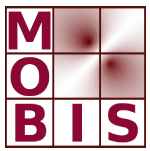
Parallelization Improvements

Gundolf Haase

Karl-Franzens University Graz

Graz, Dec 2024



What we knew

- Computation to communication ratio decreases faster then necessary
- Many multigrid levels are bad because of
  - Communication pattern
  - Efficiency

What we're looking at with Scalasca

- General computation to communication ratio
- On each multigrid level
  - Communication amount (bytes)
  - MPI-Timings
  - Computational load
  - Many more

Performance bottleneck: Communication

Solving a $8 \cdot 10^5$ system on 32 CPU-cores with PT

Accumulated MPI-Times of `communicator::accumulate()` inside the AMG preconditioner

Level	pre-smoother	post-smoother	sum	computation	ratio
0	0.15	0.89	1.04	2.42	2.32
1	0.57	0.35	0.92	1.03	1.11
2	0.42	0.15	0.57	0.44	0.77
3	0.26	0.09	0.35	0.19	0.54
4	0.14	0.11	0.25	0.14	0.56
5	0.20	0.21	0.41	0.16	0.39
6	0.27	0.27	0.54	0.12	0.22
7	0.27	0.27	0.54	0.0	0.0
8 (direct)			0.27	0.04	0.14

Performance bottleneck: Vector Assembling and Communication

Not caused by hardware!!

Problems

- Unbalanced vector accumulation (arithmetics and memory access)
- Poor computation to communication ratio on coarser MG levels
- No direct solver on GPUs $\rightarrow$ problem even worse!

Performance bottleneck: Vector Assembling and Communication

Not caused by hardware!!

Problems

- Unbalanced vector accumulation (arithmetics and memory access)
- Poor computation to communication ratio on coarser MG levels
- No direct solver on GPUs $\rightarrow$ problem even worse!

Solutions

- New communicator
  - Distribute boundary nodes equally for accumulation ($2\times$)
  - Reorder equation systems (shared nodes first)
  - Asynchronously send shared data once computed
- Improve computation-to-communication ratio
  - Hierarchic DD methods
  - System blocking on coarse MG levels

A straight-forward approach (old) to accumulation

When accumulating a vector v each communicator has to

1. Gather values at the indices B_s from v_s and store them in an appropriate buffer.
2. Send and receive values to and from his neighbours.
3. Add up the received values and stored them back in v_s at B_s.

Parallelization Problems

Quadratic communication complexity w.r.t shared node multiplicity

- Shared node *multiplicity*: Number of processes sharing a node.
- Multiplicity largely increases on coarse grids and high parallelization.
- Number of communications: $c = (m-1)m = \mathcal{O}(m^2)$

Poor load-balancing when filling communication buffers and accumulating shared nodes values $\rightarrow$ poor MPI performance.

- The number of elements per subdomain are usually well-balanced, the boundary sizes are not.
- Memory access when filling buffers has poor caching properties.
- Synchronous kernels get desynchronized just before communication.
- Synchronous parallel behavior is key for optimal MPI performance.

Accumulated values might be globally inconsistent.

$\rightarrow$ parallelization error (not spd anymore, convergence breaks).

Owner-Master concept for nodes

We have to distinguish between the owner of a node and its master.

Definition

Every process sharing one node is an **owner** of this shared node.

Definition

The **master** of a shared node has to be a *unique* process that handles all communication-related tasks for that node.

- The master should be one of the owners for optimal performance.
- Every process (= communicator) is the owner of an *unbalanced* set of nodes, and master owns a different *balanced* set of nodes.

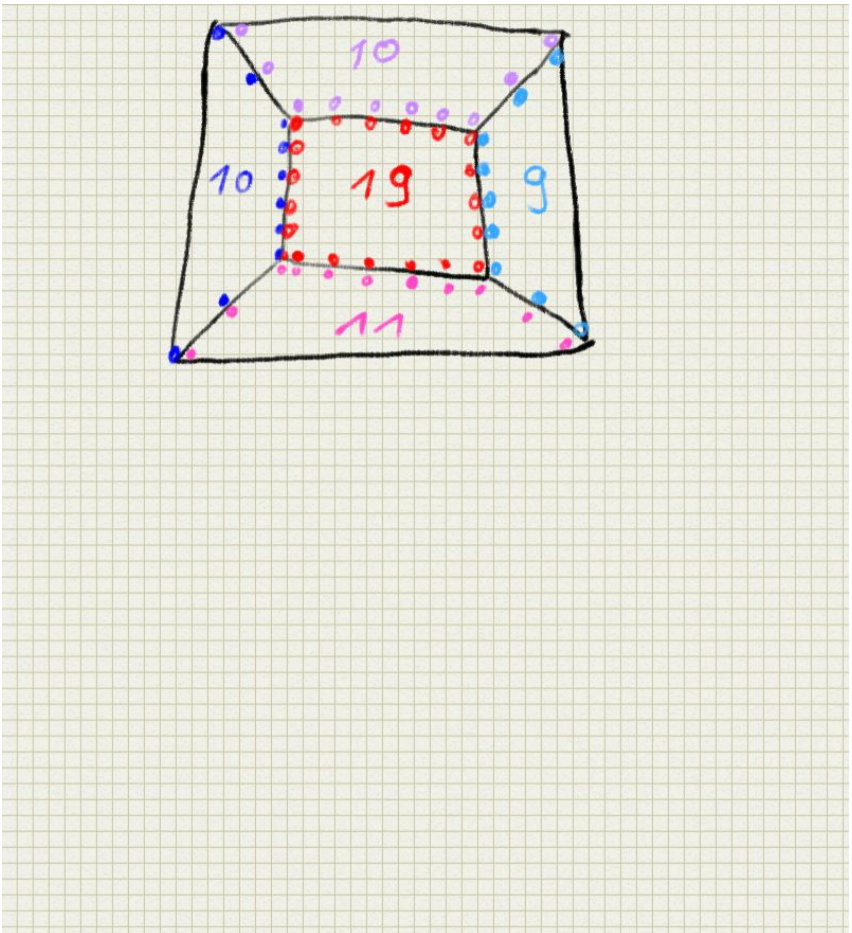
A detailed look on better balanced accumulation

When accumulating a vector v each communicator has to

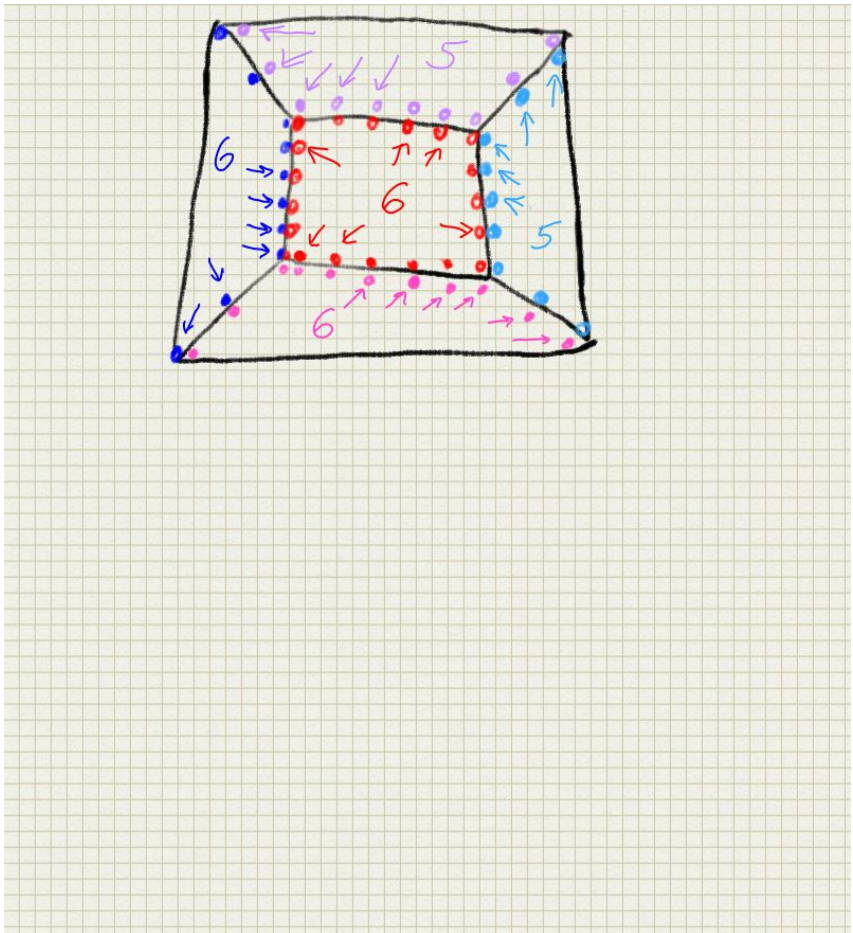
1. Gather values at the indices B_s from v_s and store them in an appropriate buffer (unbalanced).
2. Send owned shared node values values to the masters (unbalanced) and receive values of shared nodes he is master of (balanced).
3. Accumulate the values (balanced).
4. Send accumulated values back to owners (balanced) and receive values for owned shared nodes from masters (unbalanced).
5. Store received values back into v_s (unbalanced).

Illustration of master-owner concept

unbalanced distribution



balanced distribution



Accumulation intermediate

Problems solved:

1. Communication complexity
   - Number of communications: $c = 2(m-1) = \mathcal{O}(m)$.
2. Balanced accumulation
3. Value of each accumulated shared node identical on all owners.
4. Drawback: Now, communication is needed in intergrid transfer.

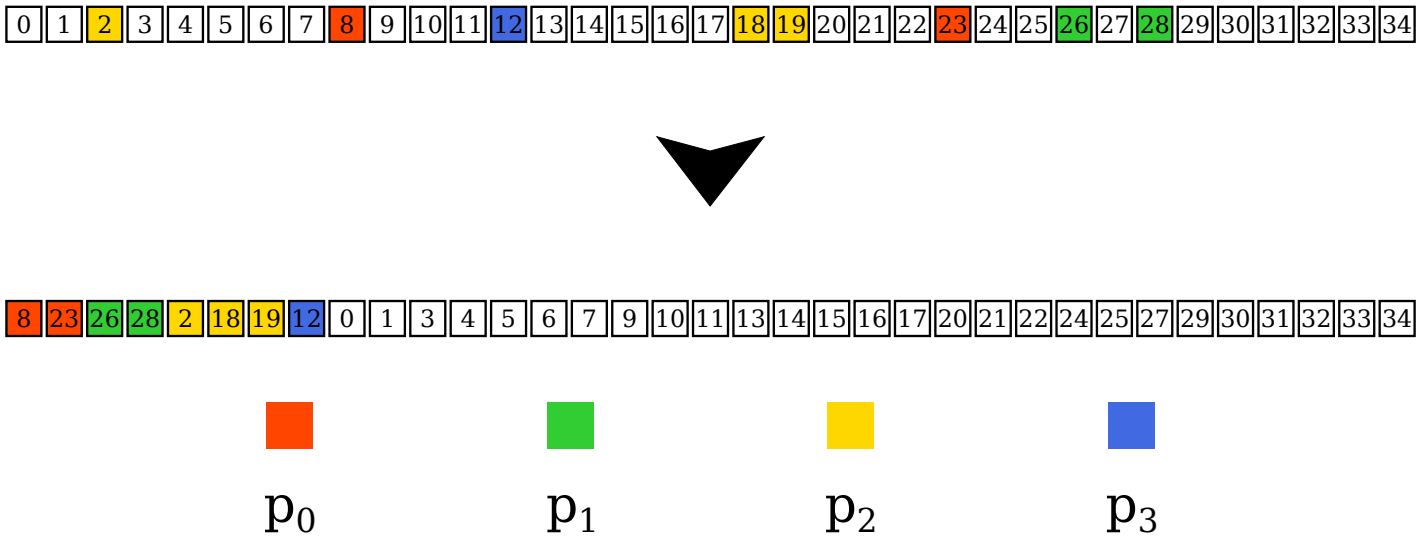
Still unsolved:

1. Unbalanced filling of MPI buffers
2. Memory access when filling buffers has bad caching properties.

System reordering

Reorder local node set in such a way, that no filling of MPI buffers is required:

- All shared nodes at beginning of local range.
- Ordered after the process they need to be sent to.



System reordering

- No need for separate MPI buffers.
- No need for unbalanced, cache-inefficient filling of buffers.
- Greatly reduces impact of the unbalanced boundary sizes.
- Does not effect computation kernels.
- Greatly simplifies other algorithms:
  - Extraction and accumulation of boundary matrix.
  - Overlapping of arithmetics and communication in Jacobi-smoother on GPUs.
- Reordering of matrices etc. neccessary (only once).

viscious remark: That was already available in TransCG in 1990.

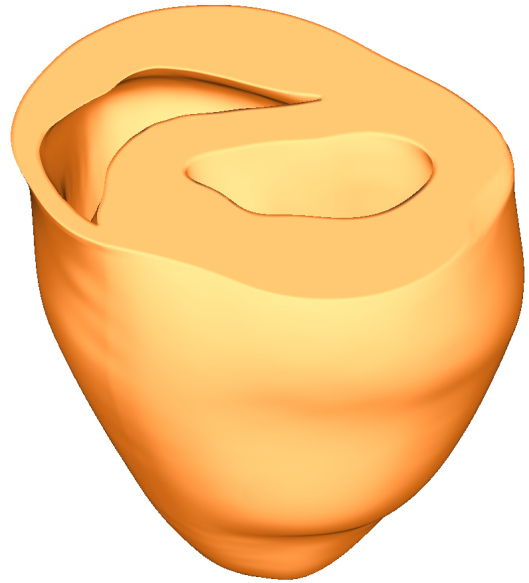
Benchmark configuration

Elliptic part of Bidomain-Equations (cardiac potential):

$$-\nabla \cdot (\bar{\sigma}_i + \bar{\sigma}_e)\nabla \phi_e = b$$

TBunnyC geometry: Simplified rabbit heart (CARP):

- 862,515 Nodes
- 5,082,272 Elements



Application performance

1 solution, 24 iterations

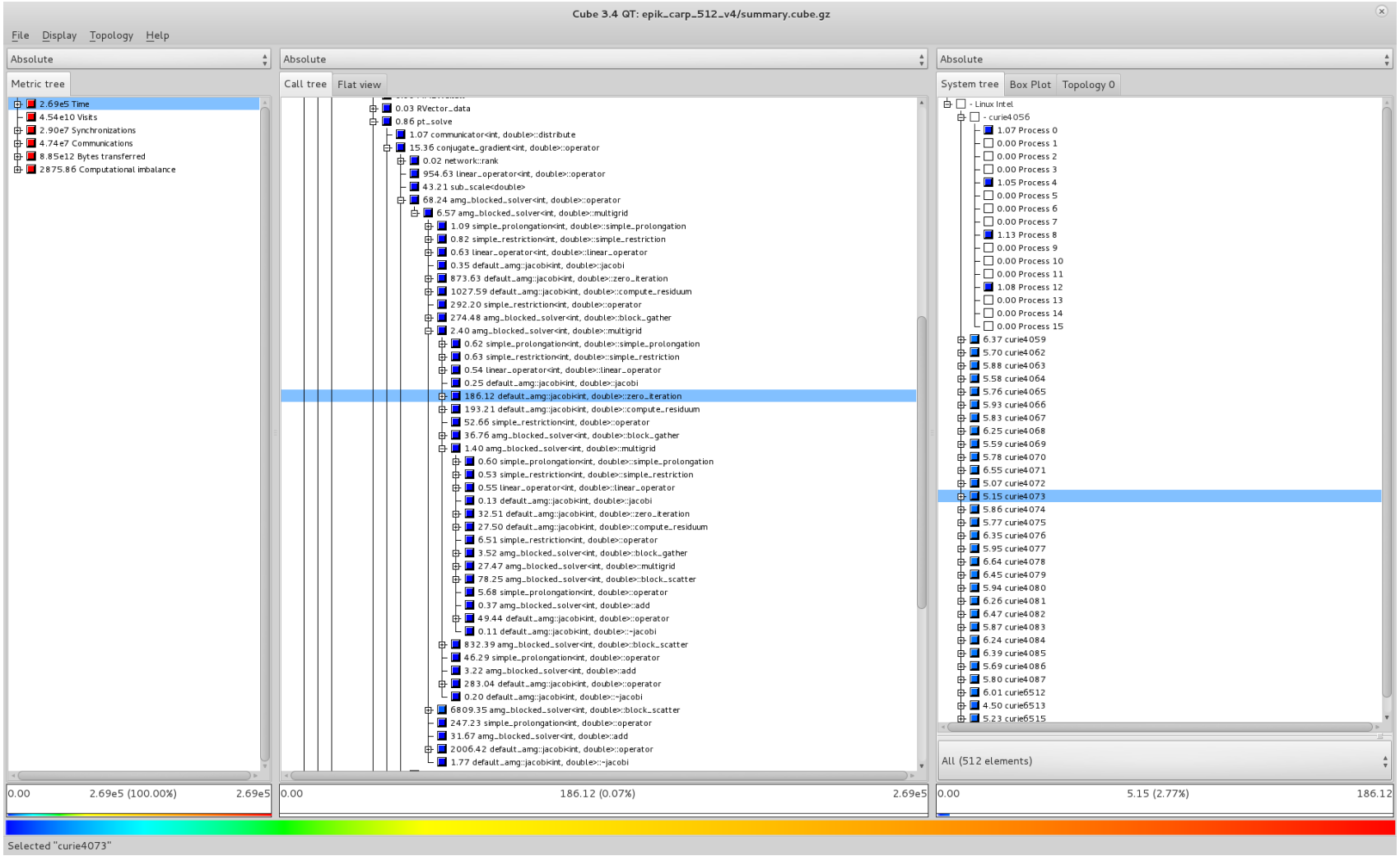
Mephisto cluster: 10 Xeon X5650 CPUs, 60 cores at 2.67GHz

Impl.1: Straight-forward algorithm

Impl.2: Reordered local systems and communicator with master-owner concept

Cores	Impl. 1 [sec]			Impl. 2 [sec]		
	Com setup	Solve	Scale	Com setup	Solve	Scale
4	0.026	1.42	1.0	0.071	1.43	1.0
12	0.012	0.70	0.68	0.031	0.62	0.76
48	0.144	**0.44**	0.26	0.160	**0.30**	0.40

Profiling with Scalasca



Profiling

Scalasca[1] profile of CG-AMG solve.

Times are global sums.

New communicator reduces imbalancies significantly.

Cores	Impl. 1 [sec] MPI	Imbalance	Impl. 2 [sec] MPI	Imbalance
4	1.8	1.3	0.5	0.3
12	3.6	2.1	1.0	0.2
48	32.3	**5.5**	13.3	**0.4**

[1] http://www.scalasca.org/

Coarse grid parallelization

Even with the new communicator the elliptic solver is only efficient up to 256 CPU-cores (7 Mill. DoF).

$\Longleftarrow$ worse ratio arithmetics to communication on coarser levels.

$\Longrightarrow$ Use less cores for each coarser levels.

- We assemble coarse data from *blocksize* neigbouring subdomains onto one process.
- Finest grid data τ_0 are distributed onto $p_0 \equiv p$ cores.
- Coarser data τ_k are distributed onto $p_k = p_{k-1}/\mathit{blocksize}$ cores $k = 1, \ldots, \ell$.

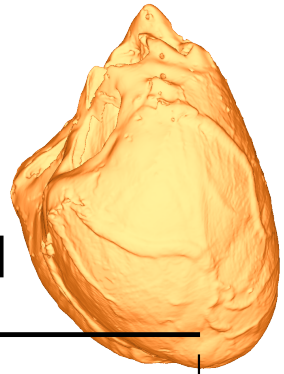
Results on curie

Oxford heart with 6.9 Mill. D.o.F.s.

200 time steps in simulation; elliptic solver part

Impl. 3: New comm. + blocked coarse grids (blocksize = 4)

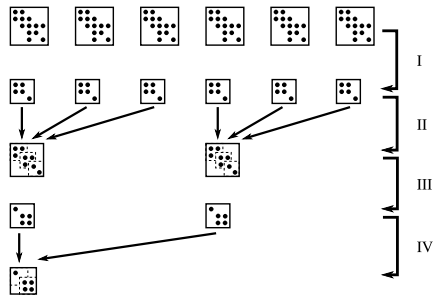
curie: 5040× 2 × 8 cores (Intel E5-2680) and 64 GB; Infiniband



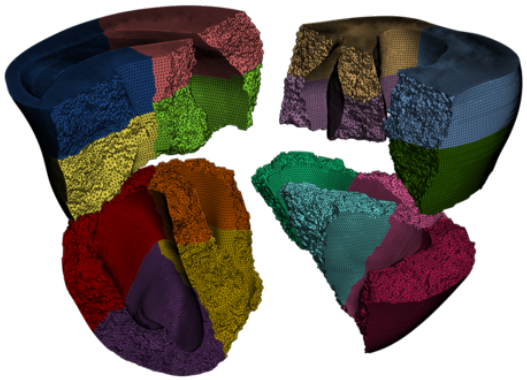
Cores	Impl. 2 [sec] time	Eff.	Impl. 3 [sec] time	Eff.	overall Eff.	DOFs/p [k]
32	481.1	1.00			1.00	216
64	254.6	0.95				108
128	**142.3**	0.85	197.0	1.00	0.61	54
192	107.3	0.75				36
256	**95.5**	0.63	102.8	0.96	**0.58**	27
320	88.3	0.55	**82.0**	0.96	**0.59**	21
512	103.5	0.29	**61.9**	0.80	**0.49**	13
768			**47.5**	0.69	**0.42**	**9**
1024			69.5	0.35	0.22	**7**

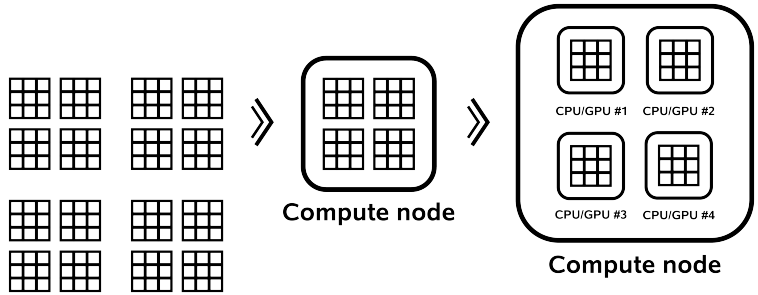
Summary: Hybrid Parallelization in the Solvers

- Equally balanced domain interfaces improve scalability.
  - new load balancing of interface nodes [A. Neic, M. Liebmann]
  - eliminates inconsistent round-off errors
- Reordering of unknowns avoids buffers for communication.
  $\Longrightarrow$ 3 $\times$ faster data exchange than standard
- Redistribution of Algebraic Multigrid Hierarchy
  - coarse-grid domain redistribution (blocking)
  - inactive compute cores on coarser grids
  - less MPI communication

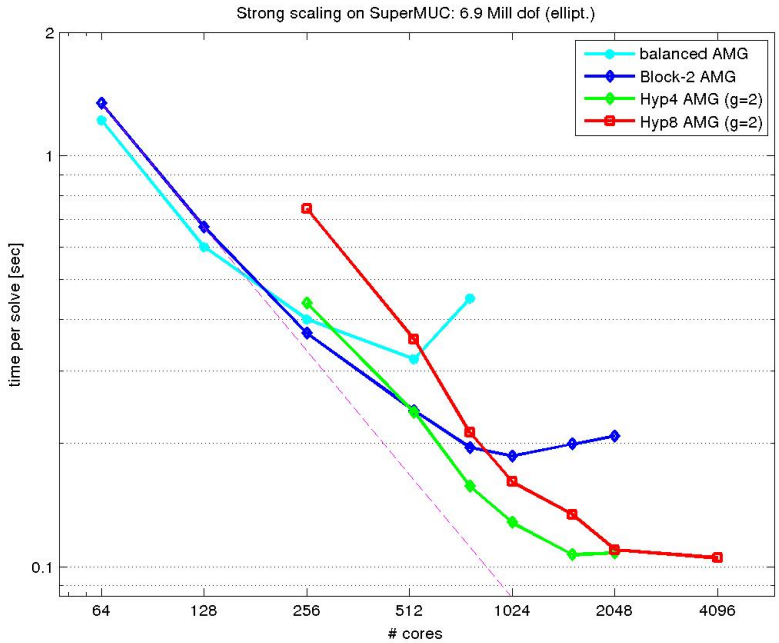


- Two–layer data distribution in the Toolbox for OpenMP + MPI





Improvements in strong scalability



Blocking of coarse grids (g=2) + hybrid OpenMP/MPI

- 6.9 Mill. dofs, elliptic part of bidomain equations, 200 solves (5 ms), 6500 AMG-PCG iterations
- blocking pays off for $n_{procs} > 128$.
- hybrid OpenMP/MPI pays off for $n_{procs} > 512$.
- varies slightly with available hardware