CMAQ 5.1 PERFORMANCE WITH PORTLAND AND INTEL COMPILERS

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, in the Chemistry Transport Model (CTM) in CMAQ [1]. This release is v6.2 and is a major redesign. It is applicable in the CMAQ version that uses either the Rosenbrock (ROS3) or SMV Gear (GEAR) [2] algorithms in the CTM. In FSparse different blocks of cells are distributed to separate threads in the parallel thread team. Performance results of the original and FSparse versions are presented. Species concentration values were compared for original and FSparse methods with some comments on numerical analysis, and error tolerances.

2. TEST BED ENVIRONMENT

2.1 Hardware

The hardware systems chosen were the platforms at HiPERiSM Consulting, LLC, shown in Table 2.1. Each of the two nodes host two Intel E5v3 CPUs with 16 cores each. For the standard U.S. EPA version the MPI executions are across both nodes. Each node has, in addition, four Intel Phi co-processor many integrated core (MIC) cards with 60 and 59 cores for the respective models. This combination allows for testing of the FSparse hybrid parallel versions of CMAQ on either host or first generation Intel Phi processor [3]. In this case the thread parallel region of the CTM is offloaded to the Phi processors.

2.2 Compiler

Most results reported here implemented the Intel Parallel Studio® suite (release 17.0) using options for either host CPU or Phi coprocessor. The latter required code modification to identify MIC attributes within a single source code. The extensive reporting options were used to investigate optimization effectiveness. A limited number of executions are also reported for the Portland compiler (release 15.7).

2.3 Episode studied

This report used the benchmark test data available in the CMAQ 5.1 download. This model episode was for July 1st, 2011, using the cb05e51_ae6 mechanism with 147 active species and 343 reactions. For day/night chemistry this results in 1224/1158 non-zero entries in the Jacobian matrix. The episode was run for a full 24 hour scenario on a 100 X 72 California domain at 12 Km grid spacing and 35 vertical layers for a total of 252,000 grid cells. This domain is some ten times smaller than that reported previously in [4]. In this report a variable number of MPI processes (NP) were used in the EPA version of CMAQ and only NP=1 in the OpenMP version.

Table 2.1. CPU platforms at HiPERiSM Consulting, LLC

Platform	Node20	Node21
Operating system	SuSE Linux 13.2	SuSE Linux 13.2
Processor	Intel™ IA32 (E5-2698v3)	Intel™ IA32 (E5-2698v3)
Coprocessor	4 x Intel Phi 7120	4 x Intel Phi 5110
Peak Gflops (SP/DP)	589 (SP)	589 (SP)
Power consumption	135 Watts	135 Watts
Cores per processor	16	16
Power per core	8.44 Watts	8.44 Watts
Processor count	2	2
Total core count	32	32
Clock	2.3 GHz	2.3 GHz
Bandwidth	68 GB/sec	68 GB/sec
Bus speed	2133 MHz	2133 MHz
L1 cache	16x32 KB	16x32 KB
L2 cache	16x256 MB	16x256 KB
L3 cache	40 MB	40 MB

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

(a) *Speedup* is the gain in runtime over the standard U.S. EPA version,

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

(b) *Scaling* is the gain in runtime with thread counts larger than 1, relative to the result for a single thread on the host CPU, or lowest thread count on the Intel Phi.

3. RESULTS FOR THE STANDARD MODEL

3.1 Profile of CMAQ on host

This section repeats the previous year's [1] profile results of the standard CMAQ 5.1 distribution in the testbed environment identified in Section 2. The optimization level with the Intel compiler was "-O2" because higher optimizations caused segmentation faults (segfaults) 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 all three CTM solvers with 1 MPI process. This report will focus on the Rosenbrock (ROS3) and Gear (GEAR) versions because they share common procedures and offer the best opportunity for efficient parallel thread tasking.

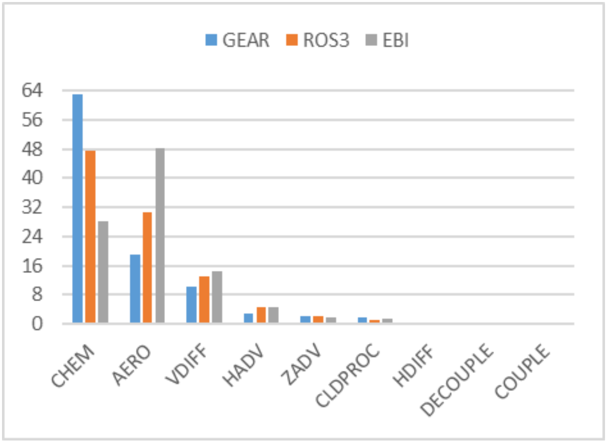

Fig 3.1: Fraction of wall clock time (percent) by science process for EBI, ROS3, and GEAR versions of CMAQ for NP=1. Note that CHEM is not the dominant process for the EBI case.

3.2 MPI performance on host with the Intel compiler

The totals of wall clock time for ROS3 and GEAR CTM solvers, with various values of NP, is shown in Table 3.1 and Fig. 3.2. The combination of MPI processes, NP = NPROW x NPCOL, is in the range 1 to 64, with doubling of row and column processes.

Table 3.1. Wall clock times (in seconds) and ratio for the U.S. EPA version of CMAQ on Intel host CPUs with ROS3 and GEAR solvers using the Intel compiler.

NPROW X NPCOL	CTM solver algorithm		
	ROS3	GEAR	GEAR/ROS3
1	14403	18638	1.29
4	3965	5595	1.41
16	1341	1733	1.29
64	623	761	1.22

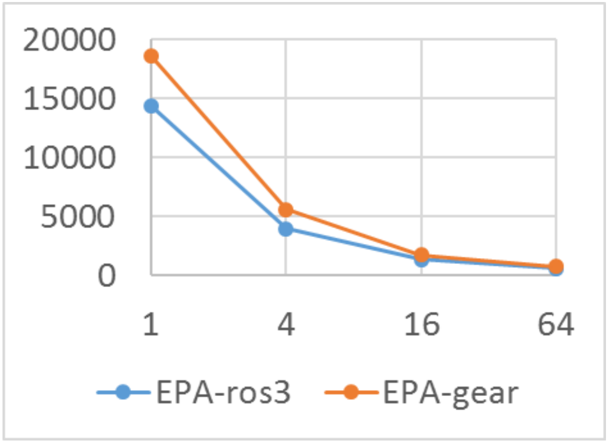

Fig. 3.2 Wall clock time (seconds) for ROS3, and GEAR solvers in the standard U.S. EPA version of CMAQ for NP=1 to 64 using the Intel compiler.

However, as shown in Tables 3.2 and 3.3, the parallel efficiency declines to ~67% when NP=16, and ~36% when NP=64. This loss in parallel efficiency is due to the diminished work load per MPI process with a domain of 252,000 cells. Partitioning amongst the available number of MPI processes, after division into blocks of 50 cells gives 252,000/50 = 5040 blocks for NP = 1, and 5040 / NP thereafter, when NP > 1. As noted previously [1], the time consumed in MPI procedures increases substantially with larger NP.

Table 3.2. MPI scaling and parallel efficiency for the U.S. EPA version of CMAQ on Intel host CPUs with the ROS3 solver using the Intel compiler.

NPROW X NPCOL	ROS3 solver algorithm	
	MPI scaling	MPI efficiency
1	1.00	1.00
4	3.63	0.91
16	10.74	0.67
64	23.12	0.36

Table 3.3. MPI scaling and parallel efficiency for the U.S. EPA version of CMAQ on Intel host CPUs with the GEAR solver using the Intel compiler.

NPROW X NPCOL	GEAR solver algorithm	
	MPI scaling	MPI efficiency
1	1.00	1.00
4	3.33	0.83
16	10.75	0.67
64	24.47	0.38

3.3 MPI performance on host with the Portland compiler

The Portland compiler was used to compile CMAQ and all associated dependencies with complier switches "-O4 –fastsse". The results of the standard U.S. EPA CMAQ version are summarized in Tables 3.4 to 3.6. In view of the superior timing results with the Intel compiler, no further effort was expended in using the Portland compiler at this time.

Table 3.4. Wall clock times (in seconds) and ratio for the U.S. EPA version of CMAQ on Intel host CPUs with ROS3 and GEAR solvers using the Portland compiler.

NPROW X NPCOL	CTM solver algorithm		
	ROS3	GEAR	GEAR/ROS3
1	23900	27739	1.16
4	6449	8326	1.29
16	2375	2680	1.13

Table 3.5. MPI scaling and parallel efficiency for the U.S. EPA version of CMAQ on Intel host CPUs with the ROS3 solver using the Portland compiler.

NPROW X NPCOL	ROS3 solver algorithm	
	MPI scaling	MPI efficiency
1	1.00	1.00
4	3.71	0.93
16	10.06	0.63

Table 3.6. MPI scaling and parallel efficiency for the U.S. EPA version of CMAQ on Intel host CPUs with the GEAR solver using the Portland compiler.

NPROW X NPCOL	GEAR solver algorithm	
	MPI scaling	MPI efficiency
1	1.00	1.00
4	3.33	0.83
16	10.35	0.65

4. OpenMP MODEL ON THE HOST

4.1 ROS3 and GEAR speedup versus EPA

An OpenMP modification (as described in Section 6) was implemented in the standard CMAQ version of the CTM procedure since the dominant amount of time is expended there for ROS3 and GEAR solvers (see Fig. 3.1). Performance results using the Intel compiler are presented in this section (and the next).

Table 4.1. Wall clock times (in seconds) and ratio for the FSparse thread parallel version of CMAQ on the host CPU with ROS3 and GEAR solvers with 1 MPI process.

Thread count	CTM solver algorithm		
	ROS3	GEAR	GEAR/ROS3
1	18413	27489	1.49
4	13199	14168	1.07
8	11446	11614	1.01
12	10896	11561	1.06
16	10453	10680	1.01

Table 4.1 lists wall clock time and Fig. 4.1 shows speedup versus thread count on the host CPU of the OpenMP parallel FSparse version over the standard U.S. EPA release of CMAQ. With 4 to 16 threads the speedup over the standard EPA version ranges from 1.1 to 1.38 for ROS3 and 1.3 to 1.75 for GEAR. The enhancement for the GEAR algorithm is due to more work per thread when compared to ROS3. The diminution of performance gain with higher thread counts is due to the smaller partitions of work per thread calculated from 5040 blocks of cells divided amongst the number of available threads. Grid cells are partitioned into blocks of size 50 and these blocks are distributed to threads in a thread team in the OpenMP version.

Also noteworthy from Table 4.1 is the observation that the wall clock time with 8 threads is nearly identical for either CTM solver algorithm. This suggests that the superior GEAR algorithm is therefore to be preferred in production use.

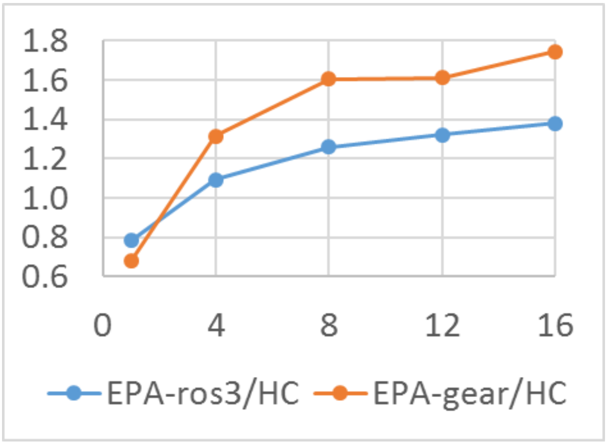

Fig 4.1: With one MPI process on the host CPU this shows the speedup of the thread parallel FSparse CMAQ version over the U.S. EPA standard release for 1 to 16 threads, with ROS3, and GEAR solvers.

5. OpenMP MODEL ON THE Phi

5.1 ROS3 and GEAR speedup versus EPA

The same OpenMP modification (as described in Section 6) was implemented in the standard CMAQ version of the CTM procedure with offload of the thread parallel region to the Intel Phi 7120 co-processor in ROS3 and GEAR solvers.

Table 5.1. Wall clock times (in seconds) and ratio for the FSparse thread parallel version of CMAQ on the Intel Phi 7120 with ROS3 and GEAR solvers with 1 MPI process using the Intel compiler

Thread count	CTM solver algorithm		
	ROS3	GEAR	GEAR/ROS3
60	12260	16610	1.35
120	11285	15432	1.37
180	11060	15723	1.42
240	11099	16035	1.44

Table 5.1 lists wall clock time and Fig. 5.1 shows speedup versus thread count on the Intel Phi processor of the OpenMP parallel FSparse version over the standard U.S. EPA release of CMAQ. With two vector processing units (VPU) per core on the Intel Phi 7120, there is a saturation visible with more than 120 threads. An additional consideration is that 5040 blocks are partitioned over a larger thread team: e.g. 5040/120, 5040/180, and 5040/240, thereby reducing the workload per thread.

5.2 Comparison of host CPU and Phi

For host CPU and Phi cases the speedup of the thread parallel version FSparse over the standard release of CMAQ increases monotonically with thread count. However, there are differences between ROS3 and GEAR results. Possible reasons include different data movement patterns, especially for the offload to the Phi. Another contributing factor is the difference in iteration patterns and their number.

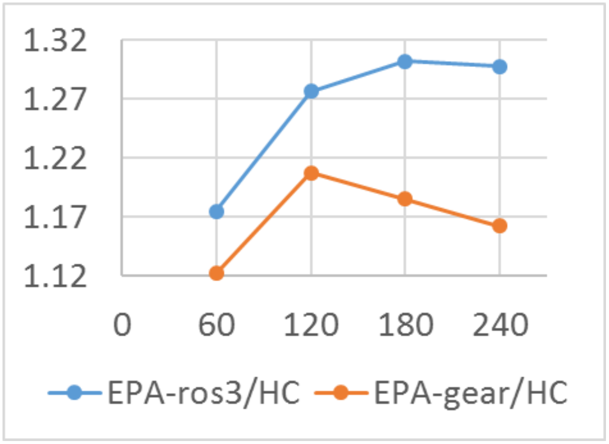

Fig 5.1: With one MPI process on the Intel Phi 7120 this shows the speedup of the thread parallel FSparse CMAQ version over the U.S. EPA standard release for 60 to 240 threads, with ROS3, and GEAR solvers.

6. NUMERICAL AND CODE ISSUES

6.1 Parallel thread code

To briefly describe the modifications to the U.S. EPA version of CMAQ, Tables 6.1 and 6.2 list the affected procedures and their replacement in the thread parallel versions. In version v6.2 of FSparse the thread parallel region in the CTM has a sequence of calls to the procedures listed below the driver. Comparing ROS3 and GEAR versions, while the standard version of CMAQ uses different procedure names, they share the same like-named procedure listed in the FSparse columns of the Tables (with only minor differences – if any).

Table 6.1. Procedures in the U.S. EPA and FSparse versions of the ROS3 solver.

EPA	FSparse v6
`rbdriver.F`	`rbdriver-v62.F`
`rbsolver.F`	`(included above)`
`rbdata_mod.F`	`fsdata_mod-mic.F`
`rbdecomp.F`	`fsdecomp-v62.F`
`rbfeval.F`	`fsfeval-v62.F`
`rbinit.F`	`fsinit.F`
`rbjacob.F`	`fsjacob-v62.F`
`rbsolve.F`	`fssolve-v62.F`
`rbsparse.F`	`fsparse.F`
	`fsrconst-v62.F`

Table 6.2. Procedures in the U.S. EPA and FSparse versions of the GEAR solver.

EPA	FSparse v6
`grdriver.F`	`grdriver-v62.F`
`grsmvgear.F`	`(included above)`
	`fsdata_mod-mic.F`
`grdecomp.F`	`fsdecomp-v62.F`
`grsubfun.F`	`fsfeval-v62.F`
`grinit.F`	`fsinit.F`
`grpderiv.F`	`fsjacob-v62.F`
`grbacksub.F`	`fssolve-v62.F`
`grsprse.F`	`fsparse.F`
	`fsrconst-v62.F`

All variables referenced in the thread parallel region are partition into shared or private categories. Furthermore, they also need to be flagged in an `OPTIONS` directive with the `offload_attribute_target=mic` attribute. Likewise, all functions referenced in the thread parallel region also need the `OFFLOAD:mic` attribute in a directive. Thus, where they occur, such changes have been applied to:

```
RXNS_FUNC_MODULE.F90
RXNS_DATA_MODULE.F90
GRID_CONF.F
GRVARS.F
```

The resulting FSparse method code is the same for either host or Phi targets and each target is selected by a compiler switch.

6.2 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 Chemistry 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). Examples of such a comparison are shown in Figs. 6.1 and 6.2, where that absolute error shown is the difference in predicted concentrations.

During debugging it was essential to carefully inspect concentration values for 10 selected species (O3, NOx, etc) for the entire domain and all 24 time steps. To within the tolerances required in ROS3 (ATOL=1.0e-07) and GEAR (ATOL=1.0e-09), agreement was observed. In fact there was negligible difference in species values in the GEAR algorithm when ATOL=1.0e-08 is used (see Fig. 6.2).

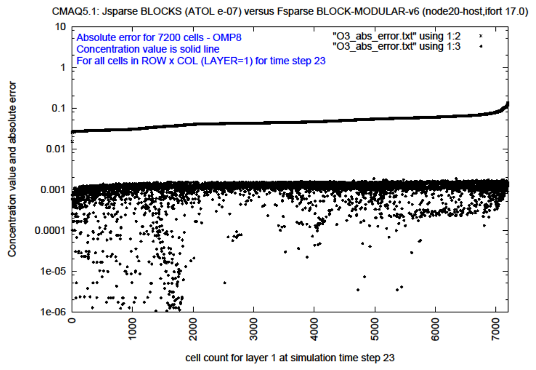

Fig 6.1: For the FSparse ROS3 solver of CMAQ on the host CPU (with 8 OpenMP threads and ATOL=10.0e-07) this shows the O3 species concentration absolute error (scattered points) and concentration value (solid line) for 7200 values in layer 1 of the domain. The ranking is in increasing concentration value from left to right.

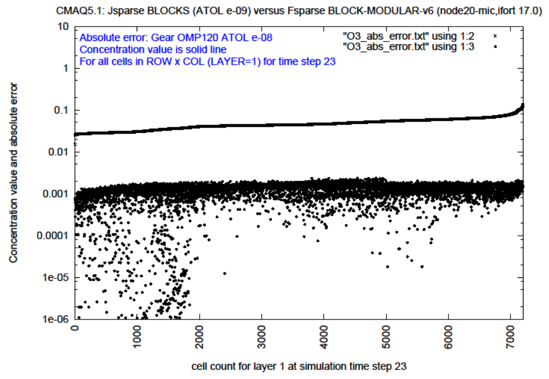

Fig 6.2: For the FSparse GEAR solver of CMAQ on the Intel Phi (with 120 OpenMP threads and ATOL=10.0e-08) this shows the O3 species concentration absolute error (scattered points) and concentration value (solid line) for 7200 values in layer 1 of the domain. The ranking is in increasing concentration value from left to right.

6.3 Code issues uncovered in this study

Several code issues were uncovered during this implementation using the CMAQ 5.1 download of the original EPA version from the CMAS site. These included the following:

- In `bldit.cctm` script the preprocessor uses a reserved name in "`set PAR=( -Dparallel )`" that corrupts OpenMP directives.
- More than 60,000 compiler warnings such as "`This name has not been given an explicit type`"
- Warning messages of the type "`Global name too long`"
- Warning messages of the type "`Source line truncated`"
- Warning messages of the type "`A dummy argument with an explicit INTENT(OUT) declaration is not given an explicit value`"

To avoid termination of the compilation the warnings were disabled with the compiler option choice "`-warn all,nodeclarations,nounused`" while others required source code and script modifications. For example, all references to `parallel` were changed to `parallel_mpi`.

7. LESSONS LEARNED

7.1 Benefits of the FSparse method

Comparing runtime performance for CMAQ 5.1 in the new OpenMP parallel version with the U.S. EPA release showed benefits such as:

- A speedup ~1.2 to 1.75 depending on the solver algorithm and thread count.
- A single source code version of the CTM suitable for either host CPU or offload to the Phi co-processor.
- Hybrid MPI+OpenMP algorithms that offer more on-node compute intensity as the number of available threads rises to 100's and beyond.
- 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:

- Examination of recent progress in sparse matrix techniques and research [5].
- Implementing further compiler and platform tuning opportunities.
- A port to the 2nd generation Intel Phi processor

8. CONCLUSIONS

This report has described an analysis of CMAQ 5.1 behavior in the standard U.S. EPA release and a new thread parallel version of CMAQ for the Rosenbrock and Gear solvers. In this version (v6.2) subroutines common to both algorithms have been successfully developed for applications on host CPUs or Intel Phi processors.

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] Delic, G., 2014: see presentation at the Annual CMAS meeting (http://www.cmasecenter.org).

[5] Duff, I.S., Erisman, A.M., and Reid, J.K., *Direct Methods for Sparse Matrices*, Oxford University Press, 2nd Ed., 2017.