Modern Environmental Science and Engineering (ISSN 2333-2581)
September 2019, Volume 5, No. 9, pp. 775-791
Doi: 10.15341/mese(2333-2581)/09.05.2019/001
Academic Star Publishing Company, 2019
www.academicstar.us

A Thread Parallel Sparse Matrix Chemistry Algorithm for the Community Multiscale Air Quality Model*

George Delic
HiPERiSM Consulting, LLC, USA

Abstract: This reports on integration of a new Chemistry Transport Model (CTM) sparse matrix algorithm (FSparse) as a replacement of the legacy JSparse algorithm in the U.S. EPA Community Multiscale Air Quality (CMAQ) model. This has been implemented in both Rosenbrock and Gear methods for aqueous chemistry in a hybrid MPI and OpenMP implementation. Both methods are well suited for an OpenMP thread-parallel version. For a 24 hour scenario, execution performance results for both MPI and OpenMP thread parallel scaling are presented with the CMAQ5.3b release on a heterogeneous cluster of 10 nodes with a total of 128 cores. The FSPARSE version of CMAQ typically provides significant speedup over the standard EPA release with similar precision in predicted species concentration values.

Key words: air quality models, sparse matrix methods, MPI, OpenMP, rosenbrock solver, gear solver

1. Introduction

This study reports on major performance enhancements for the Community Multi-scale Air Quality Model (CMAQ) that add new levels of parallelism and replaces the legacy algorithm for a sparse matrix linear equation solver. CMAQ is a major air quality model (AQM) developed by the U.S. EPA [1] and is supported through the Community Modeling and Analysis System (CMAS) [2], as a part of the University of North Carolina, Institute for the Environment, Chapel Hill, North Carolina, USA [3]. CMAQ has a world-wide user base with over 3000 downloads as reported by CMAS. The release used in this analysis is 5.3b and is available for download at Github [4]. The remainder of this section reviews some preparatory background to set the context of this work.

1.1 Air Quality Modeling: Use and Regulation

The role of air quality modeling is to study and predict the chemical interaction, atmospheric transport, and deposition of Criteria Pollutants (and other species) in out-door air on metropolitan, regional, and continental temporal scales. The AQM initiative in the USA dates from the 1970's and is a response to the Clean Air Act [5]. Title 1 of the Clean Air Act (CAA) directs the U.S. EPA to establish National Ambient Air Quality Standards (NAAQS) for common pollutants posing health risks. Federal regulation requires States to demonstrate attainment of the NAAQS and together with the U.S. EPA, they enact regulations to control industrial and commercial pollutant emission sources, while only the U.S. EPA regulates mobile emissions sources. In 1990 the CAA was amended to add Air Toxics and Acid Rain provisions and so-called non-attainment areas were classified.

The criteria pollutants identified in the CAA [5] include:

- Ground level ozone (O_3 - contributor to smog)
- Particulate Matter (PM) in the 2.5 to 10 micron range ($PM_{2.5}$ and PM_{10}) that poses atmospheric haze and respiratory risk

* In memoriam Jeffy O. Young, National Exposure Research Laboratory Atmospheric Modeling Division, U.S. EPA.

Corresponding author: George Delic, Ph.D.; research areas/interests: wind energy. E-mail: rominatripodi1991@gmail.com.

- Lead (Pb)
- Nitrogen dioxide (NO_2)
- Sulfur dioxide (SO_2)
- Carbon monoxide (CO)

CMAQ is used on continental and regional scales in the USA to predict concentrations and transport of criteria pollutants and also as a real-time forecasting model. At the State level, its use is often driven by a State Implementation Plan (SIP), as developed by a given State. A SIP defines regulations and their scope within that State and also spells out consequences for NAAQS when implemented. A SIP may only be approved by the appropriate U.S. EPA Regional Office for that State and typically uses environmental models to demonstrate NAAQS attainment. The CMAQ model, when used in the AQM, is first validated by application to historical episodes and only then is it applied to future scenarios to demonstrate compliance with the NAAQS. Thus, with the aid of an AQM, an approved SIP must demonstrate either NAAQS attainment by the predetermined date, or NAAQS non-attainment. In the case of NAAQS attainment, a 10 year Maintenance Plan is imposed. Whereas in the case of non-attainment, a Federal Implementation Plan (FIP) is developed at the local U.S. EPA Regional Office and Federal sanctions may be applied on the State in question.

Over the decades the science in the CMAQ model has increased in complexity and continues to do so. As a result, both the magnitude of the wall clock time and volume of output data has escalated over time with each new release. Therefore there is always a latent need to improve CMAQ efficiency in performing simulations on modern computer architectures. Advances in processor and memory architecture, as well as software paradigms, are regularly explored with a view to their utility for improved efficiency and scalable performance.

1.2 Parallelism in the Computing Market Place

The High Performance Computing (HPC) marketplace is now dominated by parallel architectures of several different types and requires a careful analysis of the hardware model before a port of legacy applications is attempted to any new architecture. Table 1 summarizes some traditional HPC hardware models and their acronyms, such as processing element (PE). From the end-user (or application) viewpoint this means choosing a parallel program paradigm amongst the dominant multi-platform ones currently available by categories such as those summarized in Table 2. Typical modern commodity architectures include clusters of computer nodes containing one or more central processing units (CPU), with each CPU populated by multiple cores, or PEs, each of which may operate independently in a SIMD environment (Table 1). In recent years, nodes have acquired add-on co-processor devices that host their own local memory and are populated with many (hundred's of) cores. The former will be referred to as multi-core and the latter as many-core devices. Both types of hardware environments delegate subtasks to thread processes that execute on individual cores.

Table 1 High performance computing (HPC) hardware models.

Mnemonic	Functionality	Features
SIMD	Single Instruction Stream Multiple Data Stream	All PEs execute exactly the same instruction at the same time
	SMP = Shared Memory Parallel	Memory is global to all PEs
MIMD	Multiple Instruction Stream Multiple Data Stream	All PEs execute different instructions at the same time
	DMP = Distributed Memory Parallel	Memory is local to each PE
	SMP = Shared Memory Parallel	Part of memory is global to all PEs and part is local to each PE

Table 2 High performance computing (HPC) parallel program paradigms.

Paradigm	Model	Features
MPI (Message Passing Interface)	Functional parallel (or task parallel)	Distinct tasks perform operations simultaneously (on different data)
OpenMP (compiler directives)	Master-worker	Master task spawns subtasks (workers) to other processes to distribute subtasks
MIC (Many Intergrated Core device)	Massive parallelism	Cohorts of thread teams

1.3 Parallel Programming Styles in CMAQ

The standard distribution of CMAQ [3,4] implements a MIMD/DMP hardware model (Table 1) and uses MPI (Table 2) to implement this by distributing partitions of the grid domain to different MPI processes. It also relies on instruction level parallelism [6] by invoking vector instructions (where they are not otherwise inhibited) for the innermost loops. However, the standard distribution of CMAQ does not implement a SIMD/SMP (Tables 1 and 2) hardware model. Therefore, one major performance enhancement reported here is the inclusion of a thread parallel program paradigm into the standard U.S. EPA release of CMAQ for one of the most time consuming modules of the CMAQ code. To achieve more efficient parallel performance, the legacy sparse matrix algorithm in the standard release of CMAQ is replaced with a modern version.

The new version of CMAQ described here has three levels of parallelism:

1) The outer MPI level is the one previously delivered in the standard U.S. EPA distribution.
2) Each MPI process activates its own team of threads in a thread parallel layer.
3) Instruction-level parallelism at the vector loop level is preserved for each thread.

This new hybrid version of CMAQ is suitable for either multi-core commodity processors or for many-core general purpose add-on accelerators. The suitability of the latter was explored previously [12] for the case of the Intel Many Integrated Core (MIC) architecture but is not discussed here for CMAQ.

1.4 Algorithms in CMAQ

A brief summary of the algorithm design in CMAQ is given here as a background to the descriptions of the new algorithmic changes that follow. A comprehensive survey of the science and algorithms developed and applied in CMAQ is to be found in a detailed report prepared by the U.S. EPA [8].

1.4.1 Science Processes in CMAQ

A one-atmosphere model was developed and this describes the dynamics by a set of governing equations on a regular grid of cells populating a global array dimensioned by column, row, and level (or layer), for the terrestrial atmosphere. This atmospheric model uses a transport mechanism that consists primarily of numerical algorithms for advection, with vertical and horizontal diffusion, using meteorological input data from another model. In this dynamical model, multiple science modules describe various physical processes such as advection, diffusion, photolysis, aqueous chemistry and cloud dynamics, gas-phase chemistry, etc. An operator splitting methodology allows a fractional time step implementation of the science processes that are integrated over time. The numerical integration of the advection time step imposes input synchronization at a time step interval Δt_{sync}. This time stepping method relies on the approximation that the computational grid remains constant for the duration of the interval of the synchronization time step. All chemical species concentrations are stored in a global array (indexed as described above) and this is accessible to all science processes that may affect them. Chemical transformations occur in gas, liquid, or solid phases and each is modeled separately. The gas phase is dominant and these transformations are described in the CMAQ Chemical Transport Model (CCTM). Operator splitting allows the gas-phase chemistry to be decoupled from other physical processes. The CCTM

module computes the gas phase chemistry in a numerical model of reaction kinetics for production and loss of chemical species. This is accomplished through solution of equations that arise from the mathematical representation of the gas phase chemistry where reaction rates determine production (or loss) of chemical species in the gas phase. The number of reactions that transform reactants into products varies from approximately 90 to several hundred, and the number of species may have a similar range. The selection of the chemical species and the group of governing chemical reactions, known as a chemical mechanism, are predetermined and are interchangeable in the CMAQ model as the knowledge base improves.

1.4.2 Gas Phase Chemistry Solver

Operator splitting in the CMAQ dynamical model allows gas-phase chemistry to be de-coupled from physical processes. As a consequence continuity equations for each gas-phase mechanism species are formulated and solved independently in each cell of the regular grid over column, row and level dimensions. The CCTM module computes the gas phase chemistry in a numerical model for reaction kinetics where reaction rates determine production, or loss, of chemical species in the gas phase. A simple first order ordinary differential equation (ODE) relates the rate of change of species concentration to production and loss terms on the right hand side, with one such equation for each species. Concentrations of species at a later time are obtained from an integration scheme for the first order ODE [9]. However, in the case of CMAQ, the ODE forms a coupled system of order N, the number of reacting species, with some set of initial values of each. The system is non-linear because the production and loss terms on the right hand side may include second and third order reactions for some species. Furthermore, because of widely varying time scales of the reactions, the system of ODE's is stiff (see [9] for a definition). The ratio of largest to smallest eigenvalues of the Jacobian matrix is typically of the order of 10^{10} (or larger) in atmospheric chemistry problems and this represents an extreme case. The system of ODE's of rank N needs to be solved many times for each cell in the grid domain of the advection time step scheme for the dynamical processes. Not surprisingly the execution time of the gas chemistry solver is a substantial fraction of the total wall clock time of a simulation and depends on the ODE solution method.

1.4.3 The Gear Algorithm as Applied in CMAQ

While the following description is predominantly focused on the Gear algorithm, results for the case of the Rosenbrock algorithm [10] are also included here. The Rosenbrock algorithm in CMAQ has been previously investigated by Delic [11] and uses the same sparse Gaussian elimination method discussed here for the Gear algorithm. The same FSPARSE algorithm was also applied to the Global Modelling Initiative (GMI) under contract to NASA and is reported in [12]. These are two of the three numerical integration schemes used in the CCTM module of CMAQ. The method proposed by Gear [13], was adapted by Jacobson and Turco [14]. This is explained by Jacobson [15] for AQM, and was later modified by the U.S. EPA for application to CMAQ. Even with the efficiencies developed in [14,15] the execution time is typically 60% of the total wall clock time of a simulation. Since the Gear solver is well documented [16] it will be summarized briefly here only to the extent needed to understand the application in CMAQ.

The Gear method is a numerical ODE integration formula that is a multi-step and multi-order predictor-corrector algorithm, where the corrector part implements a Newton iteration requiring computation of a Jacobian matrix. After convergence is achieved a local truncation error is computed in an L_2 norm over species and this is used to determine both the chemistry time step size and the order of the method.

On entry the dynamical time step, Δt_{sync}, is subdivided into chemistry time steps and, with an initial estimate, the Gear algorithm begins with an order one predictor formula. The predictor-corrector method proceeds until a prescribed error tolerance in

the local error is either achieved or not. If achieved, then the predicted concentration is accepted and the next chemistry time step, and the order of the integration formula, are estimated. Since the Gear method is a multi-order one, the next time step is estimated for the current order, one lower order and one higher order, based on the respective local errors. If either the convergence or error test fails, the integration is restarted at the beginning of the failed time step after a new computation of the Jacobian matrix, reduction of the time step size, and/or lowering of the order of the integration formula. These procedures are automated in the Gear algorithm subject to several heuristic choices to control computational demand including:

- Update of the Jacobian matrix only after completion of a prescribed number of successful chemistry time steps, if the order changes, or if the convergence (or error test) fails.
- Halting Newton iterations if convergence progress is insufficient
- Limiting changes to the chemistry time step and the order of the method to once every $p+1$ steps for a p-th order method for stability reasons.

The Rosenbrock algorithm differs from the Gear case in that it is not a multi-order, multi-step method. One predictor iteration is followed by three corrector iterations before computing the final solution to determine a new time step increment. At the time, modifications introduced in Refs. [14, 15] took advantage of vector processing on the pipelined vector architectures of Cray computers [17]. 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 [14] was to apply the Gear algorithm to a block of grid cells simultaneously. This modification allowed vector instructions to be applied for an innermost loop 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 dimensioned by a cell index. Nevertheless, this blocking method worked well on Cray architectures with 128 word vector registers using a choice of BLKSIZE = 500 grid cells. However, such a choice has a memory copy penalty on current commodity CPUs where a choice of BLKSIZE larger than approximately 64 leads to increased computational time. Another disadvantage of choosing larger values of BLKSIZE is that the 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 (JSPARSE herafter) in Refs. [14, 15] 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.

Two additional improvements implemented in Refs. [14, 15] are applied in the JSPARSE procedure to exploit the sparse structure of the Jacobian in the direct Gaussian linear solver:

1) Terms are ordered in the Jacobian matrix to reduce fill-in when applying decomposition and forward/back substitution.
2) algebraic manipulations involving zero numerical values are eliminated in explicit "hardwired" coding using multiple levels of indirect subscript references.

Both these improvements are implemented in symbolic manipulations that need be performed only once using the known (unchanging) sparse structure of the Jacobian matrix in the JSPARSE procedure. However, the use of complex loop ranges based on indirect array references that are evaluated only at runtime, prohibits parallelization of outer loop nests. When originally developed on Cray computers the use of indirect addressing was not a major performance inhibitor because that architecture allowed hardware

gather-scatter operations. However, today's commodity architectures do not support hardware gather-scatter instructions and indirect addressing carries a penalty because it cannot be parallelized easily. It also leads to excessive translation look-aside buffer (TLB) cache look-up that inevitably stalls a commodity CPU. For some details of analysis for this performance bottleneck see Young and Delic [18]. The exception is the Intel MIC architecture [7] which does support gather/scatter operations in hardware.

In the U.S. EPA implementation of the Gear algorithm additional changes include:

- code changes to integrate into the CCTM structure,
- prohibition of negative concentration values that are possible when rapid species depletion occurs,
- choice of a relative error (RTOL) of 10^{-3} and absolute error (ATOL) 10^{-9} ppm.

It should be noted that CMAQ, unlike the GMI [12] implementation does not apply mass conservation for species.

However, error tolerance values may be changed (as input parameters in CMAQ), and they are based on heuristic proposals by Byrne and Hindmarsh [19]. It is important to note that these tolerances are applied to the L_2 norm of species errors for all cells in a block of cells. Therefore, not all individual species in a block of cells may satisfy them. Application of a mini-max norm such as L_∞ would be considerably more stringent, but is also more expensive in computation time. Also, more accurate results would be obtained with a block size (BLKSIZE) of one, i.e., a single cell. However, this choice also increases computation time substantially.

2. New Sparse Matrix Algorithm in CMAQ

For the reasons outlined in the previous Sections, a new sparse solver was developed to replace the legacy method of Refs. [14,15] and this section gives some detail on two of the major performance enhancements for CMAQ with the Gear solver. The same description applies to the Rosenbrock algorithm since it shares the same computational modules with the Gear case. The first change replaces the sparse matrix solver used for chemical species concentrations in the standard U.S. EPA distribution. 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 was previously observed to be the case in application to CMAQ with the Rosenbrock solver [11].

2.1 Gaussian Elimination in the Gear Solver

The Gear chemistry solver in CMAQ applies direct Gaussian elimination [20] 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 = 149 in this study). While the species matrix has some N^2 = 22,201 elements, the number of non-zero entries, NZ, is 1338 (day) and 1290 (night) for chemistry mechanisms, respectively. The matrix solution has 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 by many orders of magnitude, and therefore pivoting is not applied in step (i). Scaling is applied to 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. The choice of block size is the user selectable parameter (BLKSIZE) but the actual value has consequences for cache behavior on commodity CPUs at runtime [11]. For all test cases reported here the choice was limited to a default of BLKSIZE = 50.

2.2 New Sparse Matrix Solver

This section summarizes the algorithmic choices that transform JSPARSE into a new procedure (FSPARSE hereafter) for the Gear algorithm. The same method applies to the Rosenbrock chemistry algorithm that was previously described in [11]. First of all, a few words about sparse matrix storage schemes are in order. All sparse matrix algorithms reference only non-zero elements and store the value in an array, but they differ in the storage method for the row and column 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 rows and columns of the matrix and stores column and row index values in two integer arrays. Alternatively, for NZ non-zero elements in the matrix, the Compressed Column (CC) scheme scans down successive columns and uses an integer array i of length NZ together with another pointer array p of length N+1 so that row indices of entries in column j are stored in integer arrays i(p(j) through i(p(j+1) -1). The CC scheme is described in chapter 2 of Davis [21] for the C language case. In another method, Compressed Row (CR) storage scans across successive rows and uses a similar pointer scheme described for CC (above). The starting point in FSPARSE is the CSparse C language library developed by Davis [21] which uses the CC storage form and has been implemented with substantial modification in the FSPARSE version of CMAQ. The CSparse library is quite general and extensive, but only the sparse Gaussian procedures have been adopted for this CMAQ application. 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 [14]. However, P = I (the identity matrix) is the choice in the CMAQ model because the matrix A is diagonally dominant and no pivoting is applied.

The CSparse procedures listed in Table 3 have been extracted and translated into FORTRAN for integration into the FSPARSE version of the Gear algorithm. 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 ("="). This is demonstrated by the compiler message in the extract from FSPARSE shown in Fig. 1.

However, if the indirect reference is on the right hand side then parallel/vector instructions are enabled. The transformation is achieved by using a Compressed Row (CR) storage scheme as is demonstrated in Fig. 2. The suggestion for the CR form enabling a parallel/vector algorithm is from Björck [22]. Because of this benefit of the CR form, FSPARSE has an option to convert L and U to the Compressed Row (CR) storage scheme after the sparse CC decomposition step for A = LU. This enables vector SSE instructions to schedule the inner loops of forward and backward solve steps (see Section 2.1) while also allowing parallel potential in the outer loop. Such parallel loop nests may easily be parallelized in a many core version, or whenever nested parallel threads are enabled in the OpenMP model.

In the code for the solver part of FSPARSE that corresponds to Fig. 2, the forward solve is performed for all cells in a block of cells as shown in the example of Fig. 3.

Table 3 C language procedures from CSparse translated to FORTRAN in FSPARSE.

CSparse procedure	Description
cs_compress	Map Triplet to CC storage
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[a]	Solve Lz = b
cs_usolve[a]	Solve Ux = z
cs_norm	Compute 1-norm of A
[a] Converted to parallel and vector form using Compressed Row (CR) format for L and U	

```
!   inner loop _will_ not vectorize in CC format – compiler message:
!   row_f, Loop not vectorized: data dependency
!              Loop unrolled 4 times
!
        col_f: do s_j = 0, N - 1                                    ! col index
               s_x(s_j) = s_x(s_j) / s_Lx( s_Lp(s_j,sn),sn )
               row_f: do s_k = s_Lp(s_j,sn)+1 , s_Lp(s_j+1,sn)-1
                  s_x(s_Li(s_k,sn))=s_x(s_Li(s_k,sn)) - s_Lx( s_k,sn)*s_x(s_j)
               end do row_f
        end do col_f
```

Fig. 1 Example of FORTRAN version of compressed column (CC) format for a solve loop that inhibits vector instructions on the inner loop.

```
!   inner loop _will_ vectorize in CR format – complier message:
!   col_fr: Generated 2 alternate versions of the loop
!              Generated vector sse code for the loop
!              Generated a prefetch instruction for the loop
!
        row_fr: do s_i = 1, N - 1                                   ! row index
               s_s = s_x(s_i)
               col_fr: do s_j = L_w(s_i,sn) , L_w(s_i+1,sn)-2
                  s_s = s_s - L_Cx( s_j,sn ) * s_x( L_Cj(s_j,sn) )
               end do col_fr
               s_x(s_i) = s_s
        end do row_fr
```

Fig. 2 Example of FORTRAN version of compressed row (CR) format for a solve loop that does allow vector instructions for the inner solve loop.

```
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. 3 Example of FORTRAN version of Compressed Row (CR) format for a solve loop that expands the example of Fig. 2 to vector loops over blocks of cells of length NUMCELLS.

In Fig. 3 the outer row loop (row_fr1) is not parallelizable because of the recurrence on array K1. The column loop (col_fr1) 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. At the cost of a memory copy, a temporary array (rivot) is introduced so that a vector-inhibiting recurrence is avoided on the innermost loop (# 32).

2.3 Driver Procedure

The CCTM driver procedure is CHEM in CMAQ and has major loops over the blocks of cells in a grid dimensioned by column, row, and level. The MPI implementation partitions the entire grid on the column and row dimensions into sub-domains where the number of cells in each sub-domain depends on the number of MPI processes (NP). Each sub-domain has blocks of cells that are processed in the solve steps as described in Sections 2.1 and 2.2. The number of blocks is calculated from the BLKSIZE parameter choice in a grid initialization procedure GRID_CONF. However, the number of blocks diminishes as NP increases. For each MPI process the chemistry solver time step for each block is independent of all others, and different blocks are distributed 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. Thus the Gear algorithm is applied independently by each thread in the team to its own chosen block of cells.

Table 4 shows the subroutines modified in the FSPARSE version of CMAQ. This indicates those subroutines inlined into the new version of CHEM that has two large thread 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 for each MPI process, but the first takes only a small fraction of the time spent in CHEM.

The new version of CHEM was created by successive code structure modifications of the standard U.S. EPA Gear solver without changing the science of the model in any way. Specific restructuring steps applied to the standard CMAQ gas chemistry solver included:

- New modules for procedures (see Table 3)
- Arrangement of inner loops so that they enable 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.

The modified FSPARSE version of CMAQ applies a thread parallel strategy that has three prongs:

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

The two parallel regions in the FSPARSE CHEM version invoke OpenMP thread parallel teams that execute either on a host processor or on many integrated core (MIC) processors through the offload option in the Intel compiler. 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 by creating a large parallel region for the block loop and it is this loop that has a diminishing range as the number of MPI processes increases.

Table 4 The U.S. EPA procedures of the Gear solver modified in the FSPARSE algorithm.

CMAQ procedure	Description of computational function in separate modules
GRID_CONF	Define grid and set BLKSIZE
GRVARS	Declare allocatable arrays
GRINIT	Initialize and allocate arrays
JSPARSE	Define chemistry structure and symbolic Gaussian elimination
CHEM	Loop over grid blocks and call Gear solver
CALCKS	Prepare reaction rate coefficients
PHOT	Prepare photolytic rate coefficients
SMVGEAR[a]	Implementation of Gear ODE algorithm
SUBFUN	Rate of change of species concentrations
PDERIV	Jacobian matrix
DECOMP	LU decomposition
BACKSUB	Forward and backward solve

[a] Inlined into FSPARSE CHEM procedure with calls to the others in this table

3. Test Bed Environment

3.1 Hardware

The hardware systems chosen were the platforms at HiPERiSM Consulting, LLC, shown in Table 5. Nodes 20 and 21 host two Intel E5v3 CPUs with 16 cores and each node has four Intel Phi co-processor (MIC) processors [7] with, respectively, 60 and 59 cores each. These are the base nodes of a heterogeneous cluster that includes an HP blade server 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) floating point performance in single precision. The MPI executions are launched across multiple combinations of these nodes using an Infiniband (IB) fabric with a theoretical bandwidth limit of 40G bits/sec. This cluster allows for comparison of the FSARSE hybrid (MPI + OpenMP) parallel versions of CMAQ with the original EPA JSPARSE version.

3.2 Compilers

This report implemented the Intel Parallel Studio® [7] (release 17.6), for CMAQ on 64-bit Linux operating systems. The HiPERiSM Consulting, LLC, version of CMAQ, with multi-threaded parallelism, was compiled and executed for this heterogeneous cluster. Other compilers have been used in the past, but results reported here will be confined to the Intel case.

Table 5 Test bed platforms and their attributes.

Platform	Node20-21 (each node)	Node27-34 (each node)
Processor	Intel™ E5-2698v3	Intel™ E5640
Peak Gflops (SP)	~589	~170
Power consumption	135 Watts	80 Watts
Cores per processor	16	4
Processor count	2	2
Total core count	32	8
Clock	2.3 GHz	2.67 GHz
Band-width	68 GB/sec	25.6 GB/sec
Bus speed	2133 MHz	2933 MHz
L1 cache	32 KB	32 KB
L2 cache	256 KB [a]	256 KB [a]
L3 cache	40 MB	12 MB

3.3 Episode Studied

The 5.3b release of CMAQ was used in all results reported here with the source code and model episode data available at the download site [4]. This 24 hour 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. This case represent a modest grid size but is substantial enough with the number of species and reactions included.

Partitioning of the grid 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. For example, with 8 MPI processes there are approximately 700 blocks per MPI process. As a result the workload per thread is also diminished. Thus both increasing MPI process and OpenMP thread count have consequences for performance scaling because the number of blocks is further subdivided.

4. Performance

4.1 Speedup and Scaling

In this section two performance metrics are defined to assess thread parallel performance in the FSPARSE modified code for CMAQ:

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

For the CCTM each grid of cells is partitioned into blocks of size BLKSIZE and these blocks are distributed to threads in an OpenMP thread team in FSPARSE. In the previous study for the Rosenbrock algorithm [11] values of 16, 32, 48, and 64 were

investigated for impact on wall clock times due to cache effects. However, variations in wall clock time for BLKSIZE changes in this range were small and shrank as the number of threads increased. Nevertheless, wall clock time did rise for BLKSIZE greater than 64 therefore in this study of the CMAQ, the EPA default value of BLKSIZE = 50 was used.

4.2 MPI Scaling

CMAQ in the U.S. EPA JSPARSE version was scaled on the homogeneous cluster (node20 and 21) in the MPI range 1 to 64 processes for both Gear and Rosenbrock algorithms. Wall clock time (in minutes) is shown in Fig. 4 where a sharp decline in improvement is visible, especially above 8 MPI processes. The MPI parallel efficiency in Fig. 5 is calculated from speedup divided by the process count. This average reflects values of ~70% and ~55% with 32 and 64 MPI processors respectively. The latter efficiency value suggests that, on average, the CPU is idle half of the wall clock time. This is caused by the increasing dominance of MPI communication time over arithmetic compute time, specifically an MPI barrier call at the synchronization time step.

4.3 Results for one MPI Process

This section presents results for serial execution with one MPI process (NP = 1) on node20 and compares

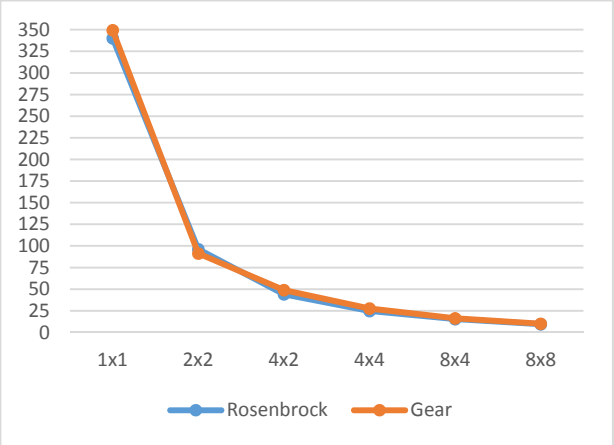

Fig. 4 Wall clock time (minutes) versus MPI process count (assigned row x column) for the EPA JSPARSE version of CMAQ for Rosenbrock and Gear algorithms.

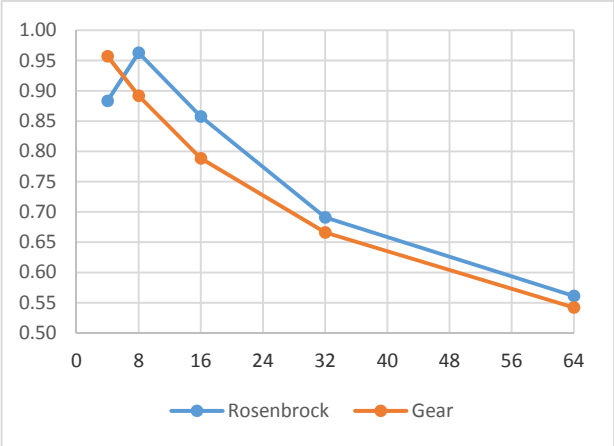

Fig. 5 Corresponding to the times of Fig. 4 this shows MPI parallel efficiency versus MPI process count for the EPA JSPARSE version of CMAQ for Rosenbrock and Gear algorithms.

JSPARSE and FSPARSE versions of CMAQ. For this discussion Table 6 defines the major CMAQ science processes and their acronyms.

To compare JSPARSE and FSPARSE version, Figs. 6 and 7, respectively, show the fraction of wall clock time as a function of science process, for Gear and Rosenbrock algorithms. The CHEM and AERO processes dominate with diminishing contributions to wall clock time for others. For the Gear case with JSPARSE (Fig. 6) the fraction of the total runtime used by CHEM dominates at ~50%. The next largest fraction, at ~26%, is AERO, and all other science processes have considerably smaller fractions. For the Rosenbrock case with JSPASE (Fig. 7) AERO is the dominant process, but CHEM is close to ~33%. Therefore any improvement in the CHEM subroutine will significantly impact the total wall clock time. Such an improvement is visible in the FSPARSE case for both Gear (Fig. 6) and Rosenbrock (Fig. 7) algorithms when comparing the effects of increasing thread count in the range 8, 12, 16 (omp8 to omp16). In the last case the FSPARSE CHEM fraction is half of the EPA JSPARSE value. This shows that the fraction of time consumed in CHEM diminishes, and as it does so the fractions of other science processes correspondingly increase.

Table 6 CMAQ science processes and the module name.

Process and function	module name
CCTM Chemical transport model	CHEM
Aerosol species processing	AERO
Asymmetric convective model (ACM) for vertical diffusion	VDIFF
Photolysis processes	PHOT
Advection in the horizontal plane	HADV
Advection in the vertical (Z) direction	ZADV
ACM and resolved cloud processes	CLDPROC
Horizontal diffusion	HDIFF
Couple concentration values for transport	COUPLE
Decouple concentration values for transport	DECOUPLE

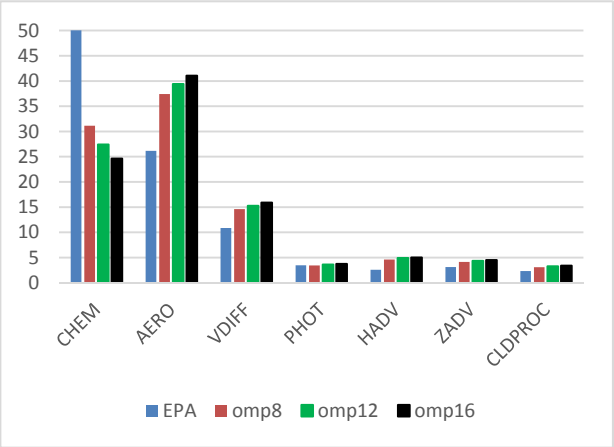

Fig. 6 Fraction of wall clock time (percent) by science process in CMAQ for the FSPARSE Gear algorithm compared to JSPARSE (EPA) for NP = 1 MPI processes and OpenMP thread counts of 8, 12, and 16 (omp8, omp12 and omp16).

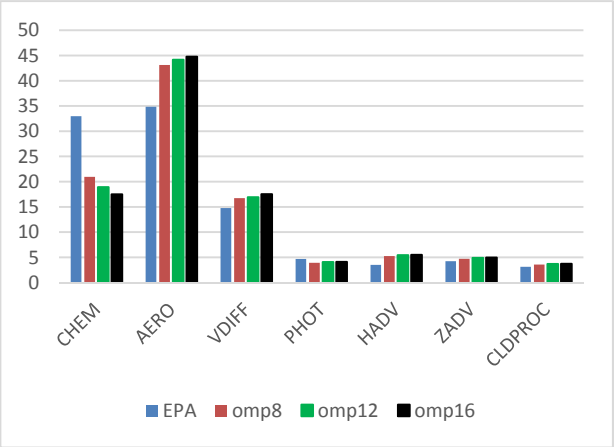

Fig. 7 Fraction of wall clock time (percent) by science process in CMAQ for the FSPARSE Rosenbrock algorithm compared to JSPARSE (EPA) for NP=1 MPI processes and OpenMP thread counts of 8, 12, and 16 (omp8, omp12 and omp16).

In more detail, Fig. 8 shows the speedup of FSPARSE over JSPARSE for thread counts of 8, 12, and 16, in 288 individual calls to CHEM for the full 24 hour simulation with the best results for 12 or 16 threads (with speedup ~3). Because of the core count limitations on the blade server (node27 to 34), a default of 8 threads is chosen for execution on the heterogeneous cluster.

Table 7 lists the total time (in minutes) expended individually for each of the physical processes in CMAQ for the 24 hour episode described in Section 3.3. The results for node20 with NP = 1 are separated for Gear and Rosenbrock CCTM algorithms in CHEM. The original (JSPARSE) results are compared with the FSPARSE version for 8 threads, and the TOTAL entry shows the speed up in parentheses: 1.32 (Gear) and 1.14 (Rosenbrock), respectively.

4.4 MPI Speedup and Scaling (Heterogeneous Cluster)

To compare the effects of increasing MPI process count for JSPARSE and FSPARE versions of CMAQ, Table 8 shows the fraction of wall clock times in MPI communication, serial computation, and OpenMP regions (in the case of FSPARSE). What was only serial computation in JSPARSE, is split into serial and OpenMP fractions in FSPARSE. Two important

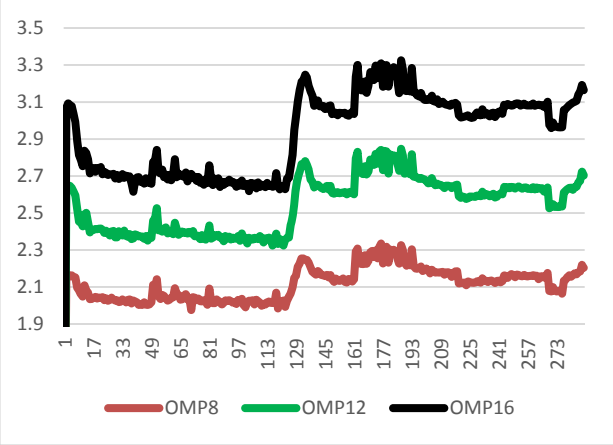

Fig. 8 Parallel thread speedup over the standard U.S. EPA model in 288 calls to CHEM with the Gear algorithm for 8, 12 and 16 threads (OMP8 to OMP16), for NP = 1 MPI process.

Table 7 CMAQ wall clock times (minutes) by science process for a 24 hour simulation with NP = 1 for the Gear and Rosenbrock algorithms in the CTM.

Science process	JSPARSE		FSPARSE (8 threads)	
	Gear	Rosenbrock	Gear	Rosenbrock
Total	496.2	390.2	374.9 (x1.32)	328.3 (x1.14)
CHEM	248.2	128.7	116.7	68.8
AERO	129.8	135.9	140.3	141.5
VDIFF	54.0	57.7	54.7	55.0
PHOT	17.3	18.3	17.4	17.2
HADV	12.9	13.7	12.9	12.9
ZADV	15.6	16.5	15.5	15.4
CLDPROC	11.6	12.2	11.7	11.7
HDIFF	2.20	2.36	2.18	2.18
Couple	1.97	2.10	1.80	1.80
Decouple	2.57	2.73	1.82	1.82

Table 8 For the heterogeneous cluster this shows CMAQ fraction of wall clock time (percent) in MPI, serial, or OpenMP time, in a one-day simulation, for the Gear algorithm in the CTM for the number of MPI processes (NP) shown in the first column.

NP	JSPARSE		FSPARSE (8 threads)		
	MPI	Scalar	MPI	Scalar	OpenMP
4	13.5	86.5	10.9	61.0	28.1
8	15.1	84.9	11.0	62.1	26.9
16	27.2	72.8	15.5	51.3	33.2

observations are that MPI process time increases with increasing NP, but less so for the FSPARSE case. Also, as expected, note the diminished scalar time in the FSPARSE case.

Fig. 9 shows the speedup of the FPSARSE version (with 8 threads) over the EPA JSPARSE original for Gear and Rosenbrock algorithms. These executions were on the heterogeneous cluster, with one MPI process on node20, and others on individual blade nodes for NP = 4, 8, 16. Speed up in the Rosenbrock case is less than that of the Gear algorithm because there is less arithmetic computation per thread (i.e., reduced computational intensity per thread). A notable feature of Fig. 9 is the diminution in speedup for NP = 16 cases. This is the consequence of two observations. First is the diminished workload per thread because of the reduced block count with increasing NP (as noted above in Section 3.3). Second is that 8 MPI processes are on each of the two fastest nodes. Overall the speedup ranges from 1.16 to 1.46 (Gear) and 1.01 to 1.25 (Rosenbrock).

4.5 MPI Speedup and Scaling (Homogeneous Cluster)

For execution on the homogeneous cluster (node20 and 21), Figs. 10 and 11 compare performance results for Rosenbrock and Gear algorithms with 8, 12, and 16 threads, on each node, and NP = 1, 4, and 8. Speedup with FSPARSE is relative to the EPA JSPARSE version executed on the same homogeneous cluster configuration. Here, for Rosenbrock, the speedup

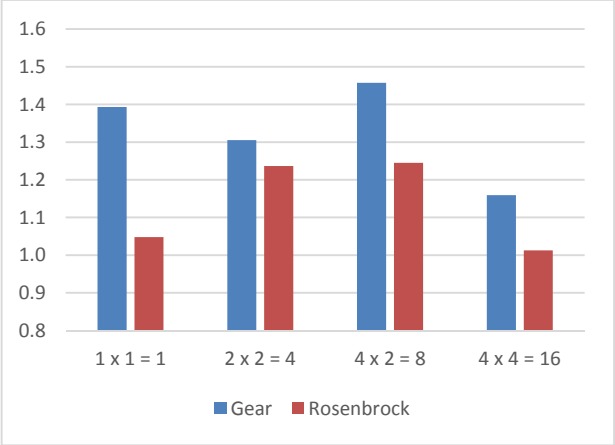

Fig. 9 Parallel thread speedup over the standard U.S. EPA model for the Gear and Rosenbrock algorithms with 8 threads, for NP = 1 to 16 MPI processes (assigned row x column) on the heterogeneous cluster.

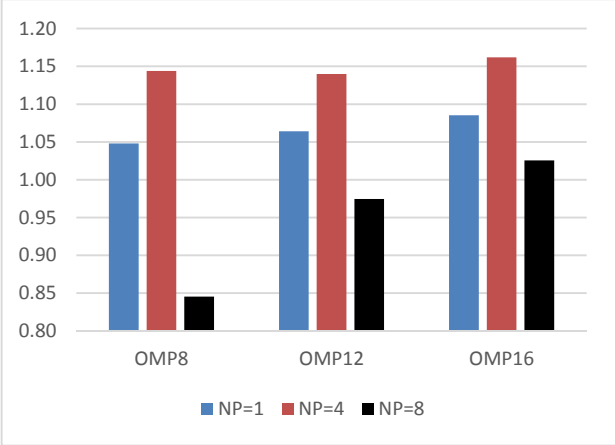

Fig. 10 Parallel thread speedup over the standard U.S. EPA model for the Rosenbrock algorithm with 8,12, and 16 threads, for NP = 1 to 16 MPI processes (assigned row *x* column) on the homogeneous cluster.

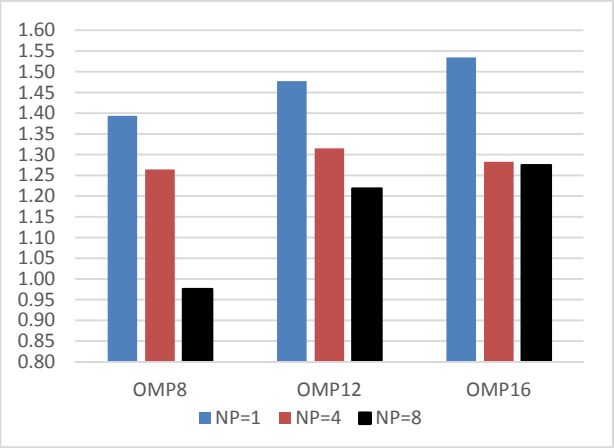

Fig. 11 Parallel thread speedup over the standard U.S. EPA model for the Gear algorithm with 8,12, and 16 threads, for NP = 1 to 16 MPI processes (assigned row *x* column) on the homogeneous cluster.

ranges from 0.85 to 1.14 (8 threads), 0.97 to 1.14 (12 threads), and 1.03 to 1.16 (16 threads). Whereas for Gear, the speedup ranges from 0.98 to 1.39 (8 threads), 1.22 to 1.48 (12 threads), and 1.27 to 1.53 (16 threads).

Speedup for the Rosenbrock algorithm is overall less than that for Gear due to less arithmetic work per thread compared to Gear. It is interesting to observe the increasing speedup for 16 threads when NP = 8, even though cores are oversubscribed. The core count is limited to 32 per node and this means that the FSPARSE case may be limited by thread population counts per node. Never the less, cores can be oversubscribed by hosting more than one thread on each. Such oversubscription occurs with16 OpenMP threads per MPI process for NP = 8 (with 4 on each node) resulting in a total 64 threads per node sharing 32 cores. For example, with 16 threads and NP = 8, speedup is in the range 1.03 (Rosenbrock) and 1.27 (Gear). Part of the explanation for this phenomenon (in the Gear case) when cores are oversubscribed, is due to the fact that calls to CHEM from different grid cell blocks are asynchronous and contention for core resources on the CPU is ameliorated.

Table 9 repeats the measurements of Table 8, but now for the homogeneous cluster case of node20 and 21 again with 8 threads. Whereas the fraction of wall clock time in MPI communication rises in the

Table 9 For the homogeneous cluster this shows CMAQ fraction of wall clock time (percent) in MPI, serial, or OpenMP time in a 24 hour simulation for the Gear algorithm in the CTM for the number of MPI processes (NP) shown in the first column.

NP	JSPARSE		FSPARSE (8 threads)		
	MPI	Scalar	MPI	Scalar	OpenMP
4	11.4	88.5	11.5	57.3	31.1
8	13.2	86.7	9.7	46.5	43.6
16	17.8	82.1	11.7	45.9	42.2

JSPARSE case, it is significantly reduced in the FSPARSE algorithm. Also the increased fraction in the OpenMP parallel region is obvious.

5. Numerical Analysis

5.1 Chemistry Convergence Criteria

To understand numerical precision this section discusses some numerical metrics that affect concentration value predictions in CMAQ. In the CCTM convergence is controlled in both Gear and Rosenbrock methods by accuracy parameters ATOL and RTOL. In the standard U.S. EPA version of CMAQ the default values chosen are RTOL = 1.E-03 and ATOL = 1.E-09 for the Gear algorithm, whereas Rosenbrock uses ATOL = 1.E-07. The choice ATOL = 1.E-09 for Gear is based on the heuristic observations in Ref. [19].

5.2 Norms in the Concentration Solution

There are two classes of error in this application of the Gear solver. The first is the global and local error metrics used in controlling the progress of the Gear, or Rosenbrock, algorithm chemistry time stepping algorithm controlled by the parameters RTOL and ATOL. The other class of error is demonstrated in metrics that show precision after the decomposition and solve steps of the sparse linear system Ax = y. Such metrics are monitored in FSPARSE with an option to calculate several types of norms including |A|, |x|, and |Ax-y|. In the CC formulation the norms are chosen as the infinity norms, norm(Ax – y, inf), norm(x, inf), and norm(y, inf), where the length of the vector (Ax-y, x, or

y) is the number of chemical species. The "inf" norm selects the maximum value of each vector. While details are not shown here these norm results suggest that the residual remains very small in the FSPARSE algorithm for the chemistry solver.

Previous study has shown that correlation between the value of ATOL and the norm of the residual for solution of the sparse linear system is negligible. This leaves open the choice that optimizes both runtime and accuracy for species concentrations.

5.3 Species Concentration Predictions

A direct comparison of accuracy for species concentration values predicted by the FSPARSE version against the U.S. EPA standard release of CMAQ is shown in Figs. 12 (a) to (d) for four selected species concentrations: O_3, CO, SO_2, and NO_2, respectively. These are absolute errors for all 8,000 concentration values of each selected species in layer 1 at the end of a one-day simulation. The solid line is the species concentration value predicted by JSPARSE for a single MPI process (NP = 1) ranked in increasing magnitude from left to right. Corresponding to each value, the difference (scattered points) is the absolute error value of the concentration between FSPARSE and the JSPARSE result. The first feature to note in the results is that O_3 and CO concentration values are of similar magnitude and differ in less than an order of magnitude over the full range. Whereas, SO_2 varies by over four orders of magnitude, and NO_2 by two orders of magnitude. Therefore a uniform precision in significant figures of accuracy would have to be more

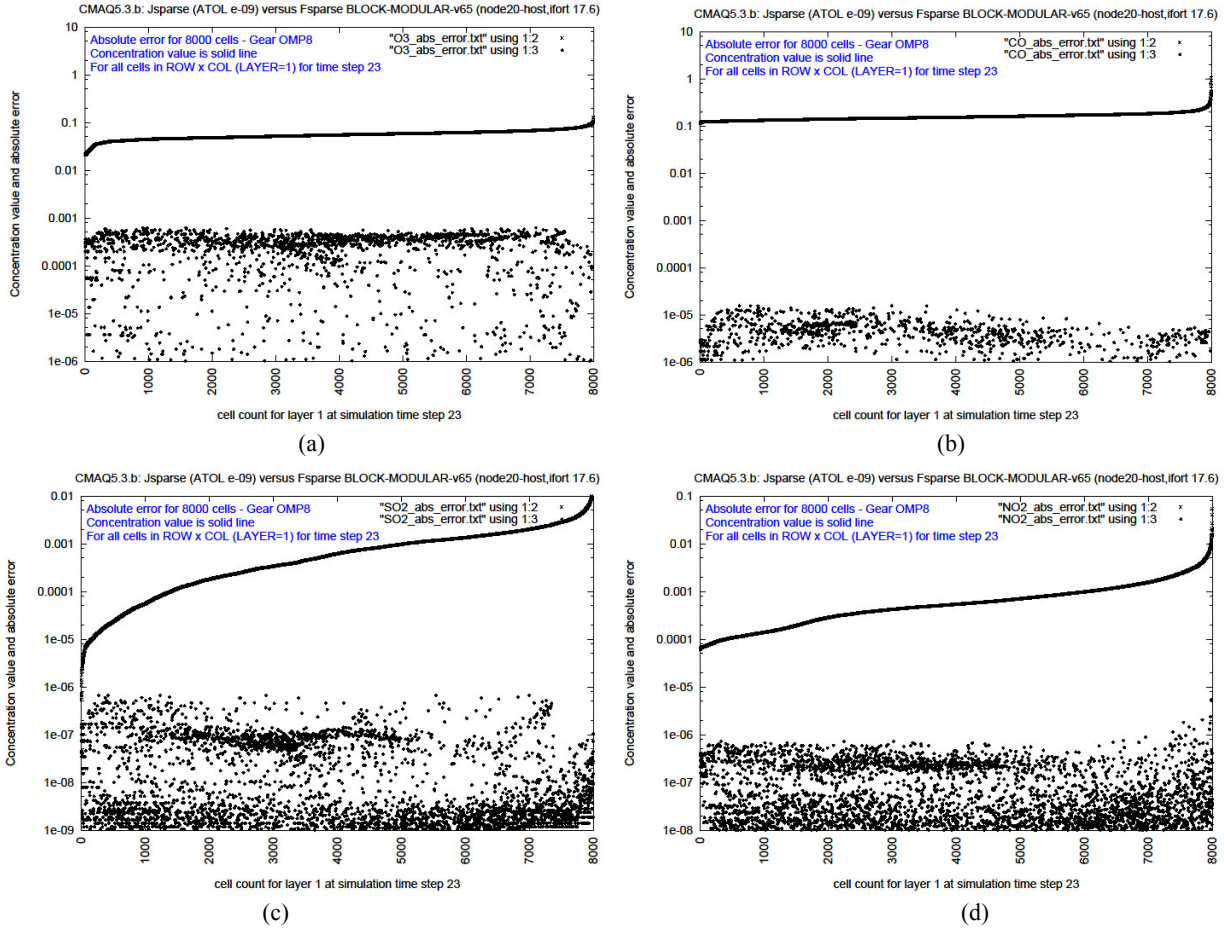

Fig. 12 For the FSPARSE GEAR solver of CMAQ (with 8 OpenMP threads) this shows the species concentration absolute error (scattered points) and concentration value (solid line) for 8000 values in layer 1 of the domain for species O_3 (a), CO (b), SO_2 (c), and NO_2 (d). The ranking is in increasing concentration value from left to right.

than 4, and this is hardly possible if a relative tolerance RTOL = 1.E-03 is applied for the L_2 norm over species concentration in the Gear convergence criterion. The second feature to note is that the absolute error threshold ranges from 1.E-02 (O_3) to 1.E-05 (CO) below the corresponding concentration value. Therefore the anticipated accuracy in the Gear solver in CMAQ differs for different species. However, in view of the precision issues noted above, these results are deemed as acceptable pending test with constrained values of RTOL and ATOL. However, such tests are limited by the use of single precision values passed to the CCTM by other CMAQ processes.

6. Lessons Learned

6.1 Benefits of the FSPARSE Method

Comparing performance for CMAQ 5.3b in the new OpenMP parallel version with the U.S. EPA release with either Gear or Rosenbrock chemistry solver showed:

- A speedup in the range 0.9 to 1.5 depending on the number of parallel thread and MPI process count.
- Comparable numerical precision in species concentration values.

6.2 Comparing Species Concentrations

A comparison of species concentration values predicted by JSPARSE and FSPARSE versions of CMAQ showed acceptable agreement for species such as O_3, NO_2, NO_3, SO_2, and others not shown. Remaining differences in species concentration values could be due to cumulative error propagation in the U.S. EPA method.

7. Conclusions

This study reported on major performance enhancements for the Community Multi-scale Air Quality Model (CMAQ) chemistry-transport model (CCTM) that add new levels of parallelism and replace the legacy algorithm in the Gear and Rosenbrock methods. The CCTM is computationally intensive when the Gear (or Rosenbrock) algorithm is used to solve a stiff system of ordinary differential equations (ODE), with sparse Jacobians, and accounts for over 50% (or 33%) of the wall clock time of a simulation. To improve performance two important changes were made, the first of which replaced the sparse matrix solver. The second modification integrated the new solver into the transit over the grid domain so that separate blocks of cells are distributed to different threads in a team. The resulting sparse solver (FSPARSE) replaced the legacy JSPARSE sparse method. The FSPARSE solver is portable across hardware and compilers that support vector and thread parallelism and it adds both to the existing distributed memory (message passing) level in the standard EPA CMAQ release. Observed numerical differences between the two methods are related to the numerical precision achieved in each, and were observed to be due (in part) to the way arithmetic precision is treated in the U.S. EPA method. On Intel platforms a 24-hour simulation on a continental U.S.A. grid of 280,000 cells, showed that with 8 to 16 threads the FSPARSE version of CMAQ typically provides significant speedup over the standard EPA release without loss of precision in predicted concentration values.

References

[1] U.S. EPA, Office or Research and Development, National Exposure Research Laboratory (NERL), Computational Exposure Division, available online at: https://www.epa.gov/aboutepa/about-national-exposure-research-laboratory-nerl-computational-exposure-division-ced.

[2] Community Modeling and Analysis System, available online at: http://www.cmascenter.org/cmaq/.

[3] University of North Carolina, Institute for the Environment, available online at: https://ie.unc.edu/.

[4] CMAQ, available online at: https://www.epa.gov/cmaq, https://github.com/USEPA/CMAQ.

[5] U.S. EPA Clean Air Act, available online at: https://www.epa.gov/laws-regulations/summary-clean-air-act.

[6] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach* (4th ed.), Morgan Kaufmann Publishers, Elsevier, Amsterdam, 2007.

[7] Intel Corporation, available online at: http://www.intel.com.

[8] D. W. Byun and J. K. S. Ching, Science algorithms of the EPA Models-3 community multiscale air quality (CMAQ) Modeling System, United States Environmental Protection Agency, Office of Research and Development, Washington, DC, EPA/600/R-99/030, March 1999, available online at: https://www.cmascenter.org/cmaq/science_documentation.

[9] Leon Lapidus and John H. Seinfeld, *Numerical Solution of Ordinary Differential Equations*, Academic Press, New York, 1971.

[10] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in FORTRAN* (2nd ed.), Cambridge University Press, New York, 1992.

[11] G. Delic, *12th Annual CMAS Conference*, Chapel Hill, NC, October 28-30, 2013, available online at: https://www.cmascenter.org/conference/2013/agenda.cfm.

[12] T. Clune, M. R. Damon and G. Delic, *14th Annual CMAS Conference*, Chapel Hill, NC, 2015, available online at: https://www.cmascenter.org/conference/2015/agenda.cfm.

[13] C. William Gear, The automatic integration of ordinary differential equations, *Comm. ACM* 14 (1971) 176-179.

[14] M. Jacobson and R.P. Turco, SMVGEAR: A sparse-matrix, vectorized Gear code for atmospheric models, *Atmos. Environ.* 28 (1994) 273-284.

[15] M. Z. Jacobson, *Fundamentals of Atmospheric Modeling* (2nd ed.), Cambridge University Press, New York, 2005.

[16] C. William Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, New Jersey, 1971.

[17] Cray computer, available online at: http://www.cray.com/, http://en.wikipedia.org/wiki/Vector_processor.

[18] Jeffrey O. Young and G. Delic, *7th Annual CMAS Conference*, Chapel Hill, NC, October 6-8, 2008, available online at: https://www.cmascenter.org/conference/2008/agenda.cfm.

[19] G. D. Byrne and A. C. Hindmarsh, Stiff ODE solvers: A review of current and coming attractions, *J. Comput. Phys.* 70 (1987) 1-62.

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

[21] T. A. Davis, *Direct Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2006.

[22] Åke Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.