Improvement of TOMCAT-GLOMAP File Access with User Defined MPI Datatypes

Mark Richardson
HECToR CSE Team
Numerical Algorithms Group Ltd.
Manchester, United Kingdom, M1 5AN
mark.richardson@nag.co.uk

Martyn Chipperfield
School of Earth and Environment
University of Leeds
Leeds, United Kingdom
martyn.chipperfield@env.leeds.ac.uk

Abstract

TOMCAT-GLOMAP is software that is used to model global atmospheric chemistry and aerosol processes in three dimensions. The model is written in FORTRAN and has evolved from originally running on vector machines to the current parallel code. It is widely used by the UK atmospheric science community on the Cray XE6 UK national computing resource (HECToR). HECToR uses a LUSTRE file system to support its 90112 computational cores. Recent developments in TOMCAT-GLOMAP include moving to a higher resolution atmosphere and more detailed chemistry processes making the file access more significant.

This paper describes the modification of the file access patterns that occur throughout the simulation. The analysis identified subroutines where the workload was not actually the accessing of the data but the processing of data either before writing it or after reading it. The main gains in the project have been achieved by reducing the load on MPI task zero.

The replacement methods include properly distributing the work such that local data is processed before any gather and subsequent writing. Also, after reading data, it is distributed out to other MPI tasks before it is processed.

The overheads of some of the gathering category of subroutines have been reduced. For example, reporting on altitude data at specific locations. In the case using 80 MPI tasks, for the original coding the time per iteration is 8.9s on the writer iteration compared to 0.98s of a non-writing iteration. The time per writing iteration for the new method is 0.99s. When more MPI tasks were applied the overhead for the original method became more significant (e.g. with 400 MPI tasks the time per writing iteration is 38s compared to the time of 0.42s per non-writing iteration). In this case the time per iteration for the new method is 0.47s.

These changes allow the higher resolution to be simulated more efficiently and to be more flexible within research work. Researchers can now choose to do more within an experiment without the overhead previously incurred when activating the functions that have been modified by this project.

Keywords **– TOMCAT; Atmospheric Chemistry, MPI Datatypes; File access; Global Chemical Advection;**

I. INTRODUCTION

A. *The Cray XE6 "HECToR"*

HECToR is a Cray XE6 of 2816 nodes with AMD Magny-Cours processors i.e. 32 cores per node (90112 cores).

This project was funded under the HECToR Distributed Computational Science and Engineering (CSE) Service operated by NAG Ltd. HECToR – A Research Councils UK High End Computing Service - is the UK's national supercomputing service, managed by EPSRC on behalf of the participating Research Councils. Its mission is to support capability science and engineering in UK academia. The HECToR supercomputers are managed by University of Edinburgh, HPCx Ltd and the CSE Support Service is provided by NAG Ltd. http://www.hector.ac.uk

B. *The software*

The TOMCAT software is a chemical advection transport code. It processes chemical aerosol data to provide the time evolution and distribution of chemical species around the whole earth atmosphere.

It is being used to simulate the atmospheric processes at higher resolution of T106 (1.2° x 1.2°). This is more than twice the resolution of the widely used T42 (2.8° x 2.8°) case, also used in previous DCSE work [1,2]. That effort improved the performance of TOMCAT by some restructuring of the key routines and modifying some of the functions that prepare data for MPI communication. This project is to examine the use of TOMCAT/GloMAP with the computational model set to the high resolution as it is anticipated that the file access times will become more significant due to the increased amount of data associated with the higher resolution.

C. *Case used in the investigation*

The specific computational resolution of this case is 320x160x60with only 6 tracers and 6 species. The simulation is for a single day (96) steps of simulation, this being the equivalent to 24 hours of 1st January 2005. Timing and analysis has been done for three different decompositions (80, 160 and 400 MPI tasks). The code provides some raw timing derived from calls to MPI_WTIME on task zero and these have guided the research to investigate subroutines in which the workload is highest. The Cray PAT tools have been used to identify the high work load sections of code. The facility in Cray PAT for file access tracing has been used to determine the amount of data being read-from, and written-to, various external files throughout the simulation. This has been done for several phases of the run.

TABLE I. MPI TOPOLOGY AND DOMAIN DECOMPOSITION

MPI tasks	**80**	**160**	**400**
NPROCI	5	5	5
NPROCK	16	32	80
MYLON	64	64	64
MYLAT	10	5	2
NIV	60	6	60
Number of Grid boxes	39040	19520	7808

The NPROCI remains the same due to numerical constraints near the north and south poles.

II. ANALYSIS OF THE EXISTING SOFTWARE.

Work on specific functions listed here is detailed in the later sections. The topics addressed are: some code structure review, two significant report generators and two functions that facilitate initialization and restart.

A. *Task 1 profile of the code.*

The user defined timing function provides a reasonable guide to what is happening per iteration. The code structure is as shown in Fig. 1 (reduced detail for illustrative purposes).

```
Program TOMCAT
iniexp
inichi
loop ncyclt (set in fort.94)
  inicycl
  loop nitert (set in fort.95)
    initer

!!!   inner iteration work

    finiter
  end loop nitert
  fincycl
end loop ncyclt
finexp
```

Fig. 1. Example of a figure caption.

Table II shows a breakdown of average times per iteration for various stages of the one day simulation. These have been done for three MPI decompositions 80, 160 and 400 MPI tasks. The PE400 configuration has the minimum possible number of latitudes on a domain i.e. 2 and this is very inefficient for reasons related to; the higher number of communications; the FFT method and any multi-threading which usually occurs around loops of latitude. The row labelled “step N” in table II provides a clear indication of the pure scalability as that step is where there is no significant file access (other than stdout or stderr). The values are broadly in-line with earlier work [3] where Open MP directives were added to the TOMCAT MPI version. This project does not deal with threading and most of the investigations are done with OMP_NUM_THREADS set to unity.

TABLE II. VARIATION OF TIME PER ITERATION DURING 24 HOUR SIMUALTION

MPI tasks	**80**	**160**	**400**
Iteration	***(seconds)***	***(seconds)***	***(seconds)***
Initial	332	327	323
N	1.06	0.6	0.47
"9, 17"	3.37	1.82	0.94
"24"	5.7	3.2	2.19
“49”	15.21	6.57	7.95
“96”	13.59	5.64	7.02

There is a distinct pattern where some iterations take longer than others. These are characterized by whether the iteration has data access or some “house-keeping” of the simulation; every 2 hours is the 1st, 9th and 17th step of each cycle; very 6 hours is the 24th step (because the work is done at the end of a cycle). The 12th Hour is on step 1 of the third cycle (i.e. step 49). Similarly the final work is done within the 96th step because this case is simulating only 1st January 2005.

Each of the six categories identified in Table II were investigated using the Cray PAT methods described in the next section, more detail can be found in [3].

Cray Performance Analysis Toolset

The Cray PAT has been used to inspect the runtime behaviour of the software. As the focus of the project is file access the test case is limited to the simulation of 24 hours. Sampling experiments are used as an initial pass to confirm the routines which will be investigated further. Trace experiments are used with tracing focussed on those subroutines plus any other routine that showed up in the sampling experiment. Additional experiments were done using the “io” group function which reports the volume and rate of data being read from and written to external files. The “pat_record” feature was used to enhance the instrumentation. This function allows the user to choose where to activate the sampling or trace information. Using the information from Table II the recording was activated in each of the zones

TABLE III. A CRAY PAT SAMPLE EXPERIMENT

Samples	Num of samples	Subroutine
100.0%	124.1	Total
84.3%	104.6	USER
42.6%	52.9	consom_
17.4%	21.6	advy2_
8.4%	10.5	advz2_
8.0%	10.0	advx2_
1.8%	2.2	MAIN_
1.3%	1.7	chimie_
13.3%	16.5	MPI
5.9%	7.3	MPI_SENDRECV
2.8%	3.5	MPI_BARRIER
1.7%	2.1	mpi_recv
1.5%	1.9	MPI_SSEND
1.0%	1.3	mpi_bcast
2.5%	3.1	ETC

Table III reveals that in the non-writing advection iteration there are five subroutines of significance. The Cray PAT analysis indicates that there is a lot of time spent inside CONSOM function on a standard advection step.

B. High Workload Functions

1) Calculation of convection effects

CONSOM, is a function for mixing tracers by convection. Historically this subroutine has had access to files and this goes some way to explain the code structure. Usually the developers organise the loop structures with the outer loop for the final index in the array dimensions (the longest stride through memory). For this subroutine the outer loop is for the latitudes (the second index). The rationale is that the convection terms per latitude could be stored on disk after they had been calculated. The calculation still adheres to that structure working in planes of altitude and longitude for all tracers before moving on to the next latitude. The innermost loops do cycle through the first index (of contiguous data) for longitude.

2) Adjusting flux terms

The subroutine CALFLU is used on each CYCLE to calculate flux components. In this case that is every 24 iterations (6 hourly). This is because every six hours a fresh set of spectral coefficients are read from external reference file. It is a long subroutine and in the first 80% there is much data movement due to the underlying FFT calculations. In the latter 20% there is some communication at patch boundaries for the north-to-south transfer of velocities on grid-box interfaces. The inherent nature is for a cascade through the latitudes with the southern latitudes cannot receive data from their south until they have sent their data to the north. For a configuration with many MPI tasks in the latitude direction (NPROCK) this could become a bottleneck.

C. Functions With File Interaction

There are several functions within TOMCAT for reading and writing data. For the core program these are INIEXP and FINCYCL there is also an option to write results files at the end of each iteration (FINITER). Additional functions are available that provide logging specific analyses that are common operations for the researchers of atmospheric chemistry. There is still ongoing work to update the NetCDF sections to use parallel NetCDF.

The four functions discussed here were considered to be the ones that would provide greatest improvement in the performance. They are GBSTAT, SORZM, PPREAD and PPWRIT.

1) Reporting altitude profiles for ground based station locations.

The GBSTAT function is used to write out data along a specific column of atmosphere. The locations of ground based stations are known either using the pre-set built-in locations or supplying a set of locations in an external file. The stations are at locations over the Earth’s surface (including ship borne in some cases) and thus might be non e on some domains and multiple occurrences on other domains. The stations might not coincide with the computational grid so interpolation has to happen. Many fields are accessed and written to a file for later assessment. Fig. 2 shows an overview of the process including that for SORZM described in the next section as they are almost identical up to the serial section on task zero.

2) Evaluating zonal mean for a field.

The SORZM function evaluates the average value of a parameter along a line of latitude. This is known as the zonal mean. It is calculated individually for several field variables at specific simulation times. The result is stored for each one on MPI task zero as a plane of longitude by altitude.

There is a phase of collecting all the data for a specific field to process zero and then evaluating the zonal mean. Each field is dealt with in turn requiring a large volume of data to be transferred. The inherent serialization of the calculation causes all other nodes to wait idle while task zero deals with the whole of the field.

The multiple copying of data is very inefficient and partly serializes the algorithm, particularly in the collection onto task zero and subsequent search and interpolate that is done only on task zero. The same process applies to GBSTAT and the zonal mean calculation as the summation and averaging for the whole atmosphere volume occurs on one processor.

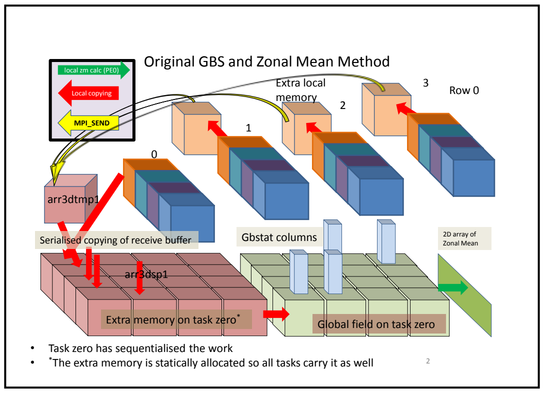
Original GBS and Zonal Mean Method
local zm calc (PE0)
Local copying
MPI_SEND
Extra local memory
Row 0
0
1
2
3
arr3dtmp1
Serialised copying of receive buffer
Gbstat columns
2D array of Zonal Mean
arr3dsp1
Extra memory on task zero*
Global field on task zero
Task zero has sequentialised the work
*The extra memory is statically allocated so all tasks carry it as well
2

Fig. 2. Existing GBSTAT and SORZM method, collection from row zero shown for one field. Process is repeated for as many fields of interest to th researcher.

3) Reading the initialization file or reading the restart file.

The experiment is initialised with a long subroutine that does a lot of processing with the data read from various sources. One of the sources is the initialisation of a restart file. A header block is read and broadcast to the other MPI tasks and then a subroutine, PPREAD, is used to read a plane at a time, several fields from a Fortran unformatted file. It reads each plane into a vector that is then reshaped into a two-dimensional plane and uses another function to distribute parts of the plane to the other MPI tasks. The function is DISTF. It is called with values for the extent of the distributed plane. There is a complex stage of packing the buffer into vectors specific to each MPI task and then looping over the processors to send the data to each one.

4) Writing the checkpoint or restart file.

The PPWRIT function is very similar to the PPREAD function with the sequence of steps done in a reverse manner. A function COLLF is used to send a sub-plane of a field to task zero from the other MPI processes. It is then written to a Fortran unformatted file and will be used for any follow-on simulation. There are other places where this function is called as part of a restart check-pointing. Another version exists that helps with the NetCDF file writing.

III. Data Types Developed For This Work

A. The need for bespoke data types

There are several data types intrinsic to MPI such as MPI_INTEGER, MPI_DOUBLE_PRECISION and they are widely used in many MPI applications. In the TOMCAT software the use of MPI is limited to these intrinsic data types and a small subset of functions. The following sub-sections describe how the new data types are defined and used.

When using a different number of MPI tasks, the topology remains the same NPROCI by NPROCK two-dimensional grid of processes. Therefore, different resolutions of the model would be able to use a set library of data types that represent the structure of data that has to be read form file, communicated or written to file. Many of the communications involve accessing three-dimensional field data through 2 dimensional planes. The resolution of those can be different to the computational space for example reading true 1x1 data (360x180 points) into the T106 simulation (320x160 points).

Three data types have been created for use with the MPI Gather and Scatter functions. They are needed by the replacement subroutines for PPREAD and PPWRIT. The data types are CLMN_NH_T, CLMN_WH_T and BLK_NH_T. The following diagrams are used to explain their relationships with the regular data structures used by TOMCAT.

The "column-no-halo" data type identifies the contiguous elements that form a row of a sub-array of the global data that is stored on MPI task zero. The MPI_Contiguous function is used to create a vector of length "MYLON" which is the first dimension of the two-dimensional array. Several of these columns are collected together to create a new data type: "block-no-halo". This identifies the whole of the sub-array that will be scattered or gathered in the new functions.

The "column-with-halo" type is a form of CLMN_NH_T that has been resized to account for the halo storage. The definition of the sub-array on the distributed domain uses negative indices so the invocation of the gather or scatter function uses a reference to the first "internal" element.

The actual invocation of the Scatter is:

```
CALL MPI_Scatterv (                          &
          & PFG, NSND, DISP, BLK_NH_T, &
          & PFL(1,1), NRCV, LTYPE,          &
          & ROOT, UCOMM, MPERR)
```

The most significant element is the DISP array. It provides the start location in the send buffer of the data type being distributed. In this example a "block-no-halo" type is being distributed to every MPI task. NSND is an array of integers indicating that one block has to be sent to each process. NRCV is a single integer for each receiver to understand that it will receive MYLAT elements of LTYPE, in this case CLMN_NH_T as it is passed through the subroutine interface. Fig. 4 shows a vector of CLMN_NH_T as it strides by NPROCIxMYLON through the global data.

Although conceptually a developer may think of the array as two-dimensional shown in Fig. 4, it is actually a line of information in memory similar to the diagrams in Fig. 5.

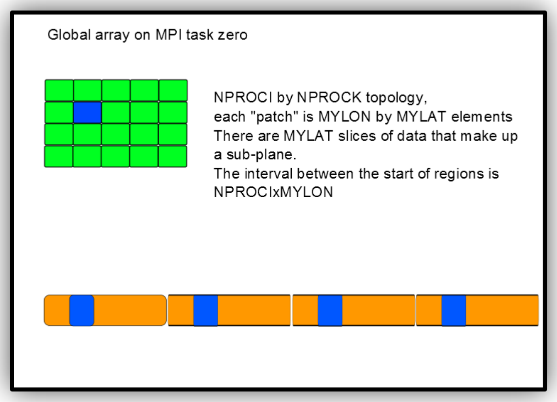
Global array on MPI task zero
NPROCI by NPROCK topology,
each "patch" is MYLON by MYLAT elements
There are MYLAT slices of data that make up
a sub-plane.
The interval between the start of regions is
NPROCIxMYLON

Fig. 3. The two-dimensional concept is in practice a vector. The sub-array is stored at intervals with a stride of NPROCIxMYLON elements. The case is shown where MYLAT is 4.

The large block of rectangles represents the global plane of data. The darker shaded rectangles indicate the data that is distributed on both cases. The lighter shaded area surrounding the second block is the halo storage of the local field.

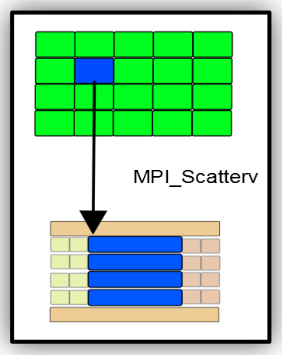

Fig. 4. Scattering the sub-domain planes (without and with halo storage).

In some circumstances data arriving from a scatter function will have to be placed in the correct location of an array that includes storage for halo data. The data being read from a file has no halo information and only the interior information is distributed to the other MPI tasks. A data type is constructed from the foundation CLMN_NH_T with the stride.

Starting at the top there is a representation of CLMN_NH_T and the extended CLMN_WH_T superimposed. Below that is a single entity representation of CLMN_WH_T and its equivalent storage as a local array. This is shown as four contiguous blocks above the true data layout. It is the darker shaded blocks that are collected during the MPI_Gather into the global sized data structure, shown at the bottom f the figure.

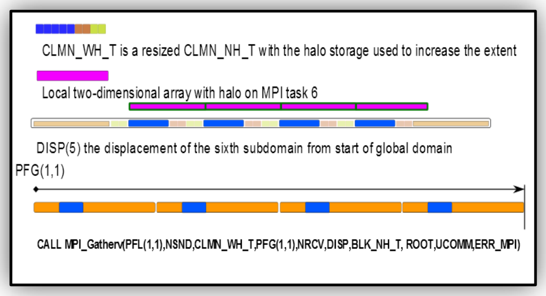
CLMN_WH_T is a resized CLMN_NH_T with the halo storage used to increase the extent
Local two-dimensional array with halo on MPI task 6
DISP(5) the displacement of the sixth subdomain from start of global domain
PFG(1,1)
CALL MPI_Gatherv(PFL(1,1),NSND,CLMN_WH_T,PFG(1,1),NRCV,DISP,BLK_NH_T, ROOT,UCOMM,ERR_MPI)

Fig. 5. the storage structure for each type for a local data structure of the sub-array.

The invocation of the Gather is very similar however the data types are reversed and the counts for send and receive are also switched.

To activate the data types, a call has to be made to:

CALL INIT_PLANE_DT (MYLON, MYLAT, NPROCI, IHALO)

This function supplies the base size of the simulation and halo in the West-East direction (I loops). Inside this function all the new types are defined and committed to (registered with) MPI.

IV. Detail Of Specific Functions

A. *Modification Of The CONSOM Function.*

The general process involves several cross-calculations between altitude layers and introduces an additional index to the already three-dimension local data structures.

```
ftn -Mneginfo -Minfo=mp -fast -Minline -Minline=reshape -
Mextend -Mbyteswapio -mp=nonuma –r8 -c orig_consom.f
consom:
    49, Invariant assignments hoisted out of loop
        Loop not vectorized/parallelized: too deeply nested
    54, Parallel region activated
    57, Parallel loop activated with static block schedule
        Loop not vectorized/parallelized: too deeply nested
    62, Generated 4 alternate versions of the loop
        Generated vector sse code for the loop
    76, Loop interchange produces reordered loop nest:
77,76,78
    78, Generated 3 alternate versions of the loop
        Generated vector sse code for the loop
    93, Generated 4 alternate versions of the loop
        Generated vector sse code for the loop
   110, Memory zero idiom, loop replaced by call to
__c_mzero8
   115, Loop interchange produces reordered loop nest:
116,115,117
   117, Generated 3 alternate versions of the loop
        Generated vector sse code for the loop
   124, Memory copy idiom, loop replaced by call to
__c_mcopy8
   131, Parallel region terminated
```

Fig. 6. Excerpt of a build log, specific to CONSOM

The loop structure is interrupted with a conditional test (unnecessary) but it seems the compiler is very helpful in identifying the structure. Fig. 6 shows an extract from the build process with options for the PGI compiler (including –Mneginfo) turned on.

The information generated by the compiler indicates that there are a couple of loop structures that were reordered. However, manually editing those loops and recompiling provided a similar message about the new loops.

Unfortunately very little improvement could be made for CONSOM without using hardware specific cache blocking algorithms.

B. *Modification Of CALFLU Subroutine.*

The section of this subroutine where communications appear inefficient was revised. Replacing this feature with an MPI_SENDRCV combined function allows the MPI system to determine the sequence of communications. Unfortunately this feature was not a significant percentage of the time for the subroutine and made very little difference to the timing.

C. *Function To Report Altitude Profiles, GBSTAT*

The GBSTAT function has been briefly described in the introduction. The modification of this function has been mainly a review of the algorithm, moving the workload from the MPI task zero out to the individual MPI tasks.

The new method lets each MPI task that contains one or more GBS to create the column needed for the results and then send that to the root process for writing in the traditional manner. The previous method of copying the full fields back to task zero has been replaced by a halo exchange just before the interpolation.

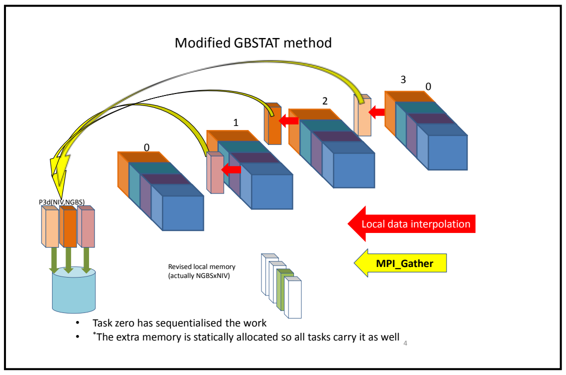
Modified GBSTAT method
3
0
2
1
0
P3d(NIV,NGBS)
Local data interpolation
MPI_Gather
Revised local memory (actually NGBSxNIV)
Task zero has sequentialised the work
*The extra memory is statically allocated so all tasks carry it as well
4

Fig. 7. The revised GBSTAT interpolation

This halo exchange is a rudimentary approach to ensure that any calculation for a GBS near the grid-box boundary will be up-to-date. A more sophisticated method was considered but the additional coding and testing would outweigh any benefit.

D. *Improvement To Reporting Zonal Means.*

A similar algorithm for collecting field data onto one process was used to calculate the zonal mean. The principle is to collect all the data on to the root MPI task and then do the work needed to process and store the data to a file. This undermines the nature of the parallel decomposition and effectively serializes the job.

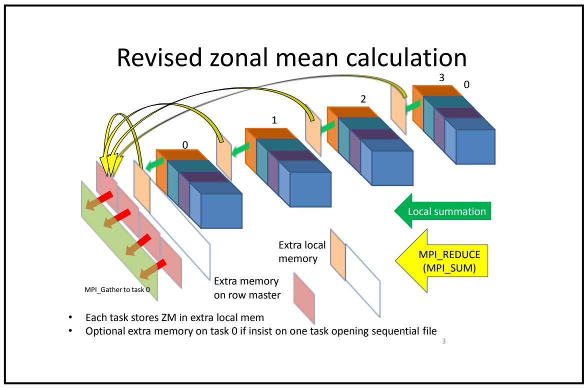
Revised zonal mean calculation
3
0
2
1
0
Local summation
MPI_REDUCE (MPI_SUM)
Extra local memory
Extra memory on row master
MPI_Gather to task 0
Each task stores ZM in extra local mem
Optional extra memory on task 0 if insist on one task opening sequential file
3

Fig. 8. The revised SORZM calculation.

The new method lets each MPI task evaluate the sum of longitudinal values into an auxiliary plane (it can be considered the west-most boundary of the domain). The sum of the auxiliary planes is evaluated by an MPI reduction in the MPI topology row to the first MPI task on that row (Fig. 8). The penultimate step is to use MPI_Gather to collect the partial arrays onto MPI task zero before finally writing in the established serial manner onto an unformatted Fortran file.

The implementation of this method includes some reuse of two-dimensional arrays; one is sized to match the west-most plane of the local domain and one is the same size as the final meridian plane used to store the zonal mean that is required by the researcher.

E. *File Reader PPREAD Is Replaced By PPRD*

The PPRD function is a replacement for both PPREAD and DISTF. It raises the DISTF function from a callee to being in-line, but then the content of DISTF is discarded in favour of the MPI_Scatterv() function. The previous requirement for a conditional flag and test has been removed as the choice of "with-halo" or "no-halo" is set by the data type of the communication. The extensive user-defined packing and unpacking is no longer required.

A version of PPRD that confines the reading to "row-master" MPI tasks rather than MPI task zero was trialled. However, it was discovered that the data file records in the INI and RST files did not correlate exactly with the latitudes. A significant amount of work will be required to convert existing restart and initialisation files so that the access by record can be used.

F. File Writer PPWRIT Is Replaced By PPWR

This function is like PPRD and accesses an unformatted Fortran file. The current algorithm follows the same theme of collecting a plane of data onto MPI task zero and then using a reshaped vector to send the whole plane to a file. The replacement routine directly calls MPI_Gatherv using the bespoke MPI data types in the same manner as PPRD:

```
CALL MPI_Gatherv (                              &
              & PFL(1,1), NSND, LTYPE, &
              & PFG, NRCV,DISP,BLK_NH_T,  &
              & ROOT, UCOMM, MPERR)
```

The integer: LTYPE is supplied in the argument list and acts as a conditional to indicate if the field has storage for haloes or not. The calling interface has changed slightly as now the extent of the plane is supplied as NIMN, NIMX, NKMN and NKMX.

V. Results

The immediate gains demonstrated in GBSTAT and SORZM are clear from the timing reports (in the four tables 2, 3, 4 and 5). The new methods become imperceptible compared to the time for an iteration. The similar change to the zonal-mean reporting again frees up the researcher to collect more results from a simulation.

TABLE IV. Effect of GBSTAT for configuration with 80 MPI tasks

Normal run	**Standard GBSTAT**	**Modified GBSTAT**	
Time (sec.)	**Time (sec.)**	**Time (sec.)**	**iteration**
384.580	300.451	307.404	1
0.531	9.333	0.981	2
0.531	0.983	0.979	3
0.527	8.996	0.985	4
0.525	0.978	0.986	5
0.528	8.789	0.980	6
0.528	0.979	0.978	7
0.528	8.995	0.983	8
1.496	3.362	3.371	9

The GBSTAT function is called on alternate iterations to give a clear indication of its effect. With the greater number of MPI tasks there is an increase in the amount of communication. This is shown in Table V with the effect of activating GBSTAT is as much as 70x the "normal" advection time step. There does appear to be some penalty with the revised method but it is still a great saving over the previous version. There is a possibility of imbalance if there are many station located in one sub-domain.

TABLE V. Effect of GBSTAT for configuration with 400 MPI tasks

Normal run	**Standard GBSTAT**	**Modified GBSTAT**	
Time (sec.)	***Time (sec.)***	***Time (sec.)***	***iteration***
375.762	303.692	298.904	1
0.343	39.418	0.788	2
0.333	0.331	0.424	3
0.322	39.366	0.474	4
0.330	0.401	0.411	5
0.327	39.011	0.459	6
0.339	0.391	0.428	7
0.335	39.351	0.540	8
0.656	0.865	0.889	9

The SORZM function has been called every iteration. This is to artificially amplify the effect and demonstrate the improvement of the modified version.

TABLE VI. Effect of SORZM for configuration with 80 MPI tasks

Normal run	**Standard SORZM**	**Modified SORZM**	
Time (sec.)	***Time (sec.)***	***Time (sec.)***	***iteration***
384.580	377.155	385.091	1
0.531	4.643	0.545	2
0.531	4.649	0.545	3
0.527	4.659	0.541	4
0.525	4.659	0.546	5
0.528	4.651	0.544	6
0.528	4.656	0.543	7
0.528	4.662	0.550	8
1.496	5.602	1.509	9
0.531	4.618	0.550	10

For 400 MPI tasks configuration the effect is an order of magnitude greater. But again the modified version is showing as hardly noticeable,

TABLE VII. Effect of SORZM for configuration with 400 MPI tasks

Normal run	**Standard SORZM**	**Modified SORZM**	
Time (sec.)	**Time (sec.)**	**Time (sec.)**	**iteration**
375.762	415.400	396.894	1
0.343	41.554	0.351	2
0.333	41.582	0.327	3
0.322	41.728	0.346	4
0.330	41.193	0.376	5
0.327	41.762	0.331	6
0.339	41.557	0.335	7
0.335	41.600	0.343	8
0.656	41.774	0.658	9
0.335	41.738	0.332	10

The overall results of the work with collectives and data-types have not yet been fully realized. Two lines of research are currently benefitting from the changes to GBSTAT. One of those is research using a high number of ground-based stations; the research can now be more extensive in coverage and request more frequent reporting. The second is a new feature of reporting satellite data as it crosses the terminator; the location of a satellite varies in longitude and latitude at that time. With the new method an MPI task need only know that a column of data is need for a number of longitudes, in the case of ground based stations that can be set at the beginning of the run whereas for the satellite data that has to be updated at the time of the call. One benefit has been to demonstrate to the developers that MPI has useful features that will help make their software more maintainable and generally easier to read and understand. This helps newer less experienced researchers get up and running quicker and will give them confidence to adopt and adapt the example functions.

VI. SUMMARY

The complexity of this software requires the storage of three dimensional data for several parameters. In several places there are calculations that stride through the data in a non-optimal manner. Attempts were made to try to reduce the computational overhead by determining a suitable sequence of the loops. Unfortunately no significant improvement was found for two key functions CONSOM and CALFLU.

An alternative version of GBSTAT did make significant improvement and allowed a new feature to be developed quite quickly (Satellite observation matching).

The modified SORZM provides significant improvement for reporting, with more fields being reported per simulation and less sensitive to the resolution of the simulation.

The PPRD subroutine provides a cleaner method for distributing data and the small improvement that was expected is lost in the timing due to the domination of the MPI_BCAST usage.

PPWR is unlikely to provide significant gains due to the difficulty in writing later records to a sequential file. These functions read the INI file and write the RST file in a serialized (although "Fortran unformatted") manner. A difficulty has arisen in that the Fortran file records do not correspond directly to a latitude set of data. This is because a sub-function is used to write out a whole plane as a single vector disregarding record boundaries.

However, both PPRD and PPWR have demonstrated the usefulness of built-in features of MPI and the data structures can be used elsewhere and extended to cope with different resolutions of auxiliary data. The maintainability of the code will be improved with the data types being defined in their own module and the simpler gathering and scattering functions making packing and unpacking automatic.

VII. FURTHER WORK, RECOMMENDATIONS

There are further sections of the software that will benefit from using the new data types and potentially gains will be achieved by modifying more of the files to be compatible with NetCDF. The existing results file is created with a serial version of NetCDF. It is currently being reviewed with a view to enabling the use of parallel NetCDF.

There several places where the modifications reported here could be further enhanced but the benefits have to be weighed against the cost of development.

The concept of using "per-row" writers for the diagnostic file and checkpoint file should be investigated along with parallel NetCDF. Discussion is in progress with the research group to see how much disruption would occur by changing the format of those files.

References

[1] M.Richardson, "Improving TOMCAT/GLOMAP Mode MPI for Use on HECToR, a Cray XT4h system", HECToR DCSE programme, www.hector.ac.uk , April 2009.

[2] M.Richardson, G.W.Mann, M.P.Chipperfield, K.Carslaw, "Implementation of OpenMP within MPI-TOMCAT-GLOMAP mode", Cray User Group Meeting, Edinburgh, May 2010. (also full report available on HECToR web-site)

[3] M.Richardson, M.P.Chipperfield, "Improving File Access Characteristics Of TOMCAT For High Resolution Simulations", HECToR DCSE programme, www.hector.ac.uk , April 2013