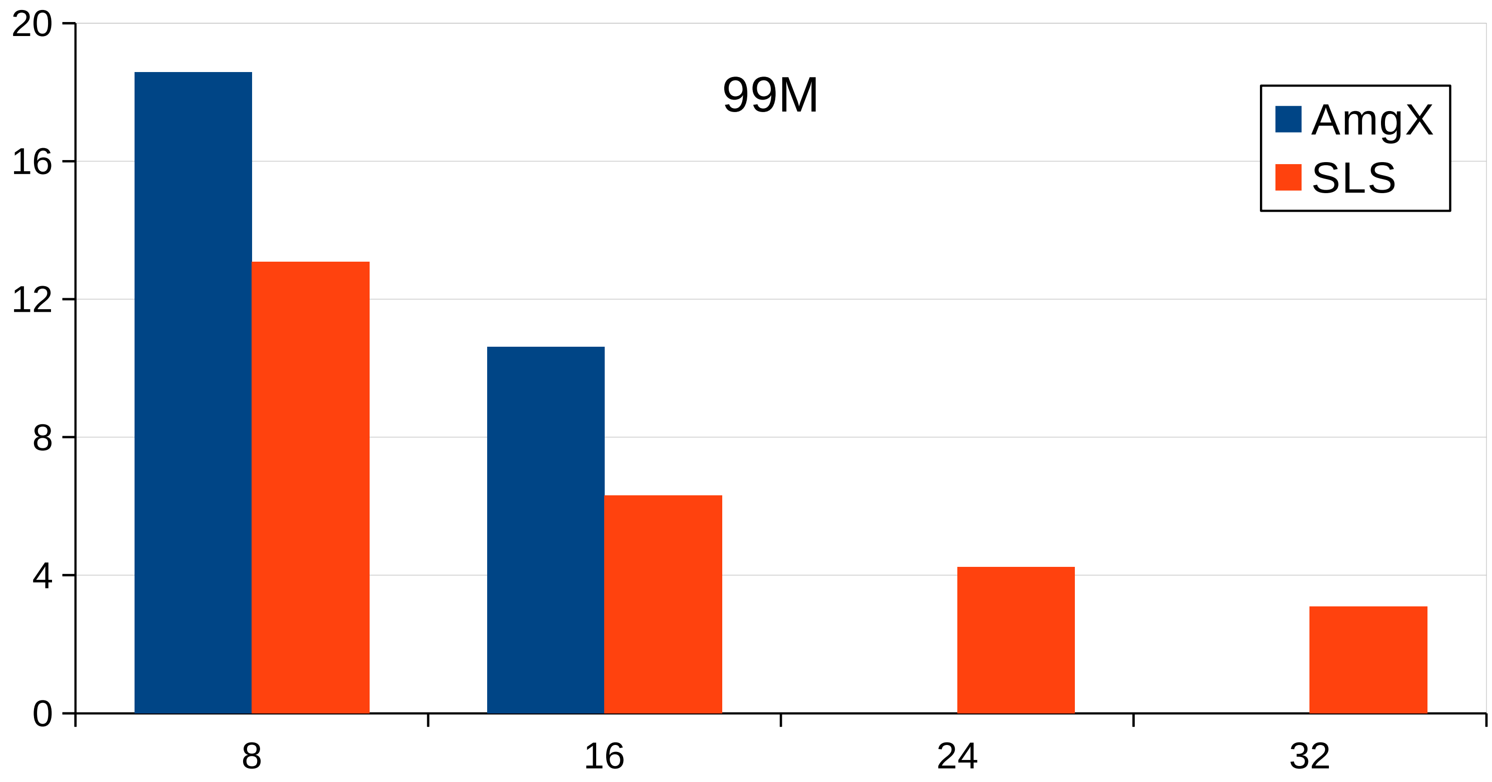
AmgX vs. SLS (2)

Solve part time comparison. 60M matrix, relative residual 1e-10



AmgX vs. SLS (3)

Solve part time comparison. 99M matrix, relative residual 1e-10



Thank You for Attention!