EFFICIENT PARALLEL I/O IN COMMUNITY ATMOSPHERE MODEL (CAM)

Yu-Heng Tseng[1]
Chris Ding[2]

Abstract

Century-long global climate simulations at high resolutions generate large amounts of data in a parallel architecture. Currently, the community atmosphere model (CAM), the atmospheric component of the NCAR community climate system model (CCSM), uses sequential I/O which causes a serious bottleneck for these simulations. We describe the parallel I/O development of CAM in this paper. The parallel I/O combines a novel remapping of 3-D arrays with the parallel netCDF library as the I/O interface. Because CAM history variables are stored in disk file in a different index order than the one in CPU resident memory because of parallel decomposition, an index reshuffle is done on the fly. Our strategy is first to remap 3-D arrays from its native decomposition to z-decomposition on a distributed architecture, and from there write data out to disk. Because *z*-decomposition is consistent with the last array dimension, the data transfer can occur at maximum block sizes and, therefore, achieve maximum I/O bandwidth. We also incorporate the recently developed parallel netCDF library at Argonne/Northwestern as the collective I/O interface, which resolves a long-standing issue because netCDF data format is extensively used in climate system models. Benchmark tests are performed on several platforms using different resolutions. We test the performance of our new parallel I/O on five platforms (SP3, SP4, SP5, Cray X1E, BlueGene/L) up to 1024 processors. More than four realistic model resolutions are examined, e.g. EUL T85 (~1.4°), FV-B (2° × 2.5°), FV-C (1° × 1.25°), and FV-D (0.5° × 0.625°) resolutions. For a standard single history output of CAM 3.1 FV-D resolution run (multiple 2-D and 3-D arrays with total size 4.1 GB), our parallel I/O speeds up by a factor of 14 on IBM SP3, compared with the existing I/O; on IBM SP5, we achieve a factor of 9 speedup. The estimated time for a typical century-long simulation of FV D-resolution on IBM SP5 shows that the I/O time can be reduced from more than 8 days (wall clock) to less than 1 day for daily output. This parallel I/O is also implemented on IBM BlueGene/L and the results are shown, whereas the existing sequential I/O fails due to memory usage limitation.

The International Journal of High Performance Computing Applications, Volume 22, No. 2, Summer 2008, pp. 206–218
DOI: 10.1177/1094342008090914

Figures 1–8 appear in color online: http://hpc.sagepub.com

Key words: CAM, climate modeling, index reshuffle, parallel I/O, parallel netCDF

1 Introduction

High resolution century-long global climate simulations using the NCAR community climate system model (CCSM) have become routine practice in understanding long term climate and provide scientific background/support for climate related policy (i.e. CCSM is the US flagship model). Software design and parallel algorithms of CCSM have been published in a special issue in the *Journal of High Performance Computing and Applications* (Vol. 19, No. 3, August, 2005). These simulations generate a tremendous amount of data. Efficient I/O is a crucial factor for such large-scale simulations on massively parallel machines, but CCSM currently uses a sequential I/O scheme (gathering data onto a single processor and then processing I/O on the single processor). This is becoming a major bottleneck for high resolution simulations.

In this paper, we describe the details of a new parallel I/O for the community atmosphere model (CAM), the atmosphere component of CCSM, to facilitate efficient and flexible parallel I/O. Given an example of the CAM Finite Volume version, 0.625° × 0.5° D-resolution simulation, the standard model history output produces ~800 MB data for each output. Assuming monthly history output (the most common output interval for climate models), at least 960 TB data will be generated for a typical century-long simulation. It is also one of the most CPU intensive parts of CCSM (see User's Manual1). In many parallel environments, I/O operations can proceed on multiple I/O channels through multiple processors, greatly increasing the data transfers. The parallel I/O of CAM combines a remapping of 3-D arrays with the existing parallel netCDF library. The remapping from its native decomposition to z-decomposition uses the ideas of index reshuffling (e.g. Fraser 1976; Edelman, Heller, and Johnsson 1994; Ding 2001). The index reshuffle is used to achieve maximum I/O bandwidth on a distributed architecture because z-decomposition is consistent with the last array dimension in disk file (Ding and He 1999; Ding 2001). Therefore, the entire data subdomain in a processor will be transferred to a contiguous location in the file space without any reshuffling (thus achieving the maximum bandwidth). Furthermore, history variables are stored in disk file in a different

[1]DEPARTMENT OF ATMOSPHERIC SCIENCES, NATIONAL TAIWAN UNIVERSITY, NO. 1, SEC. 4 ROOSEVELT RD., TAIPEI, 106, TAIWAN
(YHTSENG@AS.NTU.EDU.TW)

[2]COMPUTATIONAL RESEARCH DIVISION, LAWRENCE BERKELEY NATIONAL LABORATORY BERKELEY, CA94720

index order than the one in CPU resident memory due to different parallel decompositions and dynamic cores.

The new parallel I/O strategy also incorporates the latest parallel netCDF (PnetCDF) library developed at Argonne National Laboratory (ANL) and Northwestern University (Li et al. 2003). NetCDF is a simple, portable and self-describing file format/system. PnetCDF provides an easy-to-use interface of parallel I/O capability for netCDF data and optimized collective I/O. The library is implemented on top of MPI-IO, specified by the MPI-2 standard (Gropp et al. 1996; Gropp, Lusk, and Thakur 1999). This may resolve a long standing I/O issue because climate models use netCDF data format extensively. The I/O bottleneck has so far not hindered model simulations very much because of the relatively low resolution (e.g. controlled T42 and even T85 resolution runs; these two runs represent the spectral dynamical core with triangular spectral truncation at 42 and 85 wavenumbers, respectively) and monthly output I/O interval. However, with steadily increasing resolutions and short output interval (weekly and even daily), the demand for an efficient parallel I/O becomes urgent in the near future (Collins et al. 2006). To the best of the authors' knowledge, our incorporation of PnetCDF into CAM is the first successful experience for incorporating PnetCDF in a large-scale climate model. More model details of CCSM and CAM can be found in the special issues in this journal and *Journal of Climate* (e.g. Collins et al. 2006). We note that besides PnetCDF, there are several general purpose parallel file systems such as GPFS on IBM platforms, the Lustre file system on Cray platforms, and parallel virtual file systems (PVFS) developed at ANL (e.g. Carns et al. 2000; Prost et al. 2001).

We emphasize that this novel approach of combining the PnetCDF interface with a generic remapping to z-decomposition strategy can be used for a great variety of domain decompositions to optimize parallel I/O performance. This approach is flexible and can be easily extended to other applications. We also take advantage of the optimized MPI-IO collective I/O in PnetCDF. In this sense, it is a generic parallel I/O strategy to relax the single I/O bandwidth and memory limitation using sequential I/O and provides maximum parallel I/O rate regardless of decompositions.

We have completed the implementation of this approach in CAM, the major component in CCSM. A parallel I/O module is available for CAM users as a new I/O option.[2] The extension of this approach is straightforward for other components of CCSM, including the community land surface model (CLM) which uses a complicated unstructured grid arrangement (Collins et al. 2006). Our benchmark tests on several platforms show that this approach significantly improves the I/O performance of CAM and removes I/O bottleneck. The extensive tests use up to 1024 processors on IBM SP3. The maximum speedup scales quasi-linearly with the increasing domain size. A detailed performance analysis is given later.

We note that CAM is a large code (more than 500,000 lines of Fortran) and the parallel I/O implementation requires very substantial efforts. Therefore, the performance comparison is limited to the parallel I/O of this paper (actual codes in operation) and the earlier sequential I/O. This paper describes results and experience of the parallel I/O implementation for a widely used application. Although the parallel I/O strategy turns out to be well suited to CAM, no claim has been made that this is the best possible strategy. Our experience also indicates that a stand-alone algorithm often performs differently when being integrated in a large complex code. Software consistency also plays a significant role affecting algorithm design.

2 Model Description and Parallelism

2.1 Model Description

CAM (the current version is 3.1) is the latest global atmosphere model developed at the National Center for Atmospheric Research (NCAR). A detailed model description can be found in Collins et al. (2006), and the design of its parallel algorithms can be found in Hack et al. (1995). It covers the entire surface of earth using a uniform grid in longitude and latitude. A standard T85 resolution has 256 (longitude) $\times$ 128 (latitude) grid points, and the distance between two grid points is 1.4° (150 km). The vertical (height) dimension has 26 layers (see Table 1 for more details). The model solves the sim-

Table 1
Representative CAM simulations used in our performance tests.

Runs	Domain size	Horizontal resolution	Averaged data amount
ELU T85	$128 \times 256 \times 26$	~1.4° (150 km)	123 MB
FV B-resolution	$144 \times 91 \times 26$	$2° \times 2.5°$	49.2 MB
FV C-resolution	$288 \times 181 \times 26$	$1° \times 1.25°$	196 MB
FV D-resolution	$576 \times 361 \times 26$	$0.5° \times 0.625°$	827 MB

plified version of Navier–Stokes equations (primitive equations). The dynamics of evolving states can use one of the following three dynamic cores: Eulerian (EUL), semi-Lagrangian (SLD), and finite-volume (FV). Different variable arrangements are used and the array size differs for different variables, for example 2-D variables (fields) such as precipitation and 3-D variables such as temperature or wind velocity. The physics part goes column-wise along the vertical direction involving clouds, radiation, etc. Other chemical tracers, such as aerosols, can be included as well.

2.2 Parallel Decomposition and Implications for I/O

At present, CAM uses a parallel domain decomposition to obtain optimal performance on distributed memory parallel processors. Groups of processors are assigned to each subdomain. Most data fields such as velocity, temperature and pressure, are 2-D or 3-D fields (sixty-one 2-D variables and thirty-four 3-D variables for the default CAM output). For historical reasons and vector computer architecture optimization, the index in resident memory is in *A(ix,iz,iy)* order where *ix* is the longitudinal direction, *iy* is the latitude direction and *iz* is the vertical height. However, the output files use the standard index order: *A(ix,iy,iz)*. Thus an index swapping is always required to process I/O in CAM.

CAM uses either a 1-D or a 2-D domain decomposition. In 1-D domain decomposition, each subdomain is only a subset of latitude lines while longitude and vertical dimensions are entirely local. For the EUL and SLD dynamical cores, this *y*-decomposition (latitude) is chosen because of the constraint of spectral dynamics and the semi-Lagrangian advection transport algorithm near the poles.

In 1-D decomposition, the *iy* index (latitude) is split into different processors. However, it places relatively severe limitation on the maximum number of horizontal subdomains. In the FV dynamic core, either 1-D or 2-D domain decomposition can be used. Two-dimensional decomposition allows CAM to effectively utilize more processors in reality, so that *iz* and *iy* can be split.

Two major issues complicate the parallel I/O in CAM: (a) Challenges arise for multiple decompositions in CAM, which requires multiple parallel manipulations; (b) The index swapping could significantly slow down I/O if not properly implemented. This is because index swapping involves lots of data movements with very small block sizes. If we directly move each small block from computer memory to the appropriate location in disk file space, the overhead associated with the large number of transfers is very significant.

Thus, we resolve these problems by a unique remapping strategy, i.e., irrespective of computational decomposition. Before processing I/O, the required data arrays are remapped into a *z*-decomposition. From there we use PnetCDF to write data out collectively. The remapping also performs the index swapping between *iy* and *iz*. Therefore, the entire data array in each processor is written out simultaneously, reaching the maximum possible data transfer rates.

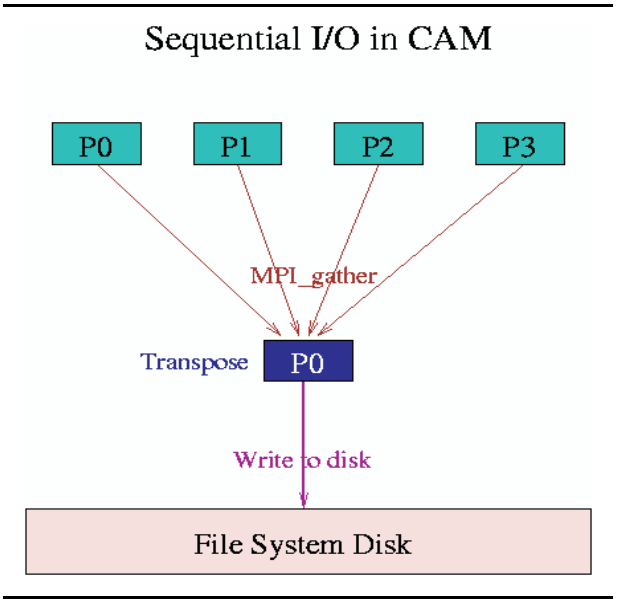

Fig. 1 The default sequential I/O strategy in CAM gathers all data blocks from computing processors and swaps (transposes) data to correct index order before writing out to disk file.

2.3 Sequential I/O in CAM

Existing CAM uses a sequential I/O scheme (as shown in Figure 1). First, the distributed 3-D array is collected into a single processor, typically Processor 0. The index of latitude and height is then swapped locally on the I/O processor. Finally, the data is written out using the netCDF file system (Rew et al. 1997; Unidata[3]). It is clear that the above steps are not scalable. The first step is limited by single processor I/O bandwidth. Furthermore, as the resolution increases to a few kilometers, the memory in a single processor is not sufficient to hold the global 3-D field of single variable. The I/O processor uses its local memory at the buffering stage. So a severe memory-capacity bottleneck also occurs. These limitations are already observed in the current CAM FV D-resolution runs and will become much worse in even higher resolution simulations (Wehner, Oliker, and Shalf 2007).

3 Parallel I/O Strategy

3.1 Three-Dimensional Field Remapping

As explained in Section 2.2, remapping is used to resolve multiple computational decompositions and index swapping. Here we further describe details of the general remapping procedure. Given a 3-D array in any decomposition, either 1-D, 2-D or 3-D decomposition, we have written a library of codes which can remap the distributed 3-D array into *z*-decomposition on a subset of processors ("I/O staging processors"). The staging processors could range from 1 to *P* (the total number of processors used by CAM). Index swapping is also performed simultaneously during the remapping procedure. These remapping and index swapping routines are collected in a library called ZioLib which is available free online (Yang and Ding 2003).[4]

The index swapping is essentially an index reshuffle of a global multi-dimensional array on a multi-processor distributed-memory system (Ding 2001). It involves local index reshuffle and global data exchange. The objective is to remap a 3-D array on processors such that data points along a particular dimension are entirely local on the processor, and that the data access along this dimension corresponds to the fastest running storage index, just as the usual array reshuffle. We used the well-known communication algorithm of transposition of 2-D arrays on distributed memory environments (Kumar et al. 1994).

In the distributed memory architecture, the remapping procedure implements a global all-to-all exchange of data blocks with size (*Nx*, *Ny/P*, *Nz/P*), where *Nx* (*Ny*, *Nz*) is the number of grid points along the *ix* (*iy*, *iz*) direction, and *P* is the total number of distributed processors. The global exchange performs essentially a pairwise block exchange where the local 3-D arrays on each processor are viewed as 1-D array of blocks (Bokhari 1991; Foster and Worley 1997)**.** This exchange involves all-to-all communication. Each processor sends *P*–1 blocks out to all other different processors and receives *P*–1 blocks from all others as follows:

```
% All processors execute the following global
    all-to-all blocks exchange simultaneously
do m=1, P–1
    send a message to destination processor d-ID
    receive a message from source processor s-ID
enddo
```

The destination ID (*d-ID*) and source ID (*s-ID*) are determined by setting

d-ID=MOD(*myID*+m, P), *s-ID*=MOD(*myID*-m, P)

here *myID* is the local processor ID and the total number of processors can be arbitrary. These sends/receives can be implemented with MPI_sendrecv, requiring a buffer of size *Nx x (Ny/P) x (Nz/P)*.

Note that the algorithm assumes *Ny* and *Nz* are integer multiples of *P* for optimal efficiency. In practical applications, such as CAM, *Ny* and *Nz* can be arbitrary. In the case of *Ny*, *Nz* is not multiples of *P*, the algorithm remains the same except that some holes in the final remapped array should be squeezed out. When both *Ny*, *Nz* are not multiples of *P*, some padding is required prior to the remapping and the holes need to be squeezed out after remapping.

PnetCDF interface facilitates an efficient parallel I/O to access a single netCDF file (Rew et al. 1997; Li et al. 2003). We will briefly describe the PnetCDF library in the next section. It is natural to implement PnetCDF directly in CAM and other large-scale models. However, satisfactory parallel scaling is not always observed in the real application (e.g. Yang, Folk, and McGrath 2004a; Yang, McGrath, and Folk 2004b). The I/O rate may not be optimal depending on the array's index order and the results may not be consistent with the sequential I/O. On the other hand, our remapping procedure to *z*-decomposition provides an efficient and flexible parallel tool to remap the distributed array and output data in parallel. Thus, our new strategy is to remap all distributed fields initially. The procedure is schematically shown in Figure 2 as the first stage. The software uses MPI collective operations and data structures and is directly implemented within CAM.

Given the example of CAM FV 2-D domain decomposition, the essential remapping involves the last two dimensions of the 3-D array. All data along the first dimension are entirely local to a processor. They are moved around during the dynamical remapping as a single block. The implementation of the above generic algorithm in CAM is simplified so that the index reshuffle is done locally and PnetCDF is responsible for the all-to-all communication.

3.2 Parallel netCDF (PnetCDF)

PnetCDF is an easy-to-use application programming interface (API) for storing and retrieving netCDF files in parallel. It is built on top of MPI-IO. PnetCDF is then used to write the *z*-decomposition data to disk file as the second stage in our implementation. Note that PnetCDF has been directly imported in the Regional Ocean Modeling System (ROMS; Yang et al. 2004a, 2004b) and many other scientific applications. Significant improvement/advantage of using PnetCDF is shown using collective I/O in ROMS when the written record size is large enough to overcome the overhead from MPI-IO. Since the data layout of the remapped array on the staging processors' memory is the same as that on disk, only block data transfer occurs during parallel I/O, achieving the maximum efficiency in collective I/O, see Figure 2. In general, the

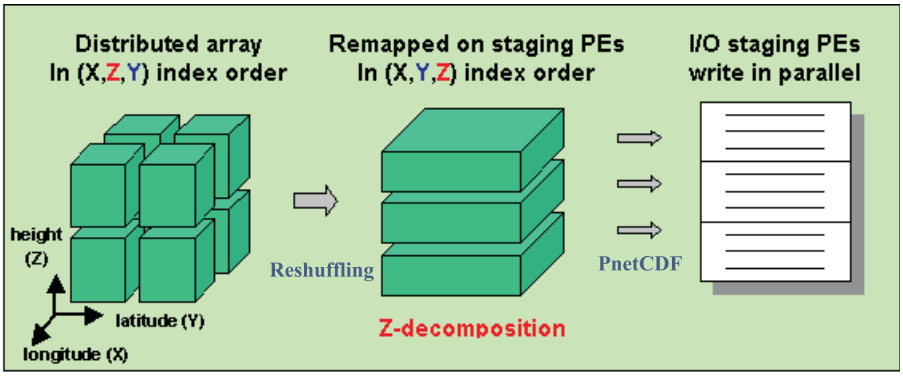

Fig. 2 The new parallel I/O strategy writes the global field of distributed array (X,Z,Y) to a disk file in (X,Y,Z) order using three I/O staging processes.

optimal performance relies mainly on the local remapping algorithm and on the easy-to-use PnetCDF as a collective I/O interface. This procedure can be applied to other large-scale applications if the output order is not contiguous, such as CAM.

We take advantage of MPI-IO collective I/O within PnetCDF for the optimal performance. The collective I/O optimization is already implemented in PnetCDF. If the contiguity information is missing, other optimization information from users, such as the number, order and record indices, is required (Li et al. 2003). Without such information, the PnetCDF cannot collect multiple I/O requests over a large amount of output variables and optimize the I/O over a large pool of data transfers. The index reshuffle could be taken care of inside PnetCDF and the remapping can be eliminated by passing an MPI_Info hint in. However, this will complicate the implementation of PnetCDF in CAM due to different I/O structures and will require experienced users to tune CAM for better performance gain. The resulting performance is unsatisfactory because of the small block-sizes and many other constraints. This is also confirmed in tests for several simple runs. Furthermore, the code structure remains relatively simple by separating remapping/index swapping and PnetCDF, which also improves code maintenance, readability, and flexibility. We further show that the reshuffle only takes a small portion of the total I/O time in Section 4, and the optimized PnetCDF collective I/O indeed dominates.

3.3 Parallel I/O Implementation in CAM

CAM is a large and complicated Fortran code containing more than 500,000 lines. NetCDF self-description files for the large number of fields have to be modified to accommodate the new parallel remapping algorithms and the use of PnetCDF. Our parallel I/O implementation takes approximately 3000 lines of code, separated in about 20 subroutines (details are skipped here). The implementation is designed as a parallel I/O module, so that CAM users can switch between the sequential I/O (default) and the parallel I/O. PnetCDF is still under development, and our parallel I/O in CAM is perhaps the first successful experience of incorporating PnetCDF in a large-scale, widely used climate model (and fixing a number of bugs of the library on the way). The PnetCDF has been used successfully in a large scale ASC/Alliances Center for Astrophysical Thermonuclear Flashes application (FLASH code[5]). We closely interacted with the developers and pointed out quite a number of bugs which were fixed during the course of this implementation. By incorporating the novel remapping procedure to optimize I/O bandwidth, our benchmark testing was perhaps the first to show significant improvement of PnetCDF in real large-scale applications.

Current Status. The implementation and benchmark testing has been done for all three dynamic cores for history related output (EUL, SLD, FV; see Section 2), and confirmed bit-for-bit agreement between the newly generated output and that generated by the existing CAM 3.1 sequential I/O. The code has been successfully ported to IBM SP series (SP3, SP4 and SP5); Cray X1E; BlueGene/L (BG/L) systems. We eliminate gather/scatter from the processors and use only local arrays within them. This significantly relaxes the memory constraint on the BG/L system. All three dynamic cores of CAM are examined and successfully validated.

3.4 A Summary of the New Remapping + PnetCDF Approach

Existing CAM 3.1 uses sequential I/O to gather all 2-D and 3-D fields into processor 0, and writes them out from there. Using the new approach, no such collective gathering exists. Instead, efficient MPI-IO collective I/O is performed on each processor. Therefore, not only the I/O speed is improved, the large memory requirement in processor 0 is also lifted. We list the major benefits here:

- Relieve memory limitation on each processor.
- Relieve congestion on I/O server nodes.
- Write/read in large blocks (no seeks) in parallel with maximum flexibility.
- Achieve robust and maximum block transfer rate regardless of parallel decomposition.
- Eliminate a temporary global field from user codes (gather/scatter/transpose).
- Improve user-friendly PnetCDF interface.

4 Performance

4.1 Performance Analysis

Here, we present the performance of CAM history I/O using the new parallel I/O algorithm, and compare with the existing sequential I/O using up to 1024 processors. Five different testing platforms and their configurations are listed briefly as follows:

1. IBM SP3 (Seaborg at LBNL/NERSC): 416 Nighthawk (NH) II 16-way SMP nodes and Colony Switch. Each node has two interconnect interfaces. Each processor is 375 MHz Power 3+ processor. GPFS file system.
2. IBM SP4 (Bluesky at NCAR): 31 32-way and 76 8-way SMP nodes and Colony Switch. Each processor is 1.3 GHz Power 4 P690 processor. GPFS file system.
3. IBM SP5 (Bluevista at NCAR): 78 8-way SMP nodes and IBM High Performance Switch (HPS). Each processor is 1.9 GHz Power 5 p575 processor. GPFS file system.
4. Cray X1E at Oak Ridge National Laboratory (ORNL): 1024 multistreaming vector processors (MSPs). Each MSP has 2 MB of cache and a peak rate of 18 GF. The interconnect functions as an extension of the memory system. NFS file system.
5. BlueGene/L at ANL: 1024 dual PowerPC 440 700 MHz, 512 MB nodes. Network uses IBM BlueGene Torus, Global Tree and Global Interrupt. NFS or PVFS2 file system.

We use realistic resolution in practical CAM simulations. The domain size, horizontal resolution and averaged output data amount are listed in Table 1. These tests include the standard spectral ELU T85 and FV B- to D-resolutions which cover a large range of resolutions. The averaged output data is based on 11 runs for all cases except the highest resolution (FV D-resolution), which uses 5 runs only. This is because the output of FV D-resolution simulation generates a large amount of data (see Table 1). The replications of each run are reasonable with a maximum deviation of approximately 15%. Note that the current CAM FV D-resolution was not ready for Linux cluster. Therefore, we skipped the implementation on the Linux cluster although it is an important type of platform in the parallel computing community. In the benchmark runs, we set the number of parallel I/O staging processors to the total number of MPI-tasks if that is less than or equal to 32. Otherwise, we set it to 32.

Figure 3 shows the average I/O bandwidth vs. total number of processors for a standard ELU core T85 run on IBM SP3. Comparing the I/O bandwidth, the speedup on the IBM SP3 for the parallel I/O vs. sequential I/O is around 4. Performance on different numbers of processors in a MPI-task is also shown. "Thread=8" refers to 8 threads per MPI-task, or 2 MPI-tasks per SP3 node. The use of "thread" is a particular feature for the IBM Power system. The number of concurrent I/O calls is associated with the number of MPI-tasks instead of "thread." For a computing node, increasing the number of threads will reduce the number of MPI-tasks. Thus a 256-processors run using "thread=4" has 64 MPI-tasks on 16 SMP nodes (i.e. IBM SP3[6]). The number of processors (or threads) in a MPI-task has clear impacts on the performance, showing the computer architecture dependency, particularly a characteristic of IBM SP, which is not clear in our evaluation. Figure 3 shows a general trend of better I/O performance as the threads per MPI-task increase. Thread=8 or thread=16 has the best performance. The worst performance is observed when a single thread per MPI-task is used. Also, as expected, the performance increases (before it saturates) when a larger number of processors are used.

Significant improvement is found in the higher resolution simulation, FV D-resolution configuration (576 longitudes × 361 latitudes × 26 levels) in Table 1. The benchmark tests use up to 1024 processors on IBM SP3 (SEABORG at NERSC). Figures 4 and 5 show the performance on IBM SP3 and SP5, respectively. The I/O performance of IBM SP4 is quite similar to the latest SP5, thus the curves are not presented herein. For this configuration, the history I/O speeds up by a factor of 14 (~139 MB/s vs. ~10 MB/s) on IBM SP3 and a factor of 9 (~380 MB/s vs. ~44 MB/s) on IBM SP5. IBM SP5 has better I/O rate (about a factor of 3 faster than IBM SP3). The two general trends still hold: the total I/O per-

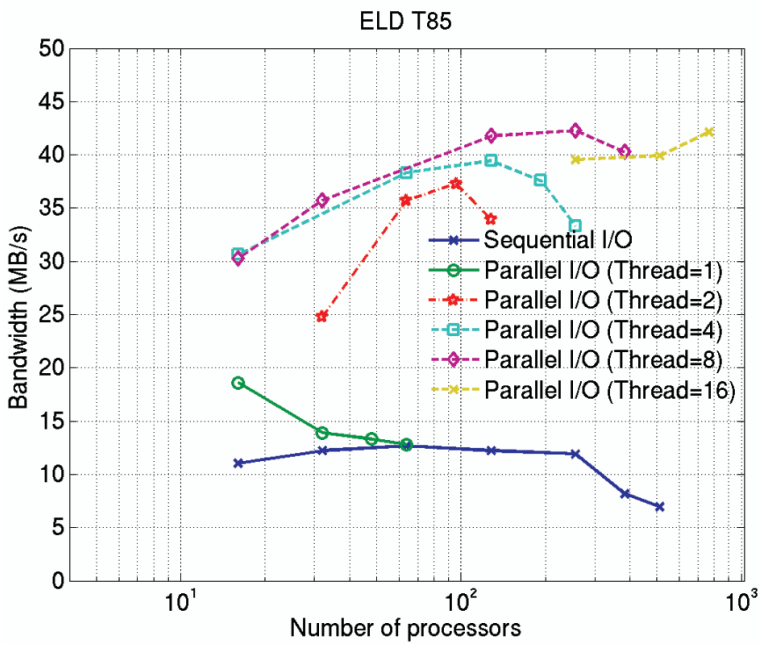

Fig. 3 History I/O bandwidth of parallel I/O EUL T85 resolution on IBM SP3 (16 processors per SMP node). "Thread=8" refers to 8 threads per MPI-task, or 2 MPI-tasks per SP3 node (16 processors).

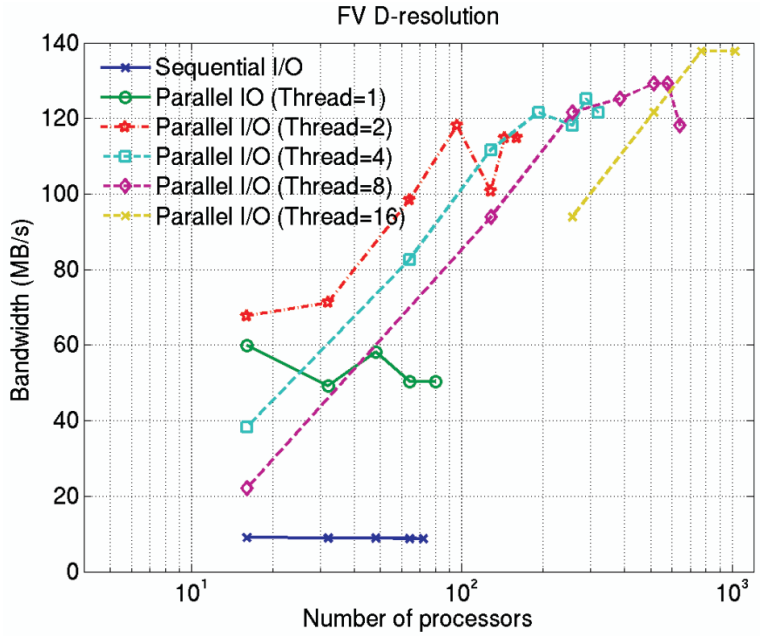

Fig. 4 History I/O bandwidth of parallel I/O for CAM 3.1 FV D-resolution (0.5°x.625°) on IBM SP3 (16 processors per SMP node).

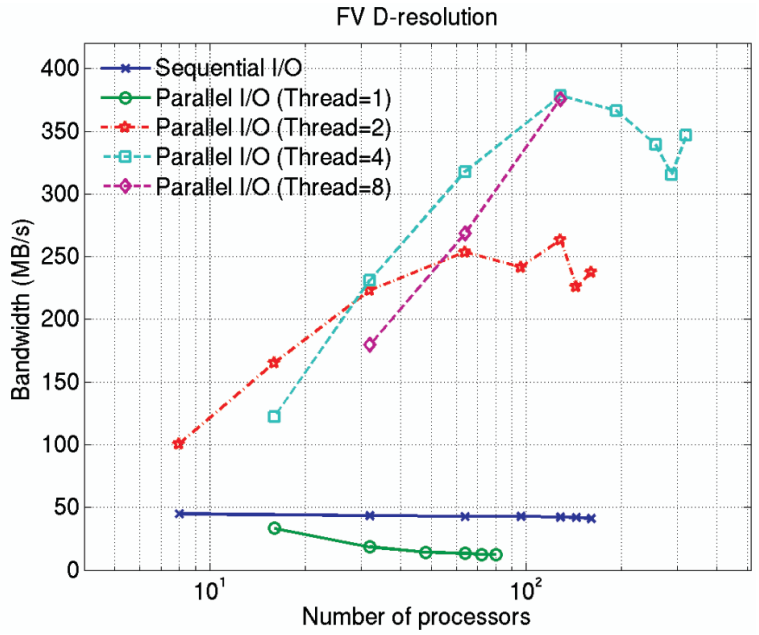

Fig. 5 History I/O bandwidth of parallel I/O for CAM 3.1 FV D-resolution (0.5°x.625°) on IBM SP5 (8 processors per SMP node).

formance increases (a) as the number of threads increases while the total number of processors is fixed; and (b) as the total number of processors increases while the number of threads is fixed.

Overall, the parallel I/O performance at thread=1 is generally poor (Figures 3 and 5, particularly). There is a dramatic gap between the thread=1 and thread=2 cases (Figures 3–5). We believe this arises from the fact that CAM is compiled with OpenMP always turned on (along with a number of other rather complicated settings) associated with IBM SP architecture. Sometimes, adopting OpenMP with thread=1 has relatively large overhead. This could explain the large degradation of CAM parallel I/O performance at thread=1.

Further, the thread=1 case corresponds to the case where every CPU within a node runs an MPI-task (i.e. pure MPI mode). Our performance tests here clearly show the I/O rate is not satisfactory in this case. The other extreme case is to set the number of threads to the maximum per SMP node, i.e., each SMP node is a single MPI-task. In this case, our results show the best performance. Ultimately, the choice of threads depends on the balance between computational efficiency and I/O performance.

Another important advantage of parallel I/O strategy in CAM is to relax the memory limitations resulting from the existing sequential I/O, which uses its local memory in the buffering stage. Many applications have data configurations larger than the local memory on the I/O processor, especially for memory-bounded hardware such as the BG/L system. Figure 6 compares the parallel I/O performance of CAM FV C-resolution ($1° \times 1.25°$) run between IBM SP5 and BG/L systems. Thread=4 is used on IBM SP5. On both the SP5 and the BG/L systems, the bandwidth increases as the number of processors for the new parallel I/O algorithm is increased, while the performance saturates at 128 and 64 processors, respectively. The comparison shows that the bandwidth is much better on SP5 rather than BG/L. The sequential I/O on SP5 is also much better than the parallel I/O approach on BG/L. This may result from the optimized I/O tuning and larger memory within each node on SP5. Note that BG/L system represents a more power-efficient alternative approach that achieves its performance through higher concurrency than mainstream cluster solutions. The core of BG/L system is a SOC (system on chip) based on the low-power PowerPC 440 embedded core, which is a standard in-order CPU core presented in many low-power embedded applications. The BG/L offers well-documented improvements to the power efficiency for many scientific applications. However, some large-scale applications may not be well suited for the architecture. Large-scale CAM FV C- (or D-) resolution is an example.

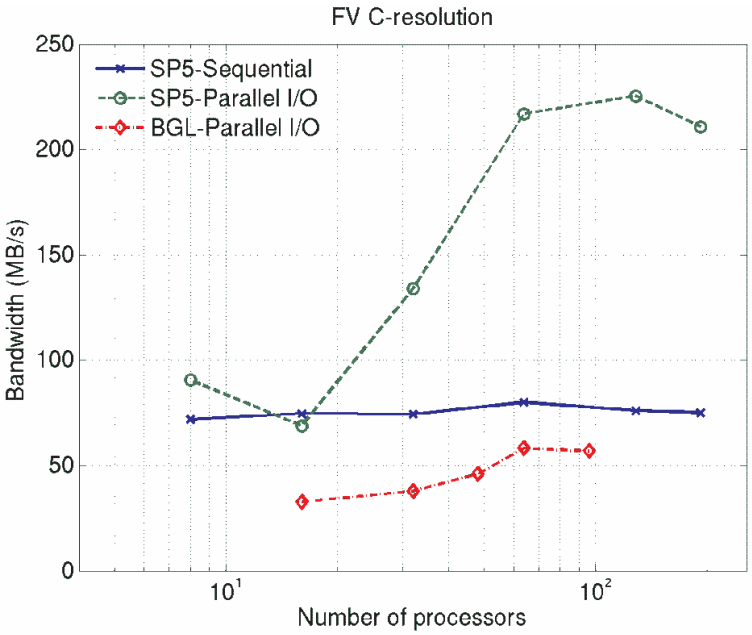

Fig. 6 History I/O bandwidth of parallel I/O for CAM 3.1 FV 1° × 1.25° C-resolution on IBM SP5 (Thread=4) and BG/L. The existing sequential I/O fails to run on BG/L.

The CAM FV C-resolution simulation causes severe I/O problems using the default sequential I/O on BG/L system because of strict memory limitation (< 512 MB). Therefore, no curve for sequential I/O on BG/L system is presented in Figure 6. Our parallel I/O strategy solves the main I/O difficulty of CAM on BG/L system. The low bandwidth on BG/L is possibly the result of the slower and unoptimized code path during the benchmark testing. We found that the behavior of PnetCDF library was unstable, and sometimes generate erroneous results on BG/L system. Further investigation indicates that this was possibly because of the MPI-IO implementation On the ANL's BG/L system, we tested the performance of both the NFS and PVFS2[7] systems. The PVFS2 is an efficient parallel file system developed by researchers at ANL and many collaborators. Both file systems lack certain unix-like characteristics (caching and locking behaviors), thus treating them as a unix file system may fail sometimes. Taking a slower and unoptimized code path suggested by the PnetCDF development team indeed helps for many cases, but does not solve the inconsistency problem fundamentally. The same problem also occurs on NCAR's BG/L system which also uses NFS-exported GFPS file system (Ghosh et al. 2006). This problem has been reported so that IBM can rebuild their MPI-IO with support for NFS.

Table 2
Standard data characteristics of CAM FV D-resolution simulation history output.

Variables	Number of variables	Size
1-D variables	2	8
2-D variables	60 + 1	$576 \times 361 + 576 * 49$
3-D variables	34	$576 \times 361 \times 26$

Table 2 documents the standard data characteristics of CAM FV D-resolution simulation history output. At each real-time output, the history file includes two 1-D variables (scalar), sixty-one 2-D variables and thirty-four 3-D variables. The size of each variable is listed in Table 2. These data characteristics are critical for the outstanding behavior of PnetCDF, consistent with early evaluation of PnetCDF in ROMS. The PnetCDF outperforms the serial netCDF when the written record is large enough to overcome the overhead from MPI-IO (Yang et al. 2004a).

4.2 Parallel Scaling Analysis

So far we have reported the total I/O bandwidth (i.e. total output time). The total I/O time includes the file "open/

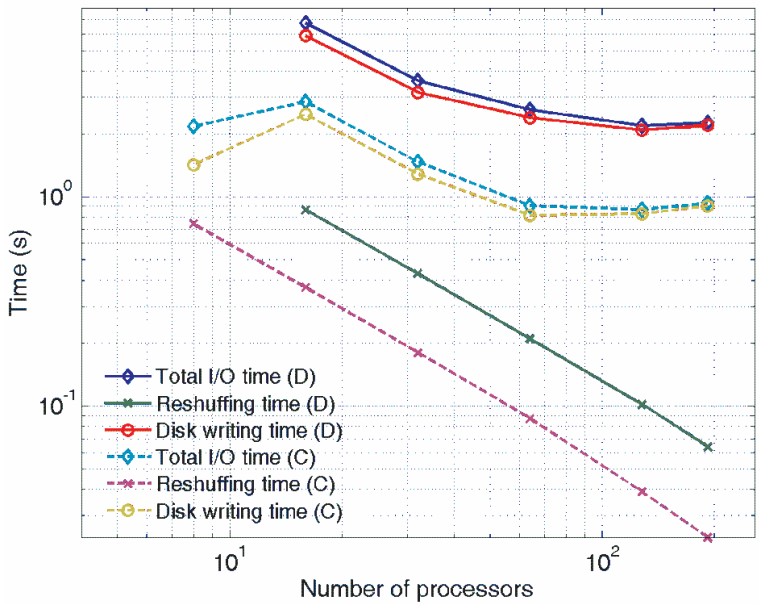

Fig. 7 Total I/O, data reshuffle and disk writing time for the parallel I/O processing of CAM on IBM SP5. The time is averaged using 5 runs. "D" represents FV D-resolution while "C" represents FV C-resolution (thread=2). Note that the horizontal resolution doubles as the alphabetical order increases. The disk writing time also indicates the total I/O time without the remapping process (collective I/O).

close" time, reshuffle time, and disk writing time. Here we further break down the total time spent in each process. Note that the file "open/close" time is very small compared with the dominant reshuffle/disk writing time, and can be neglected. This analysis represents more realistic timing in a production environment. Figure 7 shows total I/O time, data reshuffle, and disk writing time for the total I/O processing on IBM SP5, respectively. The disk writing time actually indicates the overall PnetCDF performance without the remapping algorithm in CAM. Since this remapping procedure is essentially required in CAM to convert the data into correct output index order, the total output time is mainly dominated by the optimized collective I/O in PnetCDF when the number of processors is large. "D" represents FV D-resolution, while "C" represents FV C-resolution (see Table 1 for the resolution and domain size). Note that the horizontal resolution doubles as the alphabetical order increases. In the realistic applications, roughly 15% of the total I/O time is spent on the remapping procedure for FV D-resolution, and the percentage further decreases with increasing number of processors. Figure 7 shows that the significant saving mainly results from the improvement of this process. Note that log-scale is used here to accommodate a large range of time and number of processors. In FV C-resolution run (Table 1), the disk writing time has a peak at 16 processors. This peak is expected since IBM SP5 has 8 processors per node. For the run with 8 processors, all communications are within a single SMP node. No data interconnection is transferred between nodes. For the 16 processors run, the disk writing performance degrades due to the transfer between nodes.

It is clear that the remapping (reshuffle) process time (crosses in Figure 7) scales well with increasing processors using the current parallel I/O algorithm. The straight lines indicate efficient parallelization. The high efficiency results from the significant reduction of local array size with increasing processors during the local reshuffle time. However, the behavior is slightly worse for the disk writing process, which includes the all-to-all communication overhead and I/O interface, consistent with that reported in Ding (2001). The total I/O time is mainly dominated by the disk writing time when the total number of processors is large enough. The saturation and slight degradation of the performance with a large number of processors in the parallel I/O approach comes from disk writing overhead. The remapping process time is based on the software program while the disk writing time may include many other factors, such as hardware architecture, file system, communication overhead etc. that could not be controlled by programmers.

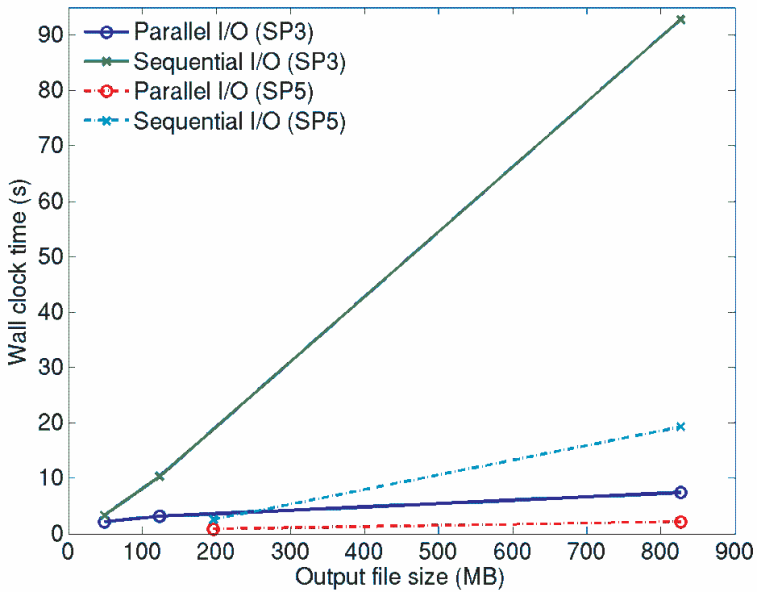

Fig. 8 Performance comparison of sequential and parallel I/O in CAM with different output file sizes (i.e. different resolution). Here IBM SP3 is at NERSC and IBM SP5 is at NCAR. 128 processors are used.

The new parallel I/O algorithm significantly benefits the realistic application. Even higher resolution runs (higher than FV D-resolution) are also proposed for CAM in the next five years (Wehner et al. 2007). The impacts of larger data sizes and higher output frequency are significant and will be further discussed in the next section.

4.3 Scalability and Impacts

In existing CAM, the sequential I/O processor (processor 0) gathers distributed data, transposes the global array, and writes to a file. Compared to this method, the new approach can speed up I/O by a factor of 14 for FV D-resolution with respect to the single processor I/O on IBM SP3. The speed-up relies on the 3-D domain size of variables. Figure 8 compares the performance of sequential (represented by crosses) and parallel I/O (represented by circles) of CAM with different output file sizes on two different platforms. The parallel I/O cases use 128 processors with thread=4. Different output file sizes also correspond to different resolutions and dynamic cores (Table 1). The output file size is obtained by taking the average of each standard output file based on eleven times output. The notations in Figure 8 for SP3 represent FV B-resolution, EUL T85 resolution, and FV D-resolution (from small to large sizes; see Table 1). The notations on Figure 8 for SP5 represent FV C- and D-resolutions (from small to large sizes), respectively. The results show the significant advantage of using parallel I/O for larger data size. As the output file size (domain size) increases, the wall clock time of model sequential I/O increases linearly. The wall clock time is proportional to the domain size; however, the wall clock time becomes almost constant for the parallel I/O runs. Using parallel I/O alleviates the output processing of large data set through multiple staging processors.

A sequential I/O bottleneck will emerge soon in CAM and CCSM related models as the history output becomes more frequent. Giving an example of estimated time for FV D-resolution on IBM SP5, the impacts of parallel I/O are given in Table 3 for 256 processors (64 MPI-tasks). IBM SP5 represents the latest development of computer architecture. Typical realistic simulations, used to assess the long term trend of global warming, need to spin up for hundreds of years (Collins et al. 2006). We present the estimated time of weekly, daily and 3-hourly interval output for a century (a hundred years) run in Table 3. Only a small fraction of the total running time (1%) is used in the I/O with weekly data output for a typical 100-years simulation of FV D-resolution on IBM SP5. The sequential I/O causes no bottleneck problem in the overall simulation. However, comparing the parallel I/O strategy with the existing sequential I/O for daily output, the estimated time for a typical 100-year simulation shows that the I/O time can be reduced from more than 8 days (wall clock) to less than 1 day. When the output interval

Table 3
Estimated time for a century-long simulation of FV D-resolution on IBM SP5 (assuming weekly, daily and 3-hourly standard output).

128 Processors	CAM 3.1 (sequential I/O)		CAM 3.1 (parallel I/O)	
	Total (days)	Sequential I/O (days)	Total (days)	Parallel I/O (days)
Weekly output	126.5	1.2	125.4	0.1
Daily output	133.7	8.4	126.1	0.8
3-Hourly output	159.0	33.7	128.8	3.5

is further reduced to 3 hours, the I/O time will take more than a month for the data output (about 21% total running time) while it takes only 3 days using parallel I/O approach.

5 Concluding Remarks

In this paper, we have described an efficient approach to solving the I/O bottleneck problem and analyzed its performance and timing in large-scale CAM. This generic approach uses the remapping to z-decomposition and PnetCDF for the file system, and has been implemented in the CAM climate model. The performance improvement is systematically studied on several different platforms. We also show the scalability and significant impacts for the next generation CAM model. This approach can be easily extended to other large-scale models using sequential I/O to handle large data sets. The speedup is scaled with the domain size because of the constraint of local memory and architecture. The parallel I/O significantly improves the I/O performance because of better local remapping processes. ON IBM SP, the new parallel I/O speeds up the history output by a factor of 9–14. For the practical impact, the estimated I/O time for a century simulation of FV core, D-resolution on IBM SP5 shows that the I/O time can be reduced from more than 8 days (wall clock) to less than 1 day for daily output. This parallel I/O also facilitates the porting of CAM to BG/L system, on which the existing sequential I/O fails to work because of memory usage limitations.

The scaling analysis shows that we can significantly improve the reshuffle time using an efficient parallel strategy in CAM. However, the improvement of disk writing time, which relies on the hardware architecture and design, is not remarkable. The optimal I/O performance also depends on the domain size and domain decomposition. It was found that ~64 MPI-tasks is the best choice for CAM FV D-resolution, and this suggests the use of dedicated parallel nodes for I/O. Future work will include the porting to more platforms (such as Linux clusters and Cray XT4), and completed software development for the optimal I/O in CCSM.

Acknowledgments

This work was supported by a DOE SciDAC climate project and partially by the NERSC Program. The first author also acknowledges the support from the National Science Council, Taiwan. We thank the NCAR CAM team, especially Mariana Vertenstein for collaboration. We thank the PnetCDF team, especially Jian Li of NWU and Rob Ross of ANL, for answering a large number of questions and the quick fixing of many bugs. We thank Pat Worley of ORNL for CAM chunking related issues and Art Mirin of LLNL for CAM FV core related issues. We thank John Tannahill of LLNL for PnetCDF performance discussions. Woo-sun Yang implemented the ZioLib from which we adapted the remapping codes. The computational support at NERSC, NCAR and ANL is acknowledged.

Author Biographies

Yu-heng Tseng is an assistant professor at the Department of Atmospheric Sciences, National Taiwan University, Taiwan. He received his Ph.D. on the development of immersed boundary method for environmental flows from Stanford University. After his Ph.D., he worked as a post-doctoral fellow at the Department of Mechanical Engineering at Johns Hopkins University on the large-eddy simulation of atmospheric boundary layer. He spent two years at the Computational Research Division, Lawrence Berkeley National Laboratory on the parallel I/O implementation for a community atmosphere model. His primary expertise is high performance computing for environmental flows and computational fluid dynamics. His recent research includes high-order numerical algorithm development and high-performance coastal and basin-scale ocean/atmospheric modeling, among others.

Chris Ding is a professor in the Department of Computer Science and Engineering, University of Texas at Arlington. He was a staff computer scientist at the Lawrence Berkeley National Laboratory from 1996. He earned a Ph.D. from Columbia University on building a parallel processor using Intel 80286s and commodity FPUs (Science, cover

page, August 1988). After his Ph.D., he worked for 6 years at the California Institute of Technology, on Caltech hypercubes with applications ranging from materials science (1990, Nature 344: 485) to computational biology. He spent 3 years at the Jet Propulsion Laboratory on atmospheric data assimilation (SIAM News, front page, October 1996) and parallel computing algorithms. His recent research includes algorithmic R&D for climate models, bioinformatics, and data mining. Present address: Department of Computer Science and Engineering, University of Texas at Arlington, Nedderman Hall, 416 Yates St., Arlington.

Notes

1 http://www.ccsm.ucar.edu/models/ccsm3.0/ccsm/

2 http://hpcrd.lbl.gov/~yhtseng/research/parallel_io

3 Unidata, netCDF, http://my.unidata.ucar.edu/content/software/netcdf/index.html

4 http://crd.lbl.gov/~cding/acpi/ZioLib/

5 http://flash.uchicago.edu

6 http://www.nersc.gov/nusers/systems/SP/

7 http://www.pvfs.org

References

Bokhari, S. H. (1991). *Complete exchange on the Intel iPSC-860 hypercube*, Technical Report 91-4, ICASE.

Carns, P. H., Ligon, W. B. III, Ross, R. B., and Thakur, R. (2000). PVFS: A parallel file system for Linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GA, October 2000, pp. 317–327.

Collins, W. D., Rasch, P. J., Boville, B. A., Hack, J. J., McCaa, J. R., Williamson, D. L., Briegleb, B. P. et al. (2006). The formation and atmospheric simulation of the Community Atmosphere Model: CAM3, *J. Climate* **19**: 2144–2161.

Ding, C. (2001). An optimal index reshuffle algorithm for multidimensional arrays and its applications for parallel architectures, *IEEE Trans. Parallel Distrib. Syst.* **12**: 306–315.

Ding, C. and He, Y. (1999). Data organization and I/O in a parallel ocean circulation Model. In *Proceedings of SC'1999*, Nov.

Edelman, A., Heller, S., and Johnsson, S. L. (1994). Index transformation algorithms in a linear algebra framework, *IEEE Trans. Parallel Distrib. Syst.* **5**: 1302–1309.

Foster, I. T. and Worley, P. H. (1997). Parallel algorithms for the spectral transform method, *SIAM J. Sci. Stat. Comput.* **18**: 806–837.

Fox, G. C., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D. (1988). *Solving problems on concurrent multiprocessors*, Prentice-Hall.

Fraser, D. (1976). Array permutation by index-digit permutation, *J. Assoc. Comp. Mach.* **22**: 298–308.

Ghosh, S., Loft, R., Tseng, Y. H., Ding, C., and Wehner, M. (2006). Computational and I/O performance study of FV CAM in Bluegene/L and Pwr5 system. In *ScicomP12*, July 18–21, Boulder, Colorado.

Gropp, W., Lusk, E., Doss, N., and Skjellum, A. (1996). A high performance, portable implementation of the MPI Message-Passing Interface standard, *Parallel Comput.* **22**: 789–828.

Gropp, W., Lusk, E., and Thakur, R. (1999). *Using MPI-2: Advanced features of the Message Passing Interface*, Cambridge, MA: MIT Press.

Hack, J. J., Rosinski, J. M., Williamson, D. L., Boville, B. A., and Truesdale, J. E. (1995). Computational design of the NCAR community climate model, *Parallel Comput.* **21:** 1545–1569.

Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994). *Introduction to Parallel Computing*, Redwood City: The Benjamin/Cummings Publishing Company, Inc.

Li, J., Liao, W. K., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A. et al. (2003). Parallel netCDF: a high-performance scientific I/O interface. In *Proceedings of SC'2003.*

Prost, J. P., Treumann, R., Hedges, R., Jia, B., and Koniges, A. (2001). MPI-IO/GPFS, an optimized implementation of MPI-IO on top of GPFS. In *Proceedings of SC'2001.*

Rew, R., Davis, G., Emmerson, S., and Davies, H. (1997). *NetCDF user's guide for C*, Unidata Program Center. http://www.unidata.ucar.edu/packages/netcdf/guidec/

Wehner, M., Oliker, L., and Shalf, J. (2007). Towards ultra-high resolution models of climate and weather (*submitted to International Journal of High Performance Computing Applications*).

Yang, M., Folk, M., and McGrath, R. E. (2004a). *Investigation of parallel netCDF with ROMS,* May 4, http://www.hdfgroup.uiuc.edu/HDF5/projects/archive/WRF-ROMS/

Yang, M., McGrath, R. E., and Folk, M. (2004b) *Performance study of parallel netCDF in ROMS,* August 27, http://www.hdfgroup.uiuc.edu/HDF5/projects/archive/WRF-ROMS/

Yang, W. S. and Ding, C. H. Q. (2003). *ZioLib: a parallel I/O library*, LBNL Tech. Report, LBNL-53521.