AMG2023

Code Description

A. General description:

The AMG2023 benchmark consists of a driver (amg.c), a simple Makefile, and documentation, which are available at https://github.com/LLNL/AMG2023. It requires an installation of hypre 2.27.0 or higher and uses hypre's parallel algebraic multigrid solver BoomerAMG in combination with a Krylov solver to solve two linear systems arising from diffusion problems on a cuboid discretized by finite differences. The problems are set up through hypre's linear-algebraic IJ interface. The problem sizes can be controlled from the command line.

For details on the algorithm and its parallel implementation/performance: see the following papers, which are available at https://github.com/hypre-space/hypre/wiki/Publications :

Van Emden Henson and Ulrike Meier Yang, "BoomerAMG: A Parallel Algebraic Multigrid Solver and Preconditioner", Appl. Num. Math. 41 (2002), pp. 155-177.

Hans De Sterck, Ulrike Meier Yang and Jeffrey Heys, "Reducing Complexity in Parallel Algebraic Multigrid Preconditioners", SIAM Journal on Matrix Analysis and Applications 27 (2006), pp. 1019-1039.

Hans De Sterck, Robert D. Falgout, Josh W. Nolting and Ulrike Meier Yang, "Distance-Two Interpolation for Parallel Algebraic Multigrid", Numerical Linear Algebra with Applications 15 (2008), pp. 115-139.

U. M. Yang, "On Long Range Interpolation Operators for Aggressive Coarsening", Numer. Linear Algebra Appl., 17 (2010), pp. 453-472.

A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang, "Multigrid Smoothers for Ultraparallel Computing", SIAM J. Sci. Comput., 33 (2011), pp. 2864-2887.

J. Park, M. Smelyanskiy, u. M. Yang, D. Mudigere, and P. Dubey, "High-Performance Algebraic Multigrid Solver Optimized for Multi-Core Based Distributed Parallel Systems", Proceedings of the International Conferencefor High Performance Computing, Networking, Storage and Analysis 2015 (SC15).

R. Li, B. Sjogreen, U. Yang, “A New Class of AMG Interpolation Methods Based on Matrix-Matrix Multiplications”, SIAM Journal on Scientific Computing, 43 (2021), pp. S540-S564, https://doi.org/10.1137/20M134931X

R. Falgout, R. Li, B. Sjogreen, L. Wang, U. M. Yang, “Porting hypre to Heterogeneous Computer Architectures: Strategies and Experiences”, Parallel Computing, 108, (2021), a. 102840

The driver provided with AMG builds linear systems for various 3-dimensional problems, which are described in Section D.

B. Coding:

hypre is written in C. The AMG solver in hypre is an SPMD code which uses MPI. Parallelism is achieved by data decomposition. The driver provided with the AMG benchmark achieves this decomposition by simply subdividing the grid into logical P x Q x R (in 3D) chunks of equal size.

C. Parallelism:

AMG is a highly synchronous code. The communications and computations patterns exhibit the surface-tovolume relationship common to many parallel scientific codes. Hence, parallel efficiency is largely determined by the size of the data "chunks" mentioned above, and the speed of communications and computations on the machine. AMG is also memory-access bound, doing only about 1-2 computations per memory access, so memory-access speeds will also have a large impact on performance.

D. Test problems

Problem 1 (default): The default problem is a 3D diffusion problem on a cuboid with a 27-point stencil. It is solved with AMG-GMRES. To determine when the solver has converged, the driver uses the relative residual stopping criteria:

$$\|r_k\|_2 / \|b\|_2 < 10^{-12}$$

Problem 2 (-problem 2): This problem solves a 3D Laplace problem on a cuboid with a 7-point stencil. To determine when the solver has converged, the driver uses the relative residual stopping criteria:

$$\|r_k\|_2 / \|b\|_2 < 10^{-8}$$

E. Building the Code

The AMG benchmark uses a simple Makefile system for building the driver.

It requires an installation of hypre 2.27.0, which can be downloaded from https://github.com/hypre-space/hypre via:

```
git clone -b v2.27.0 https://github.com/hypre-space/hypre.git
```

Information on how to install hypre is available here: https://hypre.readthedocs.io/en/latest/

Depending on the machine and desired programming models, different configurations are needed. hypre configure options can be obtained by typing

```
./configure --help
```

in hypre's src directory.

For a CPU machine, hypre generally can be easily installed by typing

```
./configure ;
make install
```

in the src directory.

If OpenMP threading within MPI tasks is desired, it should be configured as follows:

```
./configure --with-openmp --enable-hopscotch
```

If hypre should be run on Nvidia GPUs, use:

```
./configure --with-cuda
```

or

```
./configure --with-cuda --with-device-memory-pool
```

to use the memory pool option included with hypre.

If hypre is to be run on AMD GPUs, use:

```
./configure --with-hip --with-gpu-arch=gfx90a
--with-MPI-lib-dirs="${MPICH_DIR}/lib" --with-MPI-libs="mpi"
--with-MPI-include="${MPICH_DIR}/include"
```

If the problem to be run is larger than 2 billion, i.e., Px*Py*Pz*nx*ny*nz is larger than 2 billion, where Px*Py*Pz isthe total number of MPI tasks and nx*ny*nz the local problem sizeper MPI task, hypre needs to be configured with

```
--enable-mixed-int
```

since it requires 64-bit integers for some global variables. By default, hypre uses 32-bit integers.

To build the code, first modify the 'Makefile' file appropriately, then type

```
make
```

Other available targets are

```
make clean          (deletes .o files)
make distclean      (deletes .o files, libraries, and executables)
```

F. Optimization and Improvement Challenges

This code is memory-access bound. We believe it would be very difficult to obtain "good" cache reuse with an optimized version of the code.

G. Parallelism and Scalability Expectations

Previous versions of AMG have been run on the following platforms:

BG/Q - up to over 1,000,000 MPI processes
BG/P - up to 125,000 MPI processes
and more

Consider increasing both problem size and number of processors in tandem. On scalable architectures, time-to-solution for AMG will initially increase, then it will level off at a modest numbers of processors, remaining roughly constant for larger numbers of processors. Iteration counts will also increase slightly for small to modest sized problems, then level off at a roughly constant number for larger problem sizes.

For example, we get the following timing results (in seconds) for a system with a 3D 27-point stencil, distributed on a logical P x Q x R processor topology, with fixed local problem size per process given as 96 x 96 x 96:

P x Q x R	procs	setup time	solve time
8x 8x 8	512	14.91	51.05
16x16x 8	2048	15.31	53.35
32x16x16	8192	16.00	57.78
32x32x32	32768	17.55	65.19
64x32x32	65536	17.49	64.93

These results were obtained on BG/Q using MPI and OpenMP with 4 OpenMP threads per MPI task and configuring hypre with --enable-hopscotch --enable-persistent and --enable-bigint.

To measure strong scalability, it is important to change the size per process with the process topology:

The following results were achieved on RZTopaz for a 3D 7-pt Laplace problem on a 300 x 300 x 300 grid.

```
srun -n <P*Q*R> amg -P <P> <Q> <R> -n <nx> <ny> <nz> -problem 2
```

Using MPI only

P x Q x R	nx x ny x nz	setup time	solve time
1 x 1 x 1	300x300x300	43.37	61.85
2 x 1 x 1	150x300x300	31.06	42.09
2 x 2 x 1	150x150x300	15.68	22.74
2 x 2 x 2	150x150x150	8.44	12.59
4 x 2 x 2	75x150x150	5.37	8.39
4 x 4 x 2	75x 75x150	2.70	6.80

Using MPI with 4 OpenMP threads per MPI task

P x Q x R	nx x ny x nz	setup time	solve time
1 x 1 x 1	300x300x300	17.56	20.81
2 x 1 x 1	150x300x300	12.04	14.48
2 x 2 x 1	150x150x300	6.35	8.78
2 x 2 x 2	150x150x150	3.14	6.84
4 x 2 x 2	75x150x150	2.44	6.73

H. Running the Code

The driver for AMG is called `amg'. Type

```
amg -help
```

to get usage information. This prints out the following:

```
Usage: amg [<options>]

  -problem <ID>: problem ID
       1 = solves 1 problem with AMG-PCG (default)
       2 = solves 1 problem AMG-GMRES(100)

  -n <nx> <ny> <nz>: problem size per MPI process (default:
                     nx=ny=nz=10)

  -P <px> <py> <pz>: processor topology (default: px=py=pz=1)

  -print       : prints the system

  -printstats  : prints preconditioning and convergence stats

  -printallstats  : prints preconditioning and convergence stats
                    including residual norms for each iteration
```

All arguments are optional. An important option for the AMG compact application is the `-P' option. It specifies the MPI process topology on which to run.

The `-n' option allows one to specify the local problem size per MPI process, leading to a global problem size of <Px>*<nx> by <Py>*<ny> by <Pz>*<nz>.

H. Timing Issues

If using MPI, the whole code is timed using the MPI timers. If not using MPI, standard system timers are used. Timing results are printed to standard out and are divided into "Setup Phase" times and "Solve Phase" times for both problems.

I. Memory Usage

AMG's memory needs are somewhat complicated to describe. They are very dependent on the type of problem solved and the AMG options used. When turning on the '-printstats' option, memory complexities <mc> are displayed, which are defined by the sum of non-zeroes of all matrices (both system matrices and interpolation matrices on all levels) divided by the number of non-zeroes of the original matrix, i.e., at least about <mc> times as much space is needed. However, this does not include memory needed for communication, vectors, auxiliary computations, etc.

Figure J1 provides information about memory usage for Problems 1 and 2 on 1 NVIDIA V-100 GPU. The black dashed line indicates the GPU memory available on 1 GPU (V-100) on Lassen. Figure J2 provides memory use on 1 node of CTS-1 (Quartz) using 4 MPI tasks with 9 OpenMP threads each for Problem 1 and Problem 2 for increasing problem size n x n x n per MPI task.

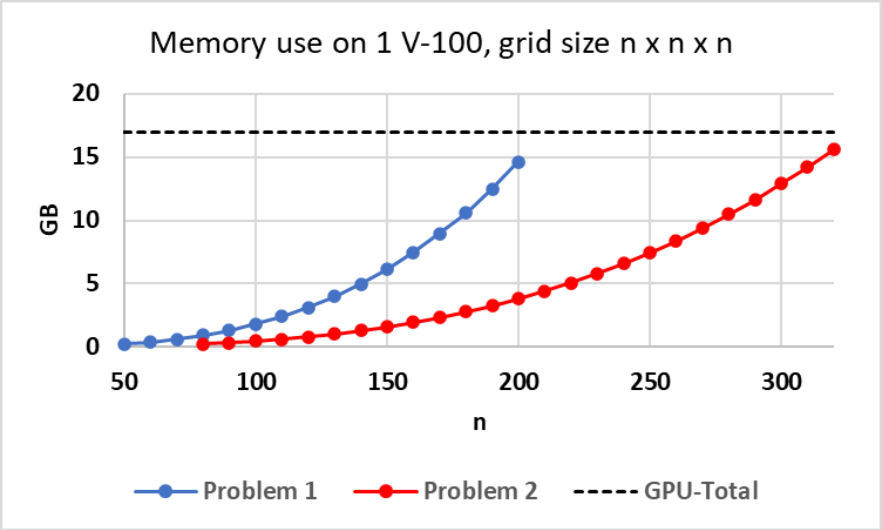

Figure J1. Approximate memory usage on 1 NVIDIA V-100 for Problem 1 (AMG-GMRES, 27pt stencil) and Problem 2 (AMG-PCG, 7pt stencil, with 1 level aggressive coarsening) for increasing problem sizes n x n x n, starting at 0.24 GB for each problem.

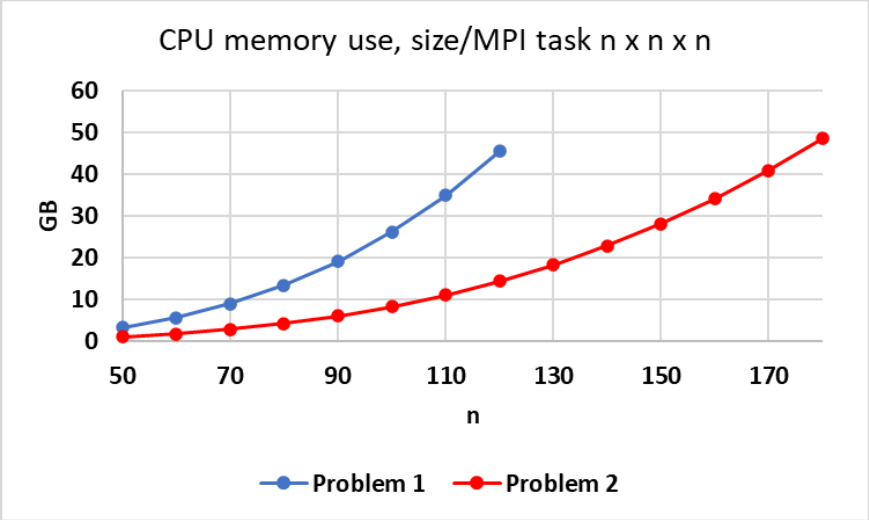

Figure J2. Approximate memory usage for Problems 1 and 2 for increasing problem size n x n x n per MPI task with 4 MPI tasks on 1 CTS-1 node.

J. Figure of Merit

The figure of merit (FOM) is calculated using the total number of nonzeros for all system matrices and interpolation operators on all levels of AMG (NNZ), AMG setup wall clock time (Setup_time), and AMG solve phase wall clock time (Solve_time).

Since in time dependent problems the AMG preconditioner might be used for several solves before it needs to be reevaluated, a parameter k has also been included to simulate computation of a time dependent problem that reuses the preconditioner for an average of k time steps.

The total FOM is evaluated as follows: FOM = NNZ / (Setup_time + k * Solve_time).

The parameter k is set to 1 in Problem 1 and to 3 in Problem 2.

K. Suggested Test Runs

We suggest the following strong scaling runs for CPUs:

The following runs were performed on RZWhippet.

A total grid size of 160x160x160 was chosen for Problem 1. The results in the first 4 columns of the table below were generated using an MPI-only configuration of hypre via

```
srun -n <P*Q*R> amg -P <P> <Q> <R> -n <nx> <ny> <nz> -problem 1
```

The results in the last 3 columns were run using a MPI/OpenMP configuration of hypre via

```
Configure –with-openmp –enable-hopscotch
```

The actual run was performed with varying numbers of OpenMP threads:

```
srun -n 1 amg -P 1 1 1 -n 160 160 160 -problem 1
```

P x Q x R	nx x ny x nz	setup time	solve time	no threads	setup time	solve time
1 x 1 x 1	160x160x160	14.92	13.85	1	15.29	13.35
2 x 1 x 1	80x160x160	10.83	6.96	2	8.21	7.08
2 x 2 x 1	80x 80x160	5.48	3.45	4	4.33	3.72
2 x 2 x 2	80x 80x 80	2.61	1.73	8	2.36	2.05
4 x 2 x 2	40x 80x 80	1.35	0.95	16	1.44	1.30
4 x 4 x 2	40x 40x 80	0.70	0.56	32	1.02	0.97
4 x 4 x 4	40x 40x 40	0.38	0.41	64	0.89	0.85
				128	1.13	0.91

We performed a similar test for Problem 2 using a total grid size of 256x256x256.

P x Q x R	nx x ny x nz	setup time	solve time	no threads	setup time	solve time
1 x 1 x 1	256x256x256	18.67	29.72	1	19.80	30.26
2 x 1 x 1	128x256x256	12.11	17.51	2	11.21	18.00
2 x 2 x 1	128x128x256	6.26	8.68	4	6.59	9.49
2 x 2 x 2	128x128x128	3.00	3.92	8	4.20	5.93
4 x 2 x 2	64x128x128	1.54	2.14	16	3.26	4.41
4 x 4 x 2	64x 64x128	0.78	1.35	32	3.12	3.88
4 x 4 x 4	64x 64x 64	0.43	1.04	64	3.84	3.66
				96	4.69	3.59
				120	5.68	4.18

We have also performed runs on 1 NVIDIA V-100 GPU increasing the problem size n x n x n.
For these runs we configured hypre as follows:

```
Configure –with-cuda
```

We increased n by 10 starting with n=50 for Problem 1 and with n=80 for Problem 2 until we ran out of memory. Note that Problem 2 uses much less memory, since the original matrix has at most 7 coefficients per row vs 27 for Problem 1. In addition, aggressive coarsening is used on the first level, significantly decreasing memory usage at the cost of increased number of iterations.

We have also included FOMs for setup and solve, which were computed as follows:

FOM_setup = NNZ / Setup time

FOM_solve = NNZ / Solve time

FOM = NNZ / (Setup time + Solve time) for Problem 1

FOM = NNZ / (Setup time + 3 * Solve time) for Problem 2.

Results for Problem 1

n	FOM	FOM_setup	FOM_solve	setup time	solve time	no iterations
50	7.135E+07	9.598E+07	2.779E+08	0.068	0.024	19
60	9.863E+07	1.357E+08	3.756E+08	0.085	0.031	19
70	1.199E+08	1.597E+08	4.811E+08	0.116	0.038	19
80	1.397E+08	2.001E+08	4.627E+08	0.138	0.060	19
90	1.629E+08	2.207E+08	6.212E+08	0.179	0.064	19
100	1.858E+08	2.576E+08	6.667E+08	0.211	0.082	19
110	2.104E+08	2.997E+08	7.062E+08	0.242	0.103	19
120	2.317E+08	3.377E+08	7.376E+08	0.280	0.128	19
130	2.496E+08	3.725E+08	7.567E+08	0.323	0.159	19
140	2.597E+08	3.910E+08	7.728E+08	0.384	0.195	19
150	2.668E+08	4.092E+08	7.666E+08	0.452	0.242	19
160	2.733E+08	4.173E+08	7.919E+08	0.539	0.284	19
170	2.827E+08	4.375E+08	7.987E+08	0.617	0.338	19
180	2.858E+08	4.432E+08	8.048E+08	0.724	0.399	19
190	2.890E+08	4.503E+08	8.070E+08	0.838	0.468	19
200	2.925E+08	4.589E+08	8.069E+08	0.960	0.546	19

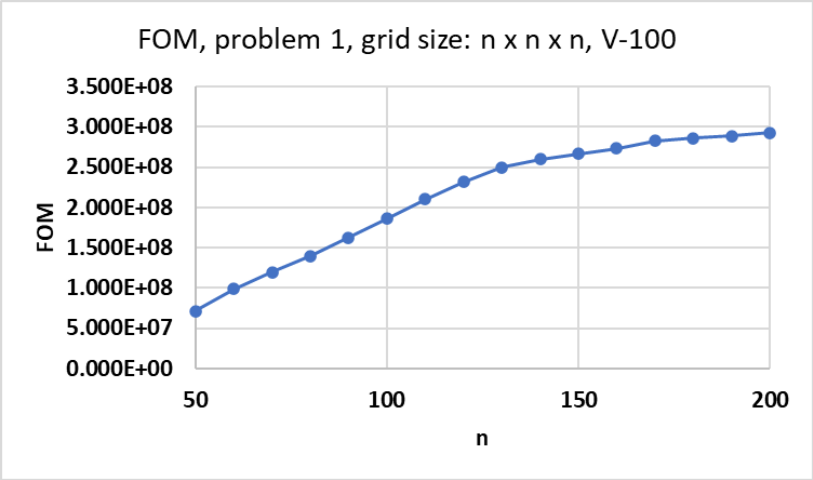

Figure K3. Total FOMs for Problem 1 on 1 NVIDIA V-100

Results for Problem 2

n	FOM	FOM_setup	FOM_solve	setup time	solve time	no iterations
80	2.931E+07	5.884E+07	1.751E+08	0.095	0.032	30
90	3.493E+07	7.006E+07	2.090E+08	0.114	0.038	29
100	4.070E+07	8.508E+07	2.341E+08	0.129	0.047	30
110	4.437E+07	9.657E+07	2.462E+08	0.151	0.059	31
120	4.511E+07	1.082E+08	2.322E+08	0.176	0.082	31
130	5.104E+07	1.139E+08	2.773E+08	0.212	0.087	31
140	5.510E+07	1.290E+08	2.886E+08	0.235	0.105	31
150	5.842E+07	1.452E+08	2.933E+08	0.256	0.127	32
160	6.075E+07	1.540E+08	3.010E+08	0.294	0.150	32
170	6.276E+07	1.748E+08	2.937E+08	0.310	0.185	33
180	6.530E+07	1.838E+08	3.038E+08	0.351	0.212	33
190	6.652E+07	1.902E+08	3.069E+08	0.398	0.247	33
200	6.823E+07	2.002E+08	3.105E+08	0.442	0.285	33
210	6.903E+07	2.074E+08	3.104E+08	0.494	0.330	33
220	6.949E+07	2.128E+08	3.096E+08	0.554	0.381	33
230	6.821E+07	2.176E+08	2.981E+08	0.619	0.452	34
240	6.856E+07	2.231E+08	2.969E+08	0.687	0.516	34
250	6.871E+07	2.279E+08	2.951E+08	0.760	0.587	34
260	6.910E+07	2.289E+08	2.969E+08	0.852	0.656	34
270	6.801E+07	2.308E+08	2.892E+08	0.946	0.755	35
280	6.825E+07	2.353E+08	2.884E+08	1.035	0.845	35
290	6.916E+07	2.398E+08	2.915E+08	1.129	0.929	35
300	6.932E+07	2.411E+08	2.919E+08	1.244	1.027	35
310	6.955E+07	2.425E+08	2.926E+08	1.365	1.131	35
320	6.978E+07	2.442E+08	2.931E+08	1.490	1.242	35

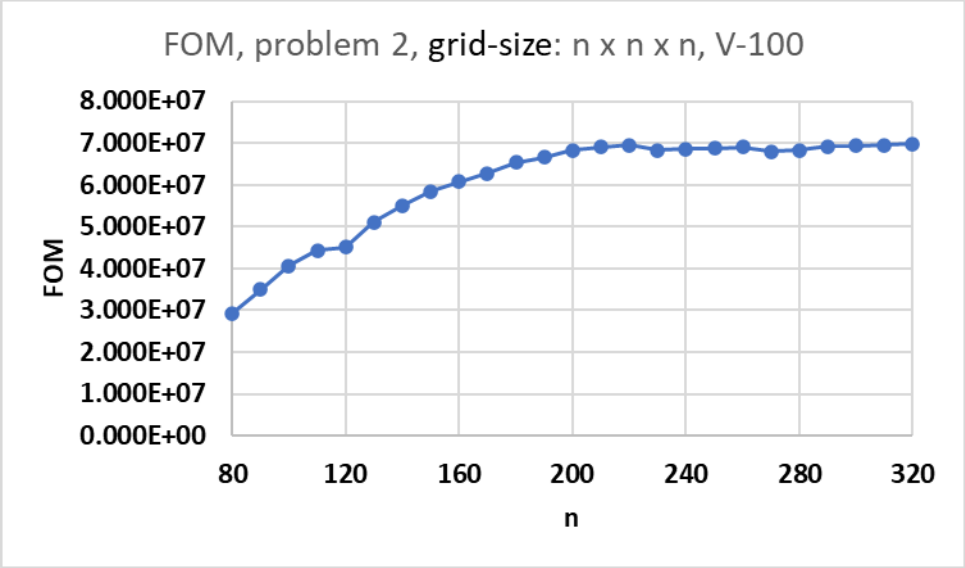

Figure K4. Total FOMs for Problem 2 on 1 NVIDIA V-100

For further information on AMG2023 contact
Ulrike Yang
email: yang11@llnl.gov
or
Rui Peng Li
Email: li50@llnl.gov