CMAQ 5.2.1 PARALLEL PERFORMANCE WITH MPI AND OPENMP**

George Delic*
HiPERiSM Consulting, LLC, P.O. Box 569, Chapel Hill, NC 27514, USA

1. INTRODUCTION

This presentation reports on implementation of the parallel sparse matrix solver, FSparse [1], in the Chemistry Transport Model (CTM) in CMAQ. In this report performance results of the original EPA [2] and FSparse versions are presented for the GEAR version of the CTM. This release is v6.3 and is a major redesign. It is applicable in CMAQ with either the Rosenbrock (ROS3) or SMV Gear (GEAR) algorithms in the CTM. In FSparse different blocks of cells are distributed to separate threads in the parallel thread team.

2. TEST BED ENVIRONMENT

2.1 Hardware

The hardware systems chosen were the platforms at HiPERiSM Consulting, LLC, shown in Table 2.1. Nodes 20 and 21 host two Intel E5v3 CPUs with 16 cores and each node has four Intel Phi co-processor many integrated core (MIC) cards [3] with 60 and 59 cores, respectively. These are the base nodes of a heterogeneous cluster that includes a HP blade server [4] hosting nodes 27 to 34 with dual 4-core Intel E5640 CPUs. The total core count of this cluster is 128 with ~2 Tflops (peak) in single precision. For the standard U.S. EPA version the MPI executions are launched across multiple combination of these nodes. This cluster allows for comparison of the FSparse hybrid (MPI + OpenMP) parallel versions of CMAQ with the original EPA version.

2.2 Compilers

Results reported here implemented the Intel Parallel Studio® suite (release 17.6, [3]) and Portland Group compiler (release 18.1, [5]) with compiler options for a heterogeneous cluster. In the Portland case, utilization of an Intel wrapper enabled linking to the Intel MPI library.

**Corresponding author:* George Delic, george@hiperism.com.

** Dedicated to the memory of Jeff Young who provided invaluable advice over several decades.

2.3 Episode studied

This report used the benchmark test data available in the CMAQ 5.2.1 download for a single day (24 hour) episode. This episode was for July 1st, 2011, using the cb6r3_ae6_aq mechanism with 149 active species and 329 reactions. For day/night chemistry this results in 1338/1290 non-zero entries in the Jacobian matrix. The episode was run for a full 24 hour scenario on a 80 X 100 California domain at 12 Km grid spacing and 35 vertical layers for a total of 280,000 grid cells. In this report a variable number of MPI processes (NP) were used in both CTM versions with 8, 12, and 16 threads (OMP) in the OpenMP case.

Table 2.1. CPU platforms at HiPERiSM Consulting, LLC

Platform	Node20-21 (each node)	Node27-34 (each node)
Operating system	OpenSuSE 13.2	OpenSuSE 42.3
Processor	Intel™ x86-64 (E5-2698v3)	Intel™ x86-64 (E5640)
Coprocessor	4 x Intel Phi 7120/5120	NA
Peak Gflops / CPU (SP/DP)	~589 (SP)	~ 43 (DP)
Power consumption	135 Watts	80 Watts
Cores per processor	16	4
Power per core	8.44 Watts	20 Watts
Processor count	2	2
Total core count	32	8
Clock	2.3 GHz	2.67 GHz
Bandwidth	68 GB/sec	25.6 GB/sec
Bus speed	2133 MHz	2933 MHz
L1 cache	16x32 KB	4x32 KB
L2 cache	16x256 KB	4x256 KB
L3 cache	40 MB	12 MB

In the following two performance metrics are introduced to assess parallel performance in the MPI and OpenMP modified code:

(a) *Speedup* is the gain in runtime over the standard U.S. EPA version,
(b) *Scaling* is the gain in runtime with MPI process or thread counts larger than 1,

relative to the result for a single MPI process or thread on the host CPU.

2.4 Interconnect fabric

Results reported here used this configuration:

1. Homogeneous cluster consisting of node20 & 21 connected via a 10GigE switch.
2. Heterogeneous cluster consisting of node 20 & 21 and the HP blade. The blade chassis has an internal switch connecting node27-34 and uplinks to the 10GigE switch to join all nodes together.

For MPI traffic in either cluster mode bandwidth is approximately 10G bits/sec. A pending upgrade to an Infiniband (IB) fabric will raise bandwidth to a (theoretical) limit of 40G bits/sec.

3. RESULTS FOR TWO CMAQ MODELS

3.1 Performance profile of CMAQ

This section repeats the profile results of the standard CMAQ 5.2.1 distribution in the testbed environment identified in Section 2. The optimization level with the Intel compiler was "-O2" because higher optimizations caused segmentation faults at runtime. This could have been caused by (as yet) unresolved code bugs in CMAQ, or the Intel compiler itself. Since the previous report, several compiler bugs were corrected, but not all have been resolved to-date, (as with some issues within CMAQ itself). In addition, several issues in the thread parallel version of CMAQ were corrected.

For a profile of where time is consumed Fig. 3.1 compares the fraction of total wall clock time expended in the dominant science processes in CMAQ. The CHEM process is the Gear (GEAR) version of the CTM. The EPA version is compared with the FSparse threaded version for 8, 12, and 16 OpenMP threads, as identified in the legend. As the fraction of time in CHEM decreases the fraction of time in the other science processes increases.

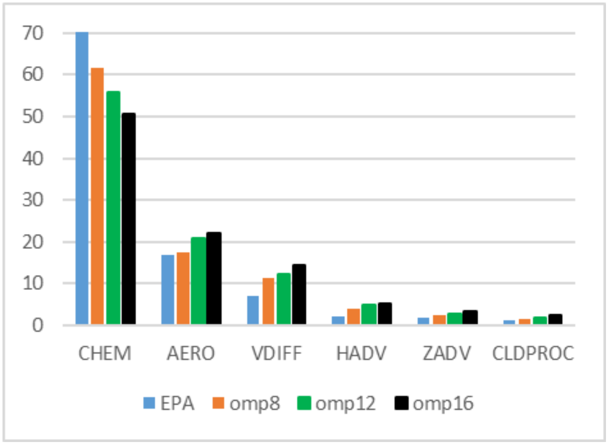

Fig 3.1: Fraction of wall clock time (percent) by science process for the FSparse GEAR version of CMAQ for NP=1 and OpenMP thread counts of 8, 12, and 16.

3.2 Intel compiler MPI performance

For the case of the Intel compiler Table 3.1 shows wall clock time, MPI scaling, and parallel efficiency for the EPA GEAR CTM solver, with various values of NP in the range 1 to 64 on nodes 20 and 21 (Table 2.1). These two nodes form a homogeneous sub-cluster consisting of the two fastest nodes. Note that the MPI parallel efficiency declines to 49% with NP=64. This loss in parallel efficiency is due to the diminished work load per MPI process with a domain of 280,000 cells. Partitioning amongst the available number of MPI processes (after division into blocks of 50 cells) gives 280,000/50 = 5600 blocks for NP = 1, and 5600 / NP thereafter, when NP > 1.

Table 3.1. MPI wall clock time (seconds), scaling (relative to NP=1), and MPI parallel efficiency for the U.S. EPA version of CMAQ with the GEAR solver using the Intel compiler on nodes 20 and 21.

NPROW X NPCOL	GEAR solver algorithm		
	Wall clock time	MPI scaling	MPI efficiency
1	20066	1.0	1.00
2x2=4	5272	3.8	0.95
4x2=8	2813	7.1	0.89
4x4=16	1612	12.4	0.78
8x4=32	1001	20.0	0.63
8x8=64	645	31.1	0.49

Results with the Portland compiler are limited for the U.S. EPA version of CMAQ because of a runtime failure due to a compiler bug that fails to pass bounds on an array by reference to a subroutine.

4. RESULTS FOR THE OpenMP MODEL

4.1 FSparse GEAR speedup versus EPA

In FSparse an OpenMP modification was implemented in the standard CMAQ version of the CTM procedure since the dominant amount of time is expended there for the GEAR solver. Performance results using the Intel compiler are presented in this section.

Fig. 4.1 shows wall clock time (in seconds) for various combinations of MPI processes for EPA and FSparse version of CMAQ on the heterogeneous cluster. Due to the core count limit on each blade, the use of only 8 OpenMP threads was the default. Fig. 4.2 shows the corresponding speedup of the OpenMP version over the EPA release. In both cases the horizontal axis shows the number of MPI processes, and the multiple values are for differing combinations of participating nodes of the cluster. Variability for any fixed value of NP is determined by how many processes are resident on the fastest nodes (20 and 21), versus the number on the slower nodes (27 to 34).

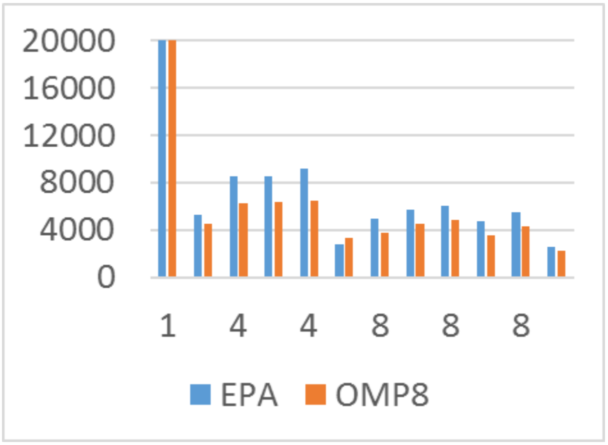

Fig. 4.1 Wall clock time (seconds) for the GEAR solver in the standard U.S. EPA and the 8 thread OpenMP versions of CMAQ for NP=1 to 16 using the Intel compiler for various combinations of nodes.

The two examples where the EPA version is faster correspond to MPI processes divided between node20 and 21 of the homogeneous cluster. In this case MPI communication is dominantly through memory.

Fig. 4.3 shows thread speed up over the U.S. EPA times for 288 calls to the CTM with NP=1 MPI processes. Note the 1.5 times performance boost with 16 threads compared to the 8 thread result.

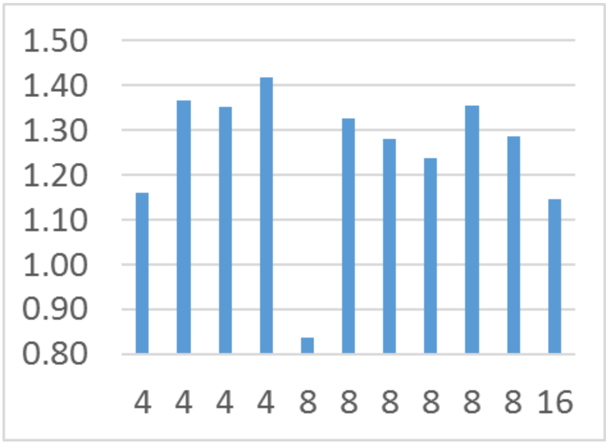

Fig. 4.2 Corresponding to Fig. 4.1 this shows OpenMP speedup with 8 threads for the GEAR solver in the thread parallel version over the standard U.S. EPA model using the Intel compiler for various combinations of nodes.

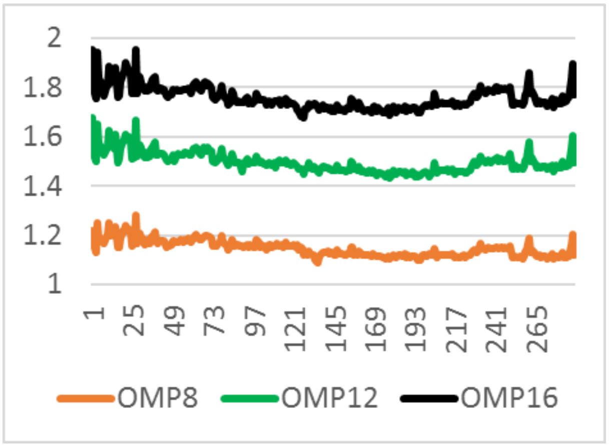

Fig. 4.3 Parallel thread speedup over the standard U.S. EPA model using the Intel compiler in 288 calls to CHEM with GEAR solver for 8, 12 and 16 threads, for NP=1 MPI process in the 24 hour episode.

4.2 MPI parallel efficiency

Another metric of performance is parallel efficiency. In the MPI case this is defined as:

$$Scaling_{(n)} = time\ (NP=1) \,/\, time\ (NP=n)$$
$$Efficiency_{(n)} = Scaling_{(n)} \,/\, n$$

This was introduced in the last column of Table 3.1 for the original EPA version of CMAQ on the homogeneous cluster of node20 and 21.

Intel trace tools allow instrumentation that measures the fraction of run time expended in MPI message passing versus computation. In the latter case trace instrumentation also separates scalar and thread parallel computation fractions. Fig. 4.4 shows this summary for the FSparse version of CMAQ. The dominant fraction is the time spent in MPI message passing at larger NP values. The exceptions are for cases when MPI processes reside on the fastest nodes where MPI communication is predominantly via on-node memory, and not the interconnect fabric.

Fig. 4.5 shows MPI scaling for EPA and FSparse versions of CMAQ. The variability could be a consequence of either (a) the 10GigE interconnect between nodes (see discussion in Section 2.4), or (b) slower processors on the blade servers. Nevertheless the OpenMP version delivers enhanced scaling. In all cases (with one exception) the FSparse version shows greater MPI speedup.

Fig 4.6 shows a regression plot of fraction of run time spent (percent) in OpenMP and scalar computation regions versus the fraction (percent) in MPI communication. This demonstrates three features:

- The fraction of time in MPI communication increases as the fraction of time in computation decreases
- The scalar computation fraction dominates whenever the MPI communication fraction is greater than 12%
- The OpenMP computation fraction dominates when the MPI communication fraction is less than 12%

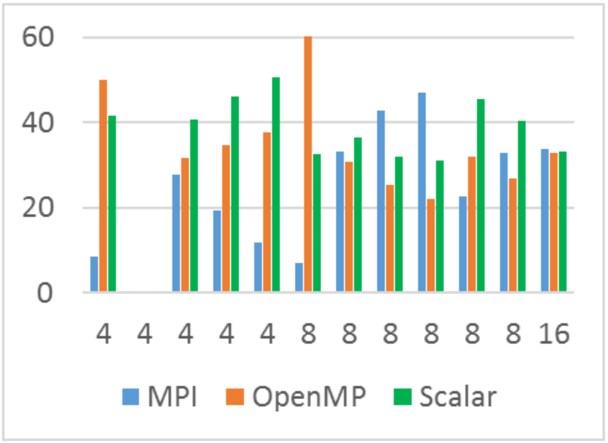

Fig. 4.4 For the 8 thread FSparse version of CMAQ this compares the fraction of total time (percent) spent in MPI message passing, OpenMP computation region, and scalar computation region with the Intel compiler for various combinations of nodes.

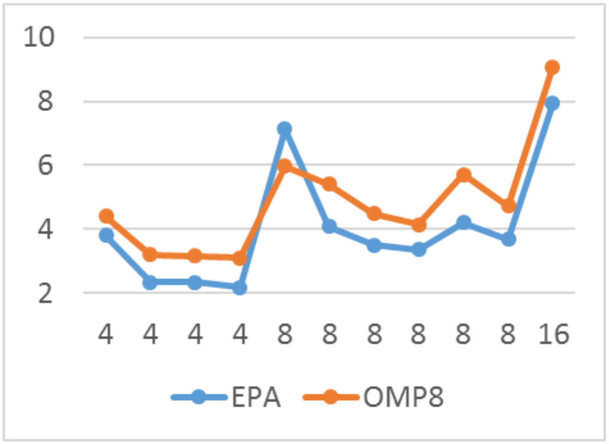

Fig. 4.5. MPI scaling versus number of processes for EPA and 8 thread FSparse versions of CMAQ of the GEAR solver with the Intel compiler for various combinations of nodes.

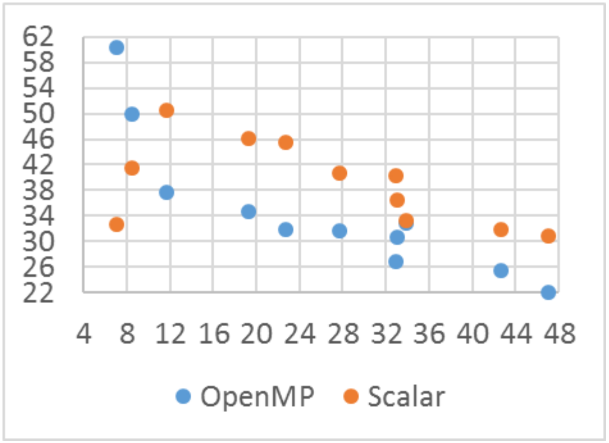

Fig. 4.6. For the 8 thread FSparse version of CMAQ this shows the percent of total time spent in the OpenMP and scalar regions (vertical) verses that in MPI communication (horizontal) with the Intel compiler for various combinations of nodes.

The negative regression trend in Fig. 4.6 for both OpenMP and scalar computation fractions is the result of the decreasing domain size per MPI process (as discussed in Section 3.2).

4.3 Portland versus Intel Compiler

Table 4.1 lists wall clock time for CMAQ in the FSparse OpenMP version using 8 threads with the number, NP, of MPI processes shown in the first column. Results of Intel and Portland compilers are compared.

Table 4.1. MPI wall clock time (seconds) on the homogeneous cluster for the 8 thread FSparse version of CMAQ with the GEAR solver using the Intel and Portland compilers.

NPROW X NPCOL =NP	GEAR solver algorithm (wall clock time)		
	Intel	Portland	Portland / Intel
1	20059	18110	0.90
2x2=4	5272	5920	1.12
4x2=8	2813	3991	1.42

5. SUMMARY OF KEY POINTS

5.1 MPI communication

- MPI communication time tends to compete with, or exceed, computation time.
- MPI scaling increases with process count
- Corresponding to this MPI parallel efficiency decreases

5.2 Computation

- CHEM is by far the dominant science process in CMAQ computation time
- Scalar computation time dominates, or competes with, OpenMP computation time
- The scalar computation fraction (of total time) dominates whenever the MPI communication fraction is greater than 12%
- The OpenMP computation fraction dominates when the MPI communication fraction is less than 12%

6. NUMERICAL ISSUES

6.1 Comparing concentration values

Any discrepancy between predictions of the JSparse [2] and FSparse [1] algorithms in the two methods is explained by the way precision is treated in each. The CTM solver uses double precision arithmetic but accepts some input data from single precision variables (temperature, pressure, photolysis rates, reaction rates, etc.). Therefore all expressions in FSparse are performed in double precision. The acid test is to compare the computed concentration values for selected species as predicted by the EPA (using JSparse) and the thread parallel version (using FSparse).

Concentration values for 11 selected species (O3, NOx, etc) were compared for the entire domain and all 24 time steps. Examples of such a comparison are shown in Figs. 6.1 to 6.4, where the absolute error is shown as the difference in predicted concentrations. To within the tolerances required in GEAR (ATOL=1.0e-09, RTOL=1.0E-03), agreement was observed for both Intel and Portland compilers.

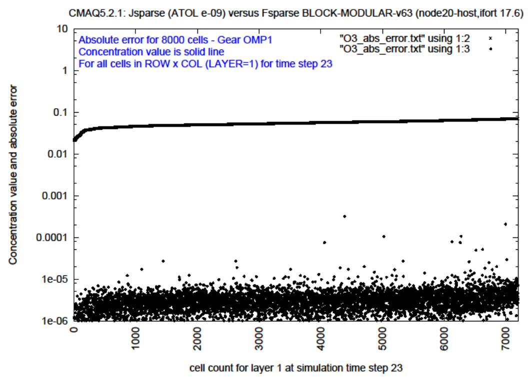

Fig 6.1: Intel compiler results for the FSparse GEAR solver of CMAQ with 8 OpenMP threads. This shows the O3 species concentration absolute error (scattered points) and concentration value (solid line) for 8000 values in layer 1 of the domain. The ranking is in increasing concentration value from left to right.

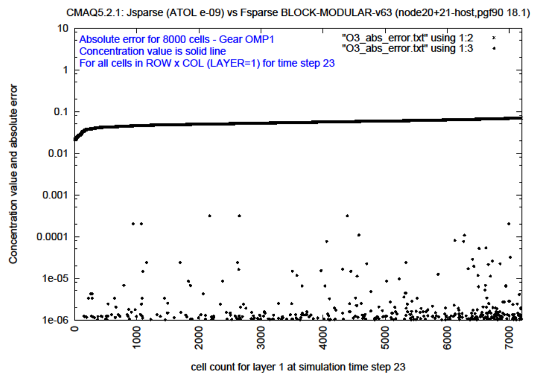

Fig 6.2: Portland compiler results for the FSparse GEAR solver of CMAQ with 8 OpenMP threads. This shows the O3 species concentration absolute error (scattered points) and concentration value (solid line) for 8000 values in layer 1 of the domain. The ranking is in increasing concentration value from left to right.

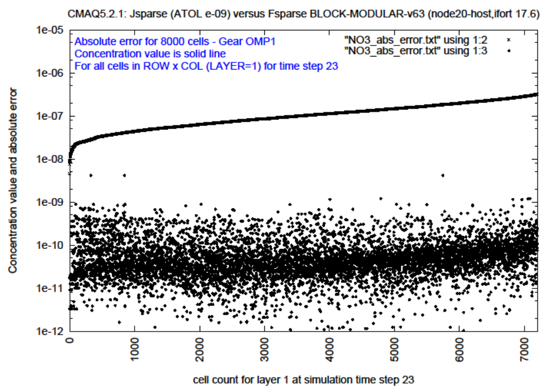

Fig 6.3: Intel compiler results for the FSparse GEAR solver of CMAQ with 8 OpenMP threads. This shows the NO3 species concentration absolute error (scattered points) and concentration value (solid line) for 8000 values in layer 1 of the domain. The ranking is in increasing concentration value from left to right.

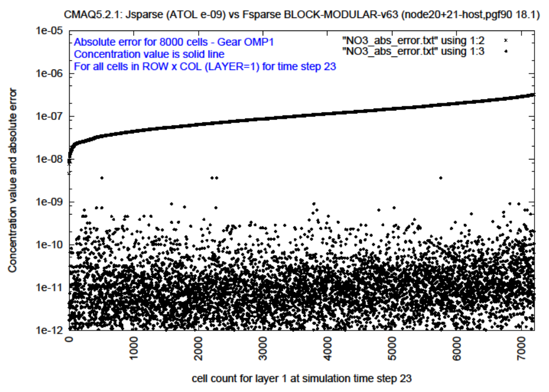

Fig 6.4: Portland compiler results for the FSparse GEAR solver of CMAQ with 8 OpenMP threads. This shows the NO3 species concentration absolute error (scattered points) and concentration value (solid line) for 8000 values in layer 1 of the domain. The ranking is in increasing concentration value from left to right.

7. LESSONS LEARNED

7.1 Benefits of the FSparse method

Comparing runtime performance for CMAQ 5.2.1 in the new OpenMP parallel version with the U.S. EPA release showed benefits such as:

- Thread speedup as high as 1.42 with 8 threads
- Hybrid MPI+OpenMP algorithms that offer more on-node compute intensity as the number of available threads rises
- Numerical values of predicted species concentration that are within the error tolerance inherent in the algorithms.

7.2 Next steps

A continuation of this work would include:

- Repeat of selected cases with the IB interconnect upgrade.
- Extension to the Rosenbrock version of CMAQ.
- Comparison of EBI, Rosenbrock, and GEAR solver versions of CMAQ.
- Further compiler comparisons of Intel and Portland results.

8. CONCLUSIONS

This report has described an analysis of CMAQ 5.2.1 behavior in the standard U.S. EPA release and a new thread parallel version of CMAQ suitable for the Rosenbrock and Gear solvers. In this version (v6.3) subroutines common to both algorithms have been successfully developed and tested in the Gear case.

The new FSparse version of CMAQ offers layers of parallelism not available in the standard U.S. EPA release and is portable across multi- and many-core hardware and compilers that support thread parallelism.

REFERENCES

[1] Delic, G., 2016: see presentation at the Annual CMAS meeting (http://www.cmasecenter.org).

[2] Jacobson, M. and Turco, R.P., (1994), Atmos. Environ. 28, 273-284.

[3] INTEL: Intel Corporation, http://www.intel.com

[4] https://en.wikipedia.org/wiki/HP_BladeSystem

[5] PGI Compilers & Tools https://www.pgroup.com