Presented at the 11th Annual CMAS Conference, Chapel Hill, NC, October 15-17 2012

A NEW PARALLEL SPARSE CHEMISTRY SOLVER FOR CMAQ

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

1. INTRODUCTION

This presentation reports on two major performance enhancements for CMAQ. The first change replaces the sparse matrix solver used for chemical species concentrations. The second modification integrates the new solver into the transit over grid cells so that separate blocks of cells are distributed to different threads. Applying both modifications together improves CMAQ efficiency. This modified version of CMAQ is tested with compilers from the Portland Group® [PGI], the Intel Corporation® [INTEL], and the Absoft Corporation® [ABSOFT]. Results are reported for multicore platforms from the Intel Corporation (Intel) and Advanced Micro Devices (AMD).

This report examines parallelism in CMAQ at the thread level in the Rosenbrock (ROS3) chemistry solver version of CMAQ 4.7.1 (hereafter ROS3-HC). This second version offers the best potential for parallel performance improvement. An earlier hybrid parallel model (ROS3-HC) with three levels of parallelism has been described in previous reports at this meeting [Delic,2003-2010]. The (outer) Message Passing Interface (MPI) level is the one previously delivered in the standard U.S. EPA distribution. The (inner) parallel layers developed at HiPERiSM have added both thread-level parallelism and instruction-level parallelism (at the vector loop level) and are suitable for either commodity processors or GPGPU targets. The next section details the HiPERiSM test bed before some description of the latest modifications is presented.

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 platforms, Intel and AMD, have a total of 8 and 48 cores, respectively. This cluster is used for either MPI only, or hybrid thread-parallel OpenMP plus MPI execution. However, to focus analysis on the new modifications only results for a single MPI processes are discussed here since they apply to each MPI process.

Table 2.1. Platforms at HiPERiSM Consulting, LLC

Platform	AMD	Intel
Processor	AMD™ Opteron 6176SE	Intel™ IA32 W5590
Peak Gflops (SP/DP)	110.4 / 55.2	106.6 / 53.3
Power consumption	105 Watts	130 Watts
Cores per processor	12	4
Power consumption per core	8.75 Watts	32.5 Watts
Processor count	4	2
Total core count	48	8
Clock	2.3GHz	3.33GHz
Band-width	42.7 GB/sec	64.0 GB/sec
Bus speed	1333 MHz	1333 MHz
L1 cache	64KB	64KB
L2 cache	512 KB(1)	256MB
L3 cache(2)	12MB	8MB
(1) Per core, (2) Per socket		

2.2 Compilers

This report concurrently used compiler versions from Intel (12.1), Portland (11.5) and Absoft (11.5) for CMAQ 4.7.1 on 64-bit Linux operating systems and hardware. The HiPERiSM Consulting, LLC, version of CMAQ (ROS3-HC) with multi-threaded parallelism was compiled and executed with all three compilers on both platforms shown in Table 2.1.

For each compiler several groups of optimization switches were tested. For each compiler group this analysis included new builds of CMAQ support libraries such as NetCDF, IOAPI, MPICH, STENEX and PARIO. In each case compiler options have been closely examined for effects on numerical precision. This study found some anomalous behavior when the highest optimization levels are implemented with some compilers and these will be described below.

3. EPISODE STUDIED

For all CMAQ 4.7.1 results reported here the model episode selected was for August 09, 2006, using data provided by the U.S. EPA. This episode has the CB05 mechanism with Chlorine extensions and the Aero 4 version for PM modeling. The episode was run for a full 24 hour scenario on a 279 X 240 Eastern US domain at 12

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

Km grid spacing and 34 vertical layers for a total of 2.3 million grid cells. The first pass over all blocks of cells in the grid domain was examined in detail.

4. SPARSE MATRIX ANALYSIS

4.1 BACKGROUND

The Rosenbrock and Gear chemistry solvers in CMAQ apply Gaussian elimination of a sparse matrix system Ax=b many millions of times per simulation. The dimension of matrix A is determined by the number, N, of reacting chemical species (N=72 in this study). While the rank of the species matrix is N^2 = 5148, the number of non-zeros, NZ, is typically an order of magnitude less (of the order of 760). An example of the matrix portrait is shown in Fig. 4.1 (after reordering of rows and columns).

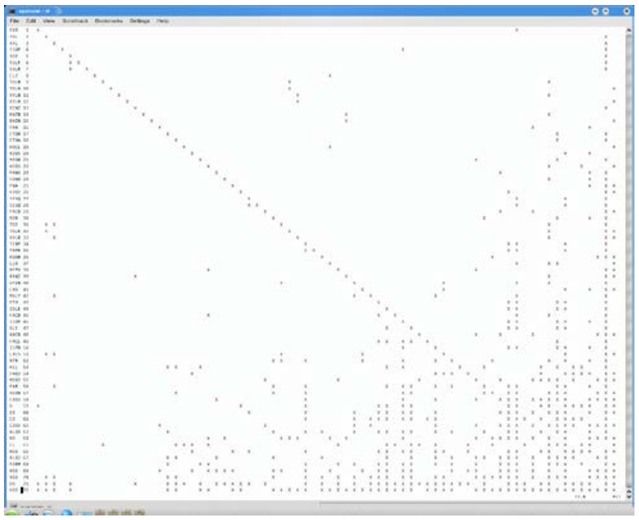
Fig 4.1: Sparse matrix portrait printed from rbsparse in CMAQ4.7.1 in the ROS3 solver version. Each row/column corresponds to the reacting species.

The matrix solution is effected in three stages:

(i) decomposition A=LU,

(ii) forward solve for Lz=b,

(iii) backward solve for Ux=z,

where L and U, are lower and upper triangular matrices such that A=LU. For CMAQ matrix A has large condition numbers and is diagonally dominant. Therefore scaling is applied for A to permit exception handling at runtime. This allows underflows and avoids the execution halting as a result of overflows when no scaling is used. The above solution is applied to each block of grid cells passed to the chemistry solver.

4.2 JSPARSE

Historically the U.S. EPA CMAQ Gear and Rosenbrock chemistry solvers have relied on the JSparse [Jacobson and Turco] procedure to perform the Gaussian Elimination outlined above. This takes advantage of the known sparsity structure before solution begins. JSparse exploits this by using a symbolic decomposition and solution steps that filter out (or minimize) computation of zero entries in sparse Gaussian elimination. This requires the use of several layers of indirect addressing of array subscript and this choice inhibits parallelization of loop nests in the Gaussian elimination algorithm used in JSparse.

Thus JSparse suffers from the deficiency that parallelism is allowed only at the instruction level on inner vector loops over cells for each block of the grid domain passed to the solver. Such indirect addressing subscripts cannot be overcome by compiler options alone.

4.3 FSPARSE

This section summarizes the algorithmic choices that transform JSparse into FSparse for the Rosenbrock (ROS3) chemistry solver.

First of all a few words about sparse matrix storage schemes is in order. All sparse matrix algorithms reference only non-zero elements and store the value in a real array, but differ in the storage method for the location in the full matrix. Each scheme requires indirect subscript references at some level, but the implementation has consequences for parallel algorithm opportunities The Triplet storage scheme (used in JSparse) scans row and columns of the matrix and stores column and row index values in two integer arrays. The Compressed Column (CC) scheme scans down successive columns and uses one integer array i of length NZ, and another pointer array p of length N+1 so that row indices of entries in column j are stored in i(p(j) through i(p(j+1) -1). The scheme is described in chapter 2 of Davis [Davis, 2006] for the C language case. The Compressed Row (CR) scheme scans across successive rows and uses a corresponding storage scheme.

The starting point in FSparse is the CSparse C language library developed by Davis [Davis, 2006] which uses the CC storage form and has been implemented with substantial modification at HiPERiSM. The CSparse library is quite general and extensive, but only the sparse Gaussian procedures have been adopted. CSparse allows a generalized factorization of the type PAQ=LU, where P is obtained from partial pivoting and Q is chosen to reduce fill-in in LU. In CMAQ the permutation matrix Q is in effect the result of the re-ordering step taken over from the JSparse procedure [Jacobson and Turco]. However, P=I (the identity matrix) is the choice in the CMAQ model because the matrix A is diagonally dominant.

Table 4.1. C procedures from CSparse translated to Fortran in the FSparse versions of the ROS3 solver.

CSparse procedure	Description
cs_compress	Map Triplet to CC storage form
cs_lu	Driver for LU decomposition
cs_spsolve	Sparse solve for L, and U
cs_reach	Reach function
cs_dfs	Depth first search
cs_lsolve[1]	Solve Lz=b
cs_usolve[1]	Solve Ux=z
cs_norm	Compute 1-norm of A
1) Converted to parallel and vector form using Compressed Row (CR) format for L and U	

The CSparse procedures listed in Table 4.1 have been extracted and translated into Fortran for integration into ROS3-HC. However, local modifications have been made. For example, cs_lsolve and cs_usolve, will not allow parallel vector instructions on inner loops because the CC form uses indirect addressing of array indexes on the left hand side of the assignment ("="). For this reason ROS3-HC converts L and U to Compressed Row (CR) format after the sparse CC decomposition step for A=LU. The suggestion for the CR form enabling a parallel algorithm is from Björck [Björck, 1996]. This enables vector SSE instructions to schedule the inner loops of forward and backward solve steps while also allowing parallel potential in the outer loop. Such parallel loop nests may easily be parallelized in a GPGPU version, or whenever nested parallel threads are enabled in any future OpenMP standard.

An example of code for the solver part of ROS3-HC is shown in Fig. 4.2.

```
row_fr1: do s_i = 1, NS - 1                            ! row
   DO NCELL = 1, NUMCELLS                              ! vector loop # 31
     rivot(NCELL) = K1( NCELL ,s_i)
   ENDDO

   col_fr1: do s_j = L_w(s_i,sn) , L_w(s_i+1,sn)-2        ! col
        DO NCELL = 1, NUMCELLS                         ! vector loop # 32
           rivot(NCELL) = rivot(NCELL) - Lr_Cx( NCELL,s_j ) *
&                                  K1( NCELL, L_Cj(s_j,sn) )
        ENDDO
   end do col_fr1

   DO NCELL = 1, NUMCELLS                              ! vector loop # 33
      K1( NCELL,s_i) = rivot(NCELL)
   ENDDO
 end do row_fr1
```

Fig 4.2: Example of the forward solve Lz=b of the ROS3 solver (where b=K1). A similar principle is used in the backward solve Ux=z. This pair of solve steps is repeated three times for each time step in the CMAQ Rosenbrock solver procedure CHEM.

The outer row loop is not parallelizable (because of the recurrence on array K1). The column loop is parallelizable because the CR format places the indirect reference on the second index of the K1 array. All loops contain a vector loop on the cell index NCELL for the current block and NUMCELLS is the blocksize. A temporary array (rivot) is introduced so that a vector-inhibiting recurrence is avoided on the innermost loop (#32).

4.4 RBDRIVER (aka CHEM)

The procedure CHEM in CMAQ has major loops over the blocks of cells that the entire grid domain has been partitioned into. Each block is then processed in the solve steps described in Section 4.1. The number of blocks is calculated from the BLKSIZE parameter choice in GRID_CONF. Since the chemistry solver time step for each block is independent of all others, it is logical to distribute different blocks amongst available threads in a thread parallel team using an appropriate scheduling algorithm. This strategy is attractive because it creates coarse parallel granularity for thread teams as a result of the substantial scope of the contained arithmetic operations.

Table 4.2. Subroutine in the U.S. EPA version of the ROS3 solver modified in the new ROS3-HC algorithm

CMAQ procedure	Description
GRID_CONF	Define grid and set BLKSIZE
rbdata_mod	Declare allocatable arrays
rbinit	Initialize and allocate arrays
rbsparse	Set up chemistry structure and symbolic Gaussian elimination
rbdriver	Loop over grid blocks and solve at each chemistry time step
rbcalcks[1]	Prepare photolytic rate coefficients
rbsolver[1]	Driver for 3-stage Rosenbrock chemistry algortihm
rbfeval[1]	Compute rate of change of species concentrations
rbjacob[1]	Compute Jacobian matrix
rbdecomp[1]	Perform LU decomposition
rbsolve[1]	Perform forward/backward solve
1) Inlined into rbdriver	

Table 4.2 shows the subroutines modified in the HiPERiSM version of ROS3. This indicates those subroutines inlined into rbdriver that has two

large parallel regions: one for reordering (as in the original JSparse version), and a second for the chemistry solution with time step integration. Both parallel regions contain loops over the total number of grid blocks, but the first takes a small fraction of the time spent in rbdriver.

The new version of rbdriver was created by successive code structure modifications of the standard ROS3 solver without changing the science of the model in any way. The modified ROS3-HC applies a thread parallel strategy that has three prongs:

1. Partitioning storage into global shared variables and those private to threads.
2. Distribution of BLKSIZE chunks of the grid domain to separate threads in a parallel thread team.
3. Ensuring each thread has inner loops that vectorize whereever possible.

Specific restructuring steps applied to the standard CMAQ gas chemistry solver included:

- Manual inline of procedure calls
- Arrangement of inner loops so that they target SSE vector instructions.
- Declaration of thread parallel regions by insertion of OpenMP directives and classification of local (thread private) and global (shared) variables.
- Simplification/streamlining of redundant code.

4.5 Status of code

This thread-vector parallel strategy can only succeed if there is sufficient coarse grain parallel work for each thread. This is achieved with the modifications described above. However, this creates a large parallel region for the block loop in the solver thread-parallel region. As a consequence, debugging parallel code can be a challenge with opportunities for memory corruption and race conditions. Significant progress has been made but some issues remain to be cleared up. Along the way some important differences between results of different compilers have been observed. Such differences appear to originate in compiler code transformations with higher optimization level choices. Others originate in the use of mixed-mode arithmetic in the standard release of the U.S. EPA CMAQ and this has consequences for numerical precision. Such issues continue to be investigated and results presented here are preliminary, but sufficient to demonstrate proof-of concept.

5. RESULTS

5.1 Compiler issues with CMAQ

CMAQ in the U.S. EPA and HiPERISM versions was compiled and executed with all three compilers listed in the introduction. However, several compiler problems were encountered. The Intel compiler fails when inter procedural analysis is enable with an internal compiler error and therefore this option was disabled. The Absoft compiler fails with an internal error if the optimization level is –O3, so this was lowered to –O2 for the whole of CMAQ, but –O3 was restored for a re-compile of rbdriver. All three compilers vectorize many of the 100 or so candidate loops in the ROS3-HC version of rbdriver, but the Intel compiler vectorizes fewer than the other two compilers (presumably because of scalar instruction scheduling of CPU resources). Also important was the differing numerical results described below.

5.2 Test case of first block

In this section, and the next, two performance metrics are introduced to assess thread parallel performance in the ROS3-HC modified code:

(a) *Speedup* is the gain in runtime over the standard U.S. EPA runtime,
(b) *Scaling* is the gain in runtime for thread counts larger than 1, relative to the result for a single thread.

A test case of the first pass over the grid was investigated for a BLKSIZE parameter of 640 (vector loop length) and 3558 blocks in the major loop of CHEM. Table 5.1 shows timings for completion of this major loop for the U.S. EPA version of CMAQ and the ROS3-HC version of the Rosenbrock solver. The time units are 10^6 microseconds, obtained from the Fortran procedure system_clock (the results for the Intel compiler appear to be anomalous).

With two threads the 3558 solve blocks were alternately distributed using a dynamic scheduling algorithm in OpenMP. Not all cases are completed at this time, but the thread scaling results are very encouraging as shown in the thread scaling column in Table 5.1 with a range of 1.26 to 1.67. This suggests good opportunities at higher thread counts.

Table 5.1. Times (in 10^6 microseconds) for the U.S. EPA (ROS3-EPA) and ROS3-HC versions of CMAQ 4.7.1. The platforms are Intel and AMD for the Absoft, Intel and Portland compilers.

Compiler	Platform	ROS3-EPA time	ROS3-HC		
			Time and thread scaling		
			1 thread	2 threads	Thread scaling
Absoft	Intel		363		
Intel		1.56	3.22	2.38	1.35
Portland		161	527	417	1.26
Absoft	AMD	507	787		
Intel		3.9	6.7	4.6	1.47
Portland		390	643	386	1.67

Table 5.2 shows speedup of ROS3-HC versus the U.S. EPA versions of CMAQ. The EPA version used the conventional JSparse procedure and ROS3-HC used FSparse. The JSparse version appears to benefit from the enhanced scalar (and cache) performance of the Intel platform and the speedup results are lower there than on the AMD platform. For the AMD platform it is clear that at higher thread counts the ROS3-HC version of CMAQ will out-perform the standard U.S. EPA distribution.

Table 5.2. OpenMP speedup for the U.S. EPA (ROS3-EPA) and ROS3-HC versions of CMAQ 4.7.1. The platforms are Intel and AMD for the Absoft, Intel and Portland compilers.

Compiler	Platform	ROS3-EPA	ROS3-HC	
			Speed up by thread count	
			1	2
Absoft	Intel			
Intel		1.0	0.49	0.66
Portland		1.0	0.30	0.39
Absoft	AMD	1.0	0.64	
Intel		1.0	0.59	0.86
Portland		1.0	0.61	1.01

5.3 Results for the 24 hour episode

Table 5.3 shows the available timing results for the full 24 hour scenario with a BLKSIZE parameter of 640. This table will be expanded as more results are completed.

5.4 Numerical precision issues

The results of the previous discussion compared three different compilers. For each compiler a careful choice was made of compiler switches that control how numerical arithmetic operations are performed. Nevertheless, in the course of the detailed investigation of the first pass over all 3558 blocks, numerical differences between compilers were observed. Specifically the number of time steps each compiler used was different. This difference is due to the calculation of the time step increment in the solver step.

Table 5.3. Wall clock times (in hours) for the U.S. EPA (ROS3-EPA) and ROS3-HC versions of CMAQ 4.7.1. The platforms are Intel and AMD for the Absoft, Intel and Portland compilers.

Compiler	Platform	ROS3-EPA	ROS3-HC			
			Time in hours by thread count			
			1	2	4	8
Absoft	Intel					
Intel						
Portland			57.0	48.2		
Absoft	AMD		156.7			
Intel			112.6			
Portland			83.9	66.4		

Table 5.4. Execution time and total number of chemistry time steps in ROS3-HC versions of CMAQ 4.7.1 for the first pass over the 3558 blocks of the entire domain. The platforms are Intel and AMD for the Absoft, Intel and Portland compilers.

Compiler	ROS3-HC (1 thread)	
	Wall clock time (sec)	total time step count
Absoft		
Intel	281	25782
Portland	301	25753
Absoft	936	25770
Intel	600	25782
Portland	613	25753

The results of Table 5.4 are for parallel loop execution where there is a halt after completion of the first pass over 3558 blocks on the entire grid domain. The wall clock time for the parallel loop is shown as is the total count of all chemistry time steps. This time step count does not depend on the hardware platform, but is does depend on the choice of compiler. Possible causes for the differences observed in Table 5.4 are two-fold:

- Choice of compiler switches controlling numerical operations
- Mixed mode arithmetic in CMAQ

The first point will affect precision in the LU decomposition and solve steps. Such changes are easily monitored in ROS3-HC with an option to calculate several types of norms including |A|, |x|, and |Ax-b|. The second point is due to mixing of single and double precision floating point variables in rbdriver and the chemistry solver. While most variables in the Rosenbrock solver are double precision, some are not (e.g. results returned from rbcalcks), and the stringent time-step increment calculation. These issues require more detailed study than space here allows.

6. LESSONS LEARNED

6.1 Compilers for the CMAQ model

Differentiating compilers for CMAQ based on performance alone is now more difficult because of the lower optimization levels that allowed for stable compilation. As complier bugs fixes are completed in new releases this situation will change.

An additional discovery at HiPERiSM Consulting, LLC, has been the consequences compiler optimization and complier choices have for numerical precision. For this reason further study is appropriate and in the interim care needs to be exercised in the use of compilers to avoid erroneous model predictions. As a consequence the most conservative choices for compiler switches that control numerical precision are advisable for all compilers used with CMAQ.

As a consequence of these observations a direct comparison of the timing of the three compilers is not appropriate at this time because of these numerical difference observations coupled with the optimization choices made for stable compilation.

6.2 CMAQ in multi-thread mode

This analysis compared runtime of CMAQ 4.7.1 in the new OpenMP parallel version with the U.S. EPA release. The observations indicated that the multi-threaded speedup:

- Showed a range of 1.26 to 1.67 with 2 parallel threads.
- Depends on the choice of hardware.
- Was dependent on compiler choice based on comparison of compilers from Absoft, Intel and the Portland Group.

6.3 Comparing hardware platforms

There is a large difference in runtime and thread scaling between AMD and Intel platforms for the same model simulation of an individual serial run. However, with more threads in the thread team good scaling is anticipated and work in this direction will continue.

7. CONCLUSIONS

This report has described an analysis of CMAQ 4.7.1 behavior in the standard U.S. EPA release and a new thread parallel version of CMAQ for the Rosenbrock solver. Opportunities exist for speedup with an increased number of parallel threads. This trend was observed with three compilers on two hardware platforms. However, issues were observed for numerical precision due in part to compiler differences and the way precision is treated in CMAQ.

Compilers from Absoft, Intel and the Portland group all offered value for the CMAQ model but some experience difficulties with higher optimization levels.

Further opportunities remain for thread parallelism in other parts of the CMAQ model outside of the solver and work in this direction continues at HiPERiSM Consulting, LLC. The new (second) version of ROS3-HC offers layers of parallelism not available in the standard U.S. EPA release and may be ported to hardware and software that supports nested parallel threads.

REFERENCES

ABSOFT: The Absoft Corporation http://www.absoft.com

Björck, Åke, Numerical Methods for Least Squares Problems, SIAM, Philadelphia, 1996.

Davis, T.A., Direct Methods for Sparse Linear Systems, SIAM, Philadelphia, 2006.

Delic, G., 2003-2010: see presentations at the Annual CMAS meetings (http://www.cmasecenter.org).

INTEL: Intel Corporation, http://www.intel.com

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

PGI: The Portland Group http://www.pgroup.com