NEW STRATEGIES FOR IMPROVING PERFORMANCE AND PRECISION IN CMAQ AND GMI MODELS

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

1. INTRODUCTION

When discussing topics such as energy, climate, and pollutant transport from regional to global scales, it is appropriate to also review some algorithms common to both the Community Multi-scale Air Quality Model (CMAQ) and the Global Modeling Initiative (GMI) chemistry and transport model (CTM), as respectively developed (or supported) by the U.S. EPA and NASA. Some motivations of new strategies for improvement in these models include issues such as increasing model complexity, changes in computer architectures, and developments in compiler technology.

To describe production and loss of chemical species in reaction mechanisms, both CMAQ and GMI allow a choice of various gas-phase solvers that implement different algorithms for integration of a sparse stiff system of ordinary differential equations. One such solver is the Gear algorithm and its use is common to both CMAQ and GMI. The implementation of this solver in both models is based on the work of Jacobson and Turco (1994) for the JSPARSE method.

The HiPERiSM version replaces the legacy sparse matrix methodology of Jacobson and Turco by a more modern one (FSPARSE) described by Delic (2014). This work reports another variant of the FSPARSE method with a memory model better suited to the commodity architectures now in use.

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, has a total of 8 and 48 cores, respectively. This cluster is used for either MPI only, or hybrid thread-parallel OpenMP plus MPI execution.

2.2 Compilers

This report implemented the Portland Group® (PGI) compiler (release 13.4) for CMAQ 4.7.1 on 64-bit Linux operating systems and hardware from the Intel Corporation (INTEL) and Advanced Micro Devices (AMD) shown in Table 2.1.

2.3 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 Km grid spacing and 34 vertical layers for a total of 2.3 million grid cells.

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
Total memory	128GB	96GB
(1) Per core, (2) Per socket		

3. CMAQ AND GMI MEMORY MODEL

3.1 Background

The chemistry-transport model in CMAQ (CCTM) and GMI in CTM allow a choice of various gas-phase solvers that implement different algorithms for integration of a stiff system of ordinary differential equations (ODE), with sparse

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

Jacobians, to describe production and loss of chemical species in reaction mechanisms. When the Gear algorithm is used for the system of ODE's it accounts for over 50% of the wall clock time of a simulation in CMAQ.

The modifications introduced in JSPARSE took advantage of vector processing on the pipelined vector architectures of Cray computers (Cray). However, on Cray computers, the cost was prohibitive if the Gear method is applied to each cell of a multidimensional grid. Therefore one modification introduced in JSPARSE was to apply the Gear algorithm to a block of grid cells simultaneously. This modification allowed vector instructions to be applied to innermost loops over the block dimension length, NUMCELLS, equal to the size of the block (BLKSIZE). This method has the disadvantage that it requires a memory copy of concentrations from an array dimensioned by column, row, and level, into a one-dimensional array indexed by cell count using indirect addressing. Nevertheless, this blocking method worked well on Cray architectures (Cray) with 128 word vector registers and hardware gather-scatter operations using a BLKSIZE ~500 grid cells. But this legacy method relies on hardware features of a 20 year old architecture that are no longer available in current commodity CPUs. Thus this choice has a memory copy penalty on commodity CPUs where a BLKSIZE larger than approximately 64 leads to increased computational time (Delic, 2013). Another disadvantage of choosing larger values of BLKSIZE is that the Gear solver time step size is the same for all cells in a block, and cells with faster rates of species concentration change may not converge as well as those cells with slower concentration rate changes (i.e. cells differ in "stiffness"). To ameliorate the negative consequences of disparate cell stiffness, the algorithm in JSPARSE offers an option to order all cells in the grid into blocks of cells having similar stiffness, with each block having a length of NUMCELLS. However, on commodity CPUs, the blocking technique used in JSPARSE leads to excessive data translation look-aside buffer (TLB) cache misses that inevitably stalls the CPU (Young and Delic, 2008). This indicator suggests that memory latency is expanded as the CPU waits for data address translation and fetch from higher up the memory hierarchy. The cause is a memory model that persistently maps from a regular grid of cells populating a global array dimensioned by column, row, and level (or layer), to a single linear array on each call to the Gear solver. Each block of cells in the linear array is then passed to the CCTM for the chemistry time step integration.

3.2 Performance profile of CMAQ

Table 6 lists time expended by each the physical processes (in SCIPROC) in the standard release of CMAQ for the 24 hour episode described in Section 2.3, with serial execution for one MPI process (NP=1), using the Portland compiler on the Intel node. The last column gives the fraction of the total runtime used by the respective processes and it is clear that CCTM dominates. Therefore an improvement in the CHEM subroutine will significantly impact the total wall clock time.

Table 3.1. CMAQ science processes and wall clock times for a one-day simulation.

Process and function	Subroutine name	Wall clock time (hours)	Fraction of total (percent)
Asymmetric convective model (ACM) for vertical diffusion	VDIFF	5.30	13.7
Couple concentration values for transport	COUPLE	0.13	0.33
Advection in the horizontal plane	HADV	2.00	5.15
Advection in the vertical (Z) direction	ZADV	0.81	2.09
Horizontal diffusion	HDIFF	0.17	0.45
Decouple concentration values for transport	DECOUPLE	0.19	0.49
ACM and resolved cloud processes	CLDPROC	0.74	1.92
CCTM	CHEM	23.34	60.1
Aerosol species processing	AERO	6.13	15.8

For the same 24 hour episode Fig. 3.1 shows a runtime profile of the fraction of wall clock time for three of the most time consuming physical processes in CMAQ: CHEM, AERO, and VDIFF. The balance between these three shifts depending on the simulation hour, but CHEM always dominates

3.3 Improvements in the memory model

HiPERiSM's improvements for the Rosenbrock solver (Delic, 2013) have been applied to the Gear ODE algorithm. A previous report by Delic (2014) describes (for the Gear ODE algorithm) how the sparse solver (FSPARSE) replaces the legacy JSPARSE sparse

solver method, but still retains the blocking method and stiffness ordering described in Section 3.1.

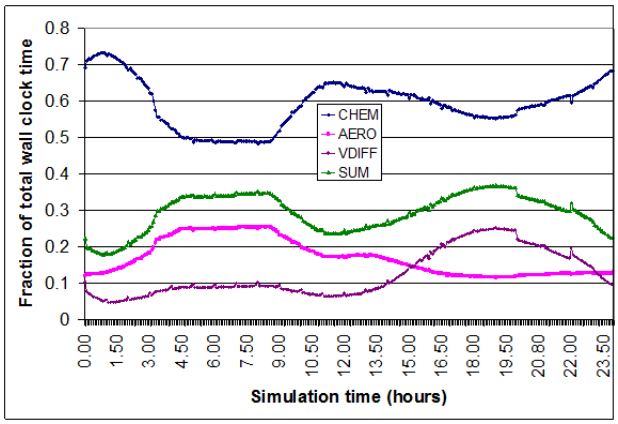

Fig 3.1. This shows the fraction of total wall clock time for selected processes in SCIPROC for all 12 x 24 calls in a 24-hour simulation of the U.S. EPA standard code with 1 MPI process using the Portland compiler on the Intel node. Only CHEM, AERO, and VDIFF are shown here, together with the sum (SUM) of the last two.

This report describes an alternative approach that improves the memory model in GEAR-HC. This alternative dispenses with copying blocks of cells into a linear array and with the stiffness ordering of blocks of cells. The FSPARSE method integrates the new solver into the transit over the regular grid domain ordered by column, row, and level. Therefore in this version of CMAQ the CCTM is called for each individual cell in the grid. This is achieved in a thread parallel region over a grid column index:

```
C$OMP DO SCHEDULE (DYNAMIC,MY_CHUNK)
      blk_col: DO COL = 1, MY_NCOLS
        blk_row: DO ROW = 1, MY_NROWS
          blk_lev: DO LEV = 1, NLAYS
             ……
          END DO blk_lev
        END DO blk_row
      END DO blk_col
```

In this cell-centric version, each thread ranges over all rows and levels for a selected column. This has two benefits (a) no stiffness ordering of cells is required as the gear solver is applied to each cell individually, and (b) the memory model is improved because a redundant memory copy is removed. To distinguish results of the two versions of GEAR-HC the following discussion will designate the block cell version as BLOCK, and the individual grid cell version as CELL.

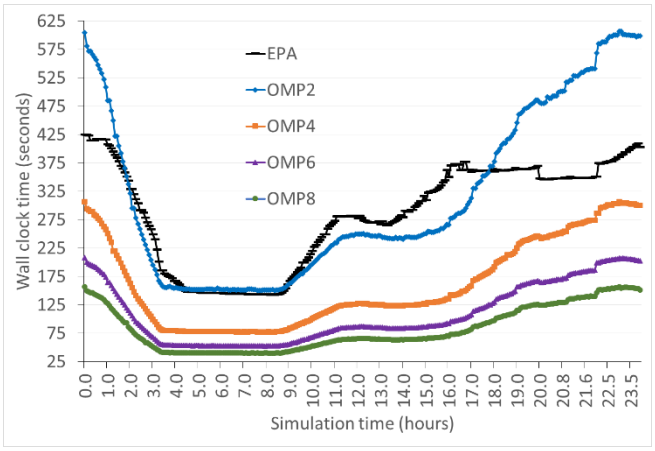

Fig 3.2. This shows the CHEM wall clock time for GEAR-EPA and GEAR-HC (in the CELL version) with 2 to 8 threads.

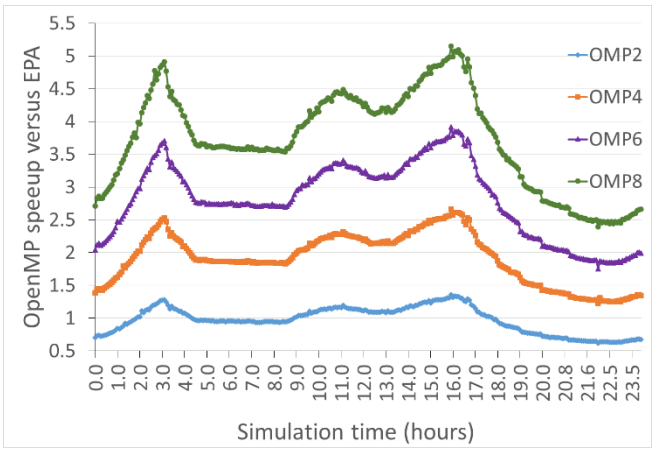

Fig 3.3. This shows the speed up in CHEM of GEAR-HC (in the CELL version) versus the GEAR-EPA, with 2 to 8 threads.

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

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

(b) *Scaling* is the gain in runtime with thread counts larger than 1, relative to the result for a single thread.

For the performance of CHEM in the 12 x 24 calls of the 24 hour simulation two more results are presented for the case of 1 MPI process using the Portland compiler on the Intel node. Fig. 3.2 shows the wall clock time for calls to CHEM in the standard EPA version (GEAR-EPA) and the CELL version of GEAR-HC with increments of OpenMP threads from 2 to 8. Except for the 2 thread case, the wall clock time is greatly reduced in all 12 x 24 calls.

To show the parallel performance more clearly, Fig. 3.3 shows the speed up for calls to CHEM by GEAR-HC (in the CELL version) versus the GEAR-EPA, with 2 to 8 threads. This speed up varies from 2.5 to 5 with 8 threads, and increments substantially from 6 to 8 threads.

4. PERFORMANCE OF GEAR-HC

4.1 Speedup and scaling for 1 MPI process

For the CMAQ chemistry solver, in the BLOCK version, the grid of 2.3 million grid cells is partitioned into blocks of size BLKSIZE and these blocks are distributed to threads in a thread team in GEAR-HC. In the standard U.S. EPA distribution BLKSIZE=50 is fixed, but in the BLOCK version of GEAR-HC BLKSIZE= 48. As described in Section 3.3, in the CELL version of GEAR-HC, no blocking of cells is used. Instead the Gear solver is applied to each cell in the grid domain. Table 4.1 shows the wall clock time (in hours) for the BLOCK and CELL versions of GEAR-HC. Also shown are the times for the standard U.S. EPA version. Values are shown for selected cases of the MPI process count (NP). Speedup and scaling metrics are calculated from the values of Table 4.1.

Table 4.1. Wall clock times (in hours) for the U.S. EPA (GEAR-EPA) and GEAR-HC in BLOCK and CELL versions of CMAQ on the AMD platform for the Portland compiler.

MPI processes (NP)	GEAR-HC	GEAR-EPA	GEAR-HC Time in hours by thread count				
			1	2	4	6	8
1	BLOCK	83.3	39.7	28.4	22.9	49.1	46.8
2		42.3				25.1	23.1
4		22.5				13.6	12.9
8		11.6				7.8	
1	CELL	83.3	136.9	84.5	59.3	51.3	46.3
2		42.3				26.4	23.7
4		22.5				14.4	13.3
8		11.6				7.9	

For the NP=1 case Fig. 4.1 shows the wall clock time for EPA and the two GEAR-HC versions of CMAQ for the 24 hour simulation. With a single OpenMP thread the CELL version uses more time because there are no vector loops over the number of cells in a block of cells. However, as is evident, the gain in performance with increasing thread count eliminates the discrepancy between BLOCK and CELL version with 6 or more threads.

Fig. 4.2 shows the speedup of GEAR-HC over GEAR-EPA corresponding to the values of Fig. 4.1. With 6 or 8 threads speedup is similar for either BLOCK or CELL versions. Fig. 4.3 shows the results for thread scaling on a very refined vertical scale. The apparent greater scaling for the CELL version of GEAR-HC is the consequence of the longer runtime with a single thread.

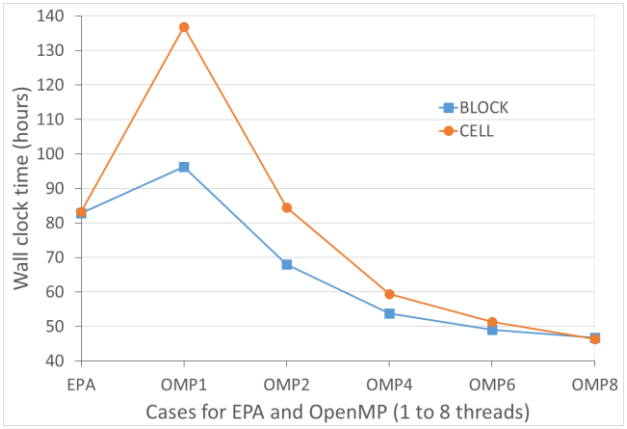

Fig 4.1. Wall clock time (hours) of GEAR-EPA (EPA) and GEAR-HC for CMAQ 4.7.1 with a single MPI process (NP=1). For OpenMP thread counts from 1 to 8 the block (BLOCK) and cell (CELL) versions of GEAR-HC are compared.

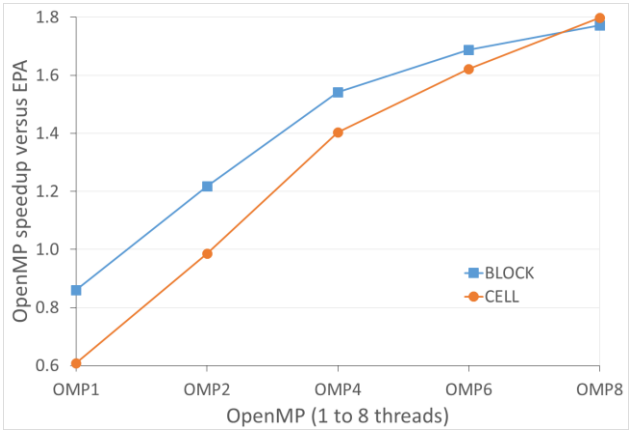

Fig 4.2. Speed up of GEAR-HC for CMAQ 4.7.1 over the serial GEAR-EPA version with a single MPI process (NP=1). For OpenMP thread counts from 1 to 8 the block (BLOCK) and cell (CELL) versions of GEAR-HC are compared.

4.2 Speedup and scaling for MPI process counts more than 1

For the NP=1 to 8 cases of Table 4.1, Fig. 4.4 shows the wall clock time for EPA and the two GEAR-HC versions of CMAQ, for the 24 hour

simulation. With 6 or 8 OpenMP threads the BLOCK and CELL versions have similar results. However the gain in performance with increasing thread count diminishes because the grid domain size for each MPI process shrinks as NP increases. This reduces the number of cells available to the thread team. This is confirmed in the speedup results shown in Fig. 4.5, where speedup results decline as the NP count increases. Nevertheless, speedup is in the range 1.5 to 1.7 when NP is greater than 2.

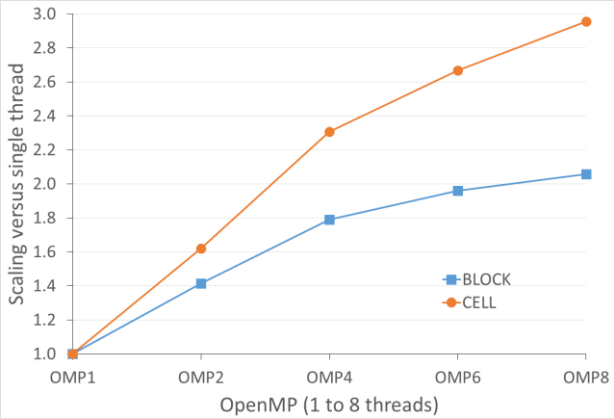

Fig 4.3. Scaling of GEAR-HC with 2 to 8 threads for CMAQ 4.7.1 over the single thread result for one MPI process (NP=1). At each thread count the block (BLOCK) and cell (CELL) versions of GEAR-HC are compared.

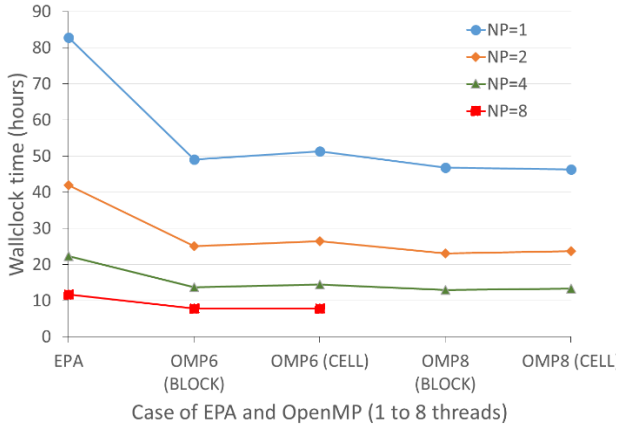

Fig 4.4. Wall clock time (hours) of GEAR-EPA (EPA) and GEAR-HC for CMAQ 4.7.1 with four different choices of the number of MPI processes (NP). This compares the block (BLOCK) and cell (CELL) version of GEAR-HC with the same OpenMP thread count (6 or 8).

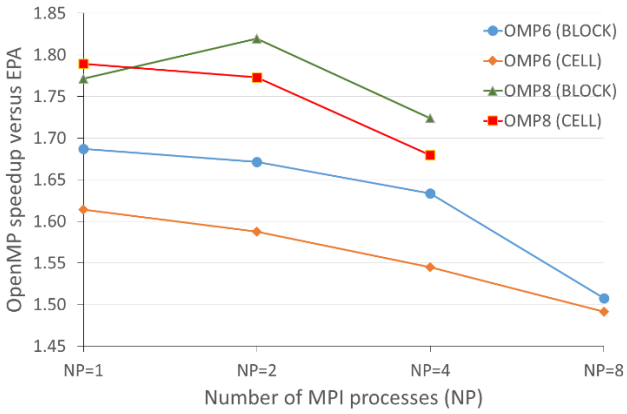

Fig 4.5 Speed up of GEAR-HC for CMAQ4.7.1 over the serial GEAR-EPA version with four different choices of the number of MPI processes (NP). For the same OpenMP thread count (6 or 8) the block (BLOCK) and cell (CELL) version of GEAR-HC are compared.

5. NUMERICAL ANALYSIS

5.1 Precision control in CCTM

The two GEAR-HC sparse methods, BLOCK and CELL, solve the same sparse system, but differ in the number of chemistry time steps required to solution. Convergence is controlled in both methods by accuracy parameters ATOL and RTOL. Inspection of the convergence pattern shows that GEAR-EPA repeatedly exceeds the upper time step bound and terminates earlier. This is the result of less accuracy in GEAR-EPA because of numerical precision issues. The consequences for precision in species concentrations have been explored in detail elsewhere (Delic, 2013, 2014).

The CCTM solver uses double precision arithmetic but accepts some input data from single precision variables (temperature, pressure, photolysis rates, reaction rates, etc.). The U.S. EPA code implements mixed mode arithmetic and is not consistent in promoting constants or variables from single to double precision arithmetic. This is particularly egregious in the CALCKS procedure where thermal and photolytic reaction rates are computed using single precision arithmetic. Therefore using an ATOL=1.E-09 is moot for the JSPARSE method in the GEAR-EPA version.

In the GEAR-HC FSPARSE algorithm arithmetic consistency has been implemented throughout the chemistry solver and because of the higher accuracy, a reduced ATOL value is tolerated. This reduces the number of chemistry

time step iterations in GEAR-HC and therefore ATOL=1.E-07 is the default used in GEAR-HC. Due to the higher precision in the convergence criterion when it is applied in the Gear ODE solver in the CELL version of GEAR-HC, no sacrifice in accuracy of chemical species concentration is expected.

6. LESSONS LEARNED

6.1 Benefits of the FSPARSE method

Comparing runtime performance for CMAQ 4.7.1 in the new OpenMP parallel version with the U.S. EPA release showed benefits such as:

- A speedup ~1.5 with 8 parallel threads and NP=8.
- A speedup ~1.7 with 6 parallel threads and NP=4.

6.2 Numerical precision issues

A comparison of numerical precision for CMAQ 4.7.1 in the new OpenMP parallel version with the U.S. EPA release suggest:

- Limitations due to the EPA method's inconsistent use of mixed mode arithmetic.
- The FSPARSE method was more precise by many orders of magnitude.
- The blocking of cells into groups required in the BLOCK method, before invoking the CCTM, means that the convergence criterion uses an average error over all cells in a block.
- The CELL method applies the convergence criterion to each individual cell and thereby offers more precision in the final solution.

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 GEAR ODE solver.

Speed up with an increasing number of parallel threads reaches the range 1.5-1.7 over the standard CMAQ release. However, numerical precision issues were observed and are due in part to the way arithmetic precision is treated in the U.S. EPA version. The precision of the GEAR-HC version in the transit over individual cells of the grid offers superior precision compared to the blocking method previously used.

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 GEAR-HC offers layers of parallelism not available in the standard U.S. EPA release and is portable across hardware and compilers that support thread parallelism.

REFERENCES

CMAQ, U.S. EPA, Office or Research and Development, National Exposure Research Laboratory, Atmospheric Modeling Division, http:// http://www.epa.gov/amad/, and Community Modeling and Analysis System, http://www.cmascenter.org/cmaq/

Cray, http://www.cray.com/About/History.aspx, http://en.wikipedia.org/wiki/Vector_processor.

Delic, G., 2013: contribution to 12th Annual CMAS Conference, Chapel Hill, NC, October 28-30, 2013, https://www.cmascenter.org/conference/2013/agenda.cfm)

Delic, G., 2014: presented at the 8th International Workshop on Parallel Matrix Algorithms and Applications (PMAA14), July 2-4, Università della Svizzera italiana // Lugano, Switzerland , submitted to Parallel Computing. (http://pmaa14.ics.usi.ch/)

GMI, http://gmi.gsfc.nasa.gov/

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

Young, J. O. and Delic, G., 2008: contribution to 7th Annual CMAS Conference, Chapel Hill, NC, October 6-8, 2008, https://www.cmascenter.org/conference/2008/agenda.cfm.