Improving TOMCAT/GLOMAP Mode MPI for Use on HECToR, a Cray XT4h system

A DCSE Project

Mark Richardson
Numerical Algorithms Group Ltd,
Wilkinson House, Jordan Hill Road, Oxford, OX2 8DR, UK,

September 2009

Abstract

The GLOMAP Mode MPI software and its host chemical transport model, TOMCAT, have been examined by DCSE support and changes have been implemented that improve their performance on HECToR, the UK National Supercomputing Resource, a Cray XT4h[1] system. Typically this code is used with a small number of cores (32) on the XT4 and for that configuration a small improvement of the order of (3.3%) is gained by changing the compiler optimisation flags from -O3 to -fast. Larger gains (16.2%) have been achieved when the code was refactored to work with planes of latitude in the chemistry section. Review of the code structure around the communications functions has resulted in improvement (9.4%). A reduction in time per step of 12.4% is seen when comparing the prevailing production version with the enhanced version, this is the accumulated effect of the three work packages.

The enhancements make the use of more cores more economical than can be used at present with an overall reduction in time per step of 16.5% when the simulation is run with 64 cores.

The improvements contribute to more efficient use of the HECToR resource and make more Allocation Units available for this research area. The analysis included in the appendices formed the basis for the improvements in communications and it can be used for further improvement such as parallel I/O and overlapping communication with computation.

[1] The "h" denotes hybrid as there is an attached Cray X2 system

Contents

Introduction

Background

This is the end of project report from the distributed Computational Scientific and Engineering (dCSE) Support project for the aerosol simulation code "GLOMAP" (Global Modelling of Aerosol Processes). This report gives an overview of the application and describes the work done to analyse its performance. It offers recommendations for changes that will improve the performance. Some of these have been implemented and the corresponding improvements are presented here. This document details the analysis, recommendations, implementation and resulting performance of GLOMAP mode MPI on HECToR XT4h.

This project was initiated by Dr. Graham Mann at the School of Earth and Environment at the University of Leeds. He is a NCAS-funded researcher. The National Centre for Atmospheric Science (NCAS) provides a national capability in atmospheric science research through its research programmes and a facilities and support infrastructure distributed over UK Universities

The distributed computational science and engineering (DCSE) support function is a resource provided by the Research Councils UK. It can be used to enhance the performance of computer programs and improve working practices for researchers using UK National High Performance Computing Facilities (HECToR). It is specifically a person identified to carry out the work in support of improving the use of national computing resources through better software engineering and education of the users of these facilities.

Numerical Algorithms Group Ltd is a not-for-profit commercial organisation that specialises in computation libraries for numerical methods in mathematics and they are a producer of a FORTRAN compiler. They support the HECToR facility by assisting users when they encounter problems running their applications. Additionally their work includes assessing the computational aspect of proposals to Research Councils and supporting the dCSE personnel.

HECToR is the High End Computing Terascale Resource. It is a Cray XT4h system with 5664 AMD64 Opteron CPUs (dual core) and 112 Cray X2 processors (more detail can be found at www.hector.ac.uk). Significant points to note for this project: there are 12 service nodes; the 8 actual "login" nodes, the two shared "aprun" job-launcher nodes and the two shared serial batch nodes. The job scheduler is PBS Pro (currently 8.1.4) with several queues that allocate various CPU resource sizes up to a maximum of 8192 and a job maximum duration of 12 hours. The default compiler is PGI and the default environment is focussed on the "XT" portion (i.e. Opteron processors). A "modules" system is used to set/change the operating environment. The system has recently changed (at the end of June) with the Dual Core processors being replaced by AMD quad-core processors (Barcelona-64), the login nodes and service nodes remain as Opteron dual core processors. In this report a "processor" refers to a component that can have multiple cores. A CPU is usually a reference to a core on a processor. PE is an abbreviation of "processing element" and is synonymous with core and

CPU. However, the X2 architecture has a single CPU per processor so PE is the preferred reference.

GLOMAP mode MPI is a FORTRAN computer program used to simulate global aerosol processes. It is a combination of a chemical transport model TOMCAT which has algorithms for integrating the gas phase chemistry/deposition and advection of the trace gas and aerosol species around the globe and the main part of GLOMAP which is a size-resolved aerosol microphysics module to simulate processes such as nucleation, coagulation, condensation and cloud-processing.

The simulation is the lower to mid-atmosphere (up to 10hPa) for the earth, the data set particular to this project has "moderate" resolution with 31 vertical levels (in hybrid σ-p co-ordinates) and T42 spectral resolution in the horizontal (2.8×2.8^{o} latitude/longitude) . The volume of fluid that forms the atmosphere is mapped onto a three-dimensional Cartesian rectangular block of computational space. The numbers of cells in each direction are as follows:

Number of latitudes: 128 (the I loop index within the code)

Number of longitudes: 64 (the K loop index within the code)

Number of vertical layers: 31 (the L loop index within the code)

GLOMAP uses "offline meteorology" to drive the transport and wet removal, reading in data produced from other numerical meteorological simulation software (6-hourly ECMWF analyses). There are two main versions of GLOMAP – Mode and Bin, using modal and sectional aerosol dynamics approaches respectively, with versions of each in Open MP and the subject of this research is GLOMAP Mode MPI. The Open MP version of the software has been used on SMP class machines (for instance the previous national HPC systems CSAR and HPCx) and thus has a soft limit maximum of number of threads set by the outermost loop (over latitude). At the resolution of the test case this is set to 64. The MPI GLOMAP versions have only been developed recently (2008) and differ from the Open MP versions mostly in the TOMCAT sections of the code, with MPI being used for communication between parallel tasks. Communications are handled via a distinct set of subroutines separate from the main TOMCAT code with similar naming convention. For example, the subroutine MPI_SENDRECV is wrapped inside MPE_SENDRECV. The interfaces are different in that fewer parameters are passed to the "MPE_" routines. Typically it is the parameter corresponding to the communicator that is omitted and within the wrapper as it is assumed that MPI_COMM_WORLD is the desired communicator. Also every "MPE_" routine has to make a call to MPE_TYPE to determine the classes of data object that is being communicated (MPI_DOUBLE, MPI_INTEGER, etc.) even though the indicator is a supplied as a parameter (MPREAL, MPINT).

The underlying chemical transport process is supported by the TOMCAT program and the GLOMAP features are inserted into that code at specific locations e.g. at the initialization stage and the aerosol chemical reaction stage. All of the GLOMAP process is based on an

individual computational cell ("grid-box") and thus has little impact on any parallel communications. It does have an impact on memory requirement. Currently a typical job configuration on HECToR will use 32 MPI Tasks and require 12 hours to simulate a year of atmospheric process. Usually simulations are broken down into jobs of one month (to enable examination of the intra-annual aerosol lifecycle) and chained through the PBS scheduler.

Geometric decomposition is used to partition up the work onto each computational node with MPI communication between each task. The lines of division lie along the lines of longitude and latitude, albeit not exact degrees (in this case the interval is 2.8125°).

The choice of domain decomposition is pre-set for GLOMAP wherein the attempt is to retain square "footprints" to reduce the ratio of surface area to volume. This is equivalent to minimising the communication to compute ratio. However, this is not necessarily the best option for the type of case for which this software is applied. Consideration should be given to loop length and hardware configuration.

When operating in MPI configuration the geometry is broken down into smaller blocks (patches) along lines of longitude and latitude. Each block retains a full column of atmosphere from the ground to the maximum altitude, for this study that is 31 layers. The choice of decomposition is pre-set based on the simulation.

Number of MPI Tasks	NPROCI	NLONMX	NPROCK	NLATMX	Remarks
2	2	64	1	64	Too coarse
4	4	32	1	64	Too coarse
8	4	32	2	32	
16	4	32	4	16	
32	4	32	8	8	
64	4	32	16	4	
128	4	32	32	2	Current maximum
256	16	8	16	4	
512	32	4	16	4	Minimum number of points in (I,K)

Table 1: range of decomposition topologies for T42 as used for this study

Table 1 shows the preset domain decomposition for T42. Although it is conceivable to use more than 128 PEs for the simulation it is not efficient as the ratio of computation to communication is reduced. The smaller decompositions require larger arrays and encounter memory limitations so they were not tested. This decomposition implies that the maximum number of sub-domains is 128 as the minimum numbers of computational cells per task in the longitudinal direction is 2 and the minimum number of cells per task in the latitudinal direction is set by a condition satisfied at the geometric poles of the earth i.e. along the planes of the minimum K value and the maximum K value. Further research into this may be beneficial as making the geometric domain decomposition simlar to the FFT decompostion would reduce the communication overheads ahead of that transformation.

Reminder of milestones

The original proposal was for 24 months effort but has been reduced to match the 6 months that was funded. The funded project followed the first four identified Tasks listed in the original proposal:

"Task 1 Examine performance & scalability of MPI-GLOMAP, MPI-TOMCAT on HECToR (month 1)
Here, performance of MPI-GLOMAP and the host MPI-TOMCAT model on HECToR will be examined in more detail using different compiler options on both the standard scalar MPP XT4 system and also the new vector "Black-widow" component introduced in August 2008. Efficient performance of the GLOMAP code on both vector and scalar systems is particularly important since the code also runs as part of the UKCA sub-model in the Met Office Unified Model which runs on Met Office NEC SX-6 and SX-8 vector supercomputers.
The most expensive GLOMAP routines will be identified as candidates for the optimization work in Task 2. Scalability tests over increasing CPUs will then be carried out at moderate resolution."

Thus, the first task is to analyse GLOMAP mode MPI, including raw wall clock timing and parallel scaling, provide a plan for enhancing the performance.

The second task includes 3 sub-tasks as seen in this excerpt from the original proposal:

"Task 2 Optimization & algorithmic improvements for MPI-GLOMAP (months 2-6). The profiling results for MPI-GLOMAP (section 3.2) suggest that the aerosol/chemistry interface "CHIMIE" and the particle coagulation routine are the two most expensive routines on the scalar MPP XT4 system. The "CHIMIE" routine mostly deals with array copying (and very little computation) and the high overheads suggest it is performing poorly on (scalar) HECToR. We expected the coagulation routine to be expensive and anticipate working with the CSE member to gain savings on this routine. Several other routines also contribute significantly and we expect to be able to achieve additional efficiency gains by inlining functions, replacing with approximations/look-up tables or using the latest algorithms from library calls (e.g. to carry out FFTs). Exploring these and other general code optimizations will form the first part of this task (M2.1). The 2nd part of the task will be to scope out and test possibilities for optimizations which exploit particular features of the HECToR architecture (M2.2) on the scalar and Cray X2 vector systems. Efficient use of cache and reduced communication overheads are likely to be important here. Task M2.3 will work to further enhance performance by exploring the use of parallel I/O in TOMCAT and GLOMAP. We expect that once (M2.1) and (M2.2) have been carried out, I/O overheads are likely to have become dominant when using many CPUs."

Thus the second milestone is (M2.1) general code optimizations, the third milestone (M2.2) is optimizations focused on the PE architectures (both Cray X2 and AMD64 Opteron), and

although the task states "use of cache" this was not investigated. The focus of this milestone was changed to be that of MPI communication efficiency. The fourth milestone (M2.3) is to analyse the file handling and recommend a plan for parallel I/O to avoid the bottleneck of the MASTER-I/O model.

Overview of existing infrastructure

A GLOMAP simulation is initiated from a pair of files: the case-specific input file and a "sub-file" which is Linux shell script to build, compile and run (including template PBS script commands). The sub-file can be submitted from the command line for testing purposes with the PBS commands and parallel specific items (e.g. aprun) removed or disabled. An additional perl script can be used to generate a sequence of PBS scripts (e.g. separate "sub-files" for each month in an annual run) with an additional job file that can be used to submit a sequence of chained steps for unattended submission of all the sub-files. This creates checkpoints (actual restart files) and allows experiments which take longer than the administration imposed queue wall time limit of 12 hours to be carried out. The first case-specific PBS script does the preparatory work that consists of generating code, building executables, copying files and launching the parallel run. The subsequent PBS scripts will issue restart instructions and preserve their result with a unique identifier.

Generating code

Several "update" files (.up) are generated using "*here documents*" and "*echo*" from within the sub-file script. The "*nupdate*" program is used to process the ".up" files that are a form of edit instructions, which modify a reference version of the TOMCAT subroutines. A final "prog.f" is generated by the nupdate system with the main GLOMAP code included via auxiliary source code files copied from reference directories and added with concatenation.

Building executables

There are several "*ftn*" statements within the job script that are issued to compile the FORTRAN source code and build the executables. The options to these are *-O3, -Mextend, -Mbyteswapio, -r8 and -i4*. They are PGI specific because that is the default compiler on HECToR. The value for optimisation level is set in the "*chained.functions*" script and stored in a shell variable *$COMPILER_SETTINGS*. It is a cautious setting and the code may benefit if it is changed. The results of the compilations are two executables **GLOMAP.exe** and **pdgc.exe**.

Copying reference data files

The script checks that a **GLOMAP.exe** has been created and if this is successful it then copies many reference files into the local directory (97 copy commands) as required when using the LUSTRE file system on HECToR. Symbolic links to other reference directories (in /work) are created that allow access to other reference data files at certain stages of the simulation.

Submitting job

In the case of a restart i.e. after the script has copied the file *"fort.30"* to the local directory, the *aprun* command is issued to launch the parallel run. The values for the aprun options are

processed by the perl script based on the content of the input file. For example, a simulation that uses 64 CPUs will have the following settings:

aprun –n 64 –N 2 ./GLOMAP.exe

Note this is used for the dual-core system. The "-N" setting will be "-N 4" for the quad core system. Some other files, *fort.93*, *fort.94* and *fort.95* are created from values set in the input file. Some of the code in the ".up" files is also modified during the processing of the script hence the need for "here" documents.

End of run - job completion

After the parallel job has finished, the standalone program "*pdgc.exe"* is run to convert the results from double precision to single precision with the newly-created fort.9 re-written to a new file "fort.25". This process reduces file size by 50% compared to the original fort.9. The fort.25 is renamed and moved at the end of script. Leeds researchers secure-copy these output files (known as PDG files) back to University of Leeds SEE and process/analyse them locally using IDL. This pdgc.exe is sequential code and is running on the queue manager node while the compute nodes are idle but still reserved. This can be detrimental for other jobs on the system and will use some of the project AUs.

Recommendations for changes to running the simulations

Recommendations for the infrastructure include separating out the unwieldy submission file into five logical stages: build, copy, parallel run, reduction, transfer result. The "nupdate" facility should be done separately and users can keep a catalogue of their own '*.up' files or enhanced versions of the "standard" *.up files. This is a lightweight sequential process that should be possible to be carried out interactively on login nodes. Building the executable can be achieved with a make file and this might also be done on the login nodes but would also work as a serial job submission. The conversion of the result ready for transfer should be done in a serial job but could probably form part of the job run if a parallel IO facility is used. The latter would speed up the creation of the fort.9 file and could embody the single precision conversion straight to a fort.25 (PDG) file.

Implementation of changes within DCSE

Some of these suggested changes have been adopted by the DCSE but have yet to get into working practice at University of Leeds. There is some work required for universal adoption of this philosophy and to make it straight-forward for any new researcher to take up the software and start working quickly.

As part of the dCSE work the creation of the case directory and source code is still part of a single submission script. However the script is exited after the restart file has been copied into the case directory, which is done only if **GLOMAP.exe** has been created. Source is then separated into a specific directory named to match the number of PEs requested and a copy of the generic make file is added. Repeat builds are simply a matter of issuing a command "*make -f make.pgf90 xtgmm*" no matter what the CPU numbers. Due to the nature of NFS and LUSTRE, the HECToR service administration recommend that source is stored in $HOME

and cases must be located on $WORK. The mechanism for bringing these together is the PBS batch file script and "**aprun**" command.

During the progress of this work prototypes and trial edits were carried out on the "prog.f" but then they were translated back into "nupdate" instructions in the single submission file. Eventually work was done on the original reference library so that multiple passes could be made for easier code generation.

Changes required for operation on Cray X2

Brief description of Cray X2

The Cray X2 is additional hardware attached to the Cray XT4, hence the designation "XT4h" indicating a hybrid machine. However, it is not truly hybrid as the X2 must be used independently from the XT4. It has a common operating system and a common file system that is hosted on the login nodes. The compute nodes have a variation of Cray Node Linux. The processor has large vector registers allowing a single instruction to be applied to many pieces of data at the same time. For example, there are *n* 128 word registers. If 128 elements of an array A are loaded into one register and 128 elements of B are loaded into a second register, a scalar C is loaded into the scalar register, then D= AxB+C can be calculated in six steps and not 768 steps it might take on a scalar processing system.

The compiler (CFTN 6.0.0.1) is hosted on the login node and cross compilation can occur at the command line interactively. The user has to activate the environment with "module load x2-env" then the "ftn" command is used for accessing the compiler. A different set of options are required for the compiler so a separate Makefile has been created. This allows the executable to be re-created a number of times without re-running the full job script.

Infrastructure adopted for work on Cray X2

The submission script, for an XT run, is halted before it issues the "aprun" command. At this stage the source code has been generated and the GLOMAP executable has been built using the PGI compiler. The source code has to be manually moved into separate directories from the case directory so that they are preserved for repeat builds using the "X2 Makefile (***make.cftn***)" and to help in using the performance analysis tools. The usual working practice in the GLOMAP group is to run the whole script for every simulation so source code is deleted with every run of the first PBS script. For this work a separate PBS script has been created to launch the parallel jobs. It is initiated from the login node command line, but could be embedded as a sub-step of an automatic job control harness. The modules for X2 environment have to be activated before the source code is compiled. The following command is now available for users: "module load x2-env".

A significant amount of effort was expended to get the code to work on the X2. Initially the compiler would flag warnings and sometimes error messages. Each of these was addressed and fixes were communicated with the code owner. During the run of the simulation the code would fail issuing "segmentation fault" errors. Most of these were tracked down and fixed.

Basic scaling data on Cray X2

To date, the code changes have been mainly to get the port to work. The difference between the ports is not only the hardware but the compiler and its compliance with FORTRAN standards. The Cray compiler (CFTN 6.0.0.1) is stricter than the PGI compiler and that highlighted several coding problems during the build. It also has a different memory management strategy that manifested at runtime with memory faults. The CFTN compiler is not as verbose as other compilers so the NAG FORTRAN compiler was used for the compilation stage to reveal areas of weakness. It is available on HECToR (F95 v5.1 for Linux 64bit). During the building process several coding errors were found in the TOMCAT sections and these were corrected. All changes were communicated back to the University of Leeds collaborators. The use of Cray PAT relies on successful completion of simulations so no significant work was done with that facility.

Table 2 shows timings of GLOMAP-MODE-MPI v1_gm2 on the X2 system. Figure 1 compares these with the equivalent for XT4. Note that the XT4 timings were generated on the upgraded quad-core XT4 system using the same source GM2 version of the code as for the X2 timings. They were compiled with the -fast flag and run with two only two cores per node.

GLOMAP Mode v2 on Cray X2 (base code) time in seconds			
Number of MPI tasks	8	16	32
Initialisation time	95	92	93
End step 143	1074	477	260
End step 144	1099	500	283
End of simulation	1131	534	321
Average time per step	6.89	2.71	1.17
Equivalent XT4 timing on two cores per QC node	12.57	5.25	2.61

Table 2: runtimes for GM2 on both X2 and XT4 for various domain decompositions

This shows a “super-scalar” behaviour compared to the expected non-ideal behaviour usually demonstrated by this category of application i.e. the “time per step” improves by a factor greater than 2. The runs that use higher number of PEs will have smaller domains that lead to better memory and cache usage. For the X2 there is 8GB per core, on the XT4 there is 2GB per core but in this case there is 4GB per MPI task, as the simulation is run with only two MPI tasks per node. The innermost loop length remains as 32 (1 to NLONMX) whereas the loop over latitudes is reducing by a factor of 2 (as seen in table 1; NLATMX is 32, 16, 8, respectivley for 8, 16 and 32 PEs).

The work using the X2 has been halted in favour of the XT4 system. Some of the reasons include: the effort to port the code, potential delays in tracking down each problem, the limited access to the X2 resource, the limited number of nodes on the system, competition in the queues. Also the fact that “mixed-mode” operation is not available on this X2 system has reduced the emphasis of research on the system even though the performance of GM2 on X2 showed it to be significantly quicker than the XT nodes. One bug remains within GM2 that manifested as a runtime failure when run on 64 PEs. Recent tests show that GM3 and GM4

also have the problem on 64 PEs on X2 and will require significant work to get either of them to run on this system.

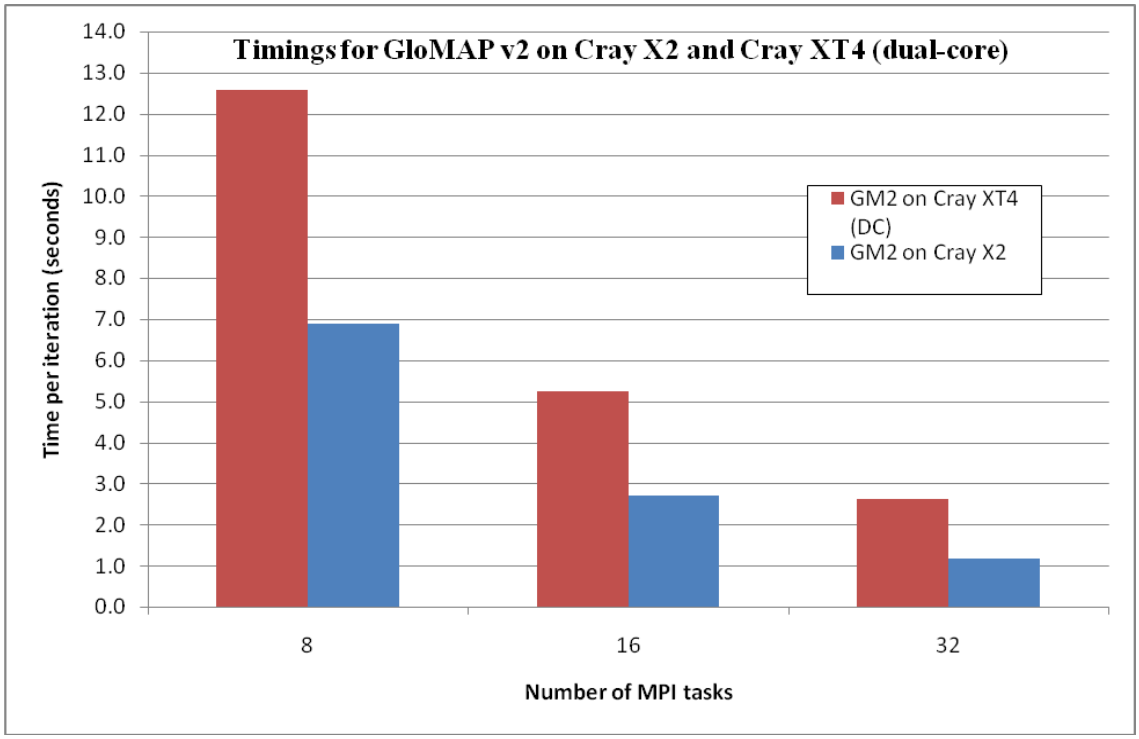


Figure 1: bar chart shows the performance of GM2 on both X2 and XT4

Summary of X2 work

Generally the performance of the Cray X2 hardware is expected to be much better than a node of the Cray XT4. It is rated at 25.6 Gflop/s where an XT4 core (in Dual Core form) is only 5.5 Gflop/s with the recent upgrade this has changed to 9.2 Gflop/s. However, those figures are arrived at from idealised LINPACK measurements. This investigation was limited by time as unforeseen problems arose. When the code has been completely reviewed for the XT4 system, then further investigation should be directed at the inlining and cache blocking options. These should provide further gains as will using the informational flags to identify sections of code where the compiler was unable to complete any automatic optimisations.

Task 1: Examine change in performance with various compiler options

Role of compiler options

Compilers can do a lot of work towards improving the performance of a specific piece of code. In this computer program the method for simulating the atmospheric processes is written in a high level language and is likely a good representation of the mathematical method. However, that does not always suit the platform on which the calculation takes place. Reviewing the code structure is one way to improve performance and that is discussed in task 2.1. Another method is to use the features of the compiler that analyse the high level language and translate it into efficient machine code tuned specifically to the target hardware.

The GLOMAP code is compiled with "-O3" because earlier work by the team showed a reasonable performance compared to using the default options (-O2). Two useful options are "-Mneginfo" and "-Minfo" that make the compilations more informative. As shown in this excerpt from the build:

pgf95 -Mneginfo -Minfo -fast -o xtgmm.exe prog.f produces this:

```
***102046, Memory copy idiom, loop replaced by call to __c_mcopy8***
102054, Loop not vectorized/parallelized: contains call
102058, Generated an alternate loop for the inner loop
       Generated vector sse code for inner loop
       Generated 1 prefetch instructions for this loop
       Generated vector sse code for inner loop
       Generated 1 prefetch instructions for this loop
```

For GLOMAP, the log file of a build shows many places where the compiler reports that some optimization could not be done. Not all of those have been investigated so there are opportunities to make changes and allow the optimization to proceed.

Investigation of compiler options

Compiler options are affected by code structure. For example, if a print statement or conditional is placed within a do loop then the vectorisation will fail. The current production version of the code is built using the PGI compiler using optimization level -O3. Adding "-fast" ahead of this option will activate several features of PGI compiler that target the AMD64 chip and expect to provide performance improvements. The features include vectorisation, inlining and loop unrolling.

-fast : Common optimizations; includes **-O2 -Munroll=c:1 -Mnoframe -Mlre -Mautoinline -Mvect=sse -Mscalarsse -Mcache_align -Mflushz**

Unfortunately the combination of -fast with -O3 gave erroneous results dicussed later.

The option "-Minline" directs the compiler to analyse the supplied file for sub-programs that are called often and to make the assembler code for them inserted at the point of calling. This leads to larger sized executables but they are more efficient at runtime.

The following options are desirable in addition to those stated above:

-Mipa=inline -Mipa=reshape

Many codes benefit from a feature known as "inter-procedural performance analysis" that is available with the PGI compiler using the "-Mipa=" option. During the compilations of the sub-programs the compiler identifies sections where in-lining can be used. There are many places where simple wrapper functions are used and these can be removed by the compiler. At the link stage the object files are re-visited and compiled with the new information. This gives a more efficient executable. In many places the shape of the array is changed when entering the sub-routine, converting two-dimensional arrays into one-dimensional arrays so the option "reshape" is also supplied.

However, attempting to apply inlining to GLOMAP had two effects: first, the advanced IPA inlining revealed a bug in the compiler where the compilation failed with compiler internal error messages. Secondly, using -fast with -O3 resulted in incorrect results and so the simpler optimization was retained (-fast) as it gives an improvement of speed due to better vectorisation and some "safer" inlining. Time should be set aside with future projects to review the compilation flags and compiler version when the bug-fix is available.

The source code is generated as a monolithic file (plus a couple of GLOMAP specific files) that is useful for a multiple pass compiler as the fast flag includes a simple single level inline facility. When all functions are in the same source file the compiler can more easily identify candidates for inlining. It provides a better chance for the optimisations to be applied automatically. However, it may be that some optimisations are detrimental to specific subroutines so a method of applying compiler options to individual subroutines would help target specific optimisations. Thus it would have been useful if the IPA function had worked correctly. Another reason that this has not been done for GLOMAP is that it would deviate widely from the existing working practice of keeping jobs simple for researchers.

Results of varying compiler options

The initial work was done with "GM2" but updated versions were supplied during the project timescale. GM3 was provided in February 2009 and GM4 was provided in March 2009

Initial tests were evaluated using the dual core system. Table 3 shows the overall runtime difference for both GM3 and GM4 for various compiler optimisation levels.

Optimisation level	-O3	-fast	-fast -O3*	-Minline -fast -O3
Version GM3 (DC)	390.34	-	369.09	385.81
Version GM4 (DC)	349.64	332.91	327.93	372.55
Percentage change from "-O3 only"	0	-4.78%	-6.21%	+6.55%

Table 3: Performance changed by compiler flags for 64 MPI Tasks on Cray XT4 (Dual Core). NOTE: **the combination of –O3 and –fast give incorrect results.*

In these early tests the attention was mainly on the total elapsed time. But soon it was realised that the initialisation time was inconsistent so later analyses break up the timings into initial step, time for 143 steps, 144 steps and the end of the simulation. Thus, the measure of the

time per step is more representative of the normal use of the program. This is how the results are presented in tables 4 and 5.

The tests were repeated for the upgraded quad-core system for a selection of decompositions and provides a measure of how the quad-core affects the existing working practice of using the -O3 option for optimisation. Tables 4 and 5 show the break down of the runtime for each optimisation over several decompositions.

NCPU	**8**	**16**	**32**	**64**
Tinit	98	51	148	168
T143	1583	834	597	502
T144	1597	845	608	538
Ttotal	1605	855	621	563
Ttot-T143	22	21	24	61
T143-Tini	1485	783	449	334
(T143-Tinit)/142	**10.45**	**5.51**	**3.16**	**2.35**
Speed-up	1.00	1.89	3.31	4.44

Table 4: Baseline GM4 using "-O3" fully populated quad-core nodes time in seconds

NCPU	**8**	**16**	**32**	**64**
Tinit	116	56	63	108
T143	1503	798	497	410
T144	1527	807	507	423
Ttotal	1534	817	519	445
Ttot-T143	31	19	22	35
T143-Tini	1387	742	434	302
(T143-Tinit)/142	**9.77**	**5.23**	**3.06**	**2.13**
speed up	1.00	1.86	3.19	4.59
Change in time per step over "-O3 only"	-6.51%	-5.08%	-3.16%	-9.36%

Table 5: Baseline GM4 using "-fast" fully populated quad-core nodes time in seconds

The additional "hidden" compiler option is "*-tp=barcelona-64*" that is activated using "module load xtpe-quadcore" [now xtpe-barcelona] ahead of the build. If the build is initiated within the job script then the modules command has to be issued within the script as well. The same source code was compiled into an executable using the standalone Makefile (make.pgf95 seen in Appendix A) with the optimisation level set to "-fast" and with the xtpe-quadcore module loaded.

These values are the best achieved from several simulation runs and the time per step for each is plotted in figure 3. The logistics of running a sequence of these tests is straightforward in terms of setting up the PBS scripts to chain in turn. However, there is some question about the repeatability as in a few tests the initialisation times are slower and the "-fast" simulations were slower than the "-O3" simulations. There is a further level of investigation needed relating to how quickly files are available to the program through the LUSTRE file system. It may be that slow initialisation is similar to the lag that users experience when issuing file commands (for example, "ls" can take a few seconds when issued withing the LUSTRE file system).

The changes were validated by comparing the fort.25 files to a reference file using IDL on the SEE departmental server. Figure 2 shows an example of a failed comparison that is indicated by the large variation between the “perturbed” and “control” data.

As well as the general results failing due to the combination of -fast and -O3 setting of the compiler, there are also clear artifacts of the data decomposition in the form of vertical lines along the domain boundaries. This “failed” comparison compares a reference 4 PE decomposition with a 16 PE decomposition and was traced to an error in the array dimensions in SPEGRD, SPETRU1 and GATHERROW and SCATTERROW. The fix has been supplied to the code owner. In a successful comparison these figures would be uniform white and show zero variation.

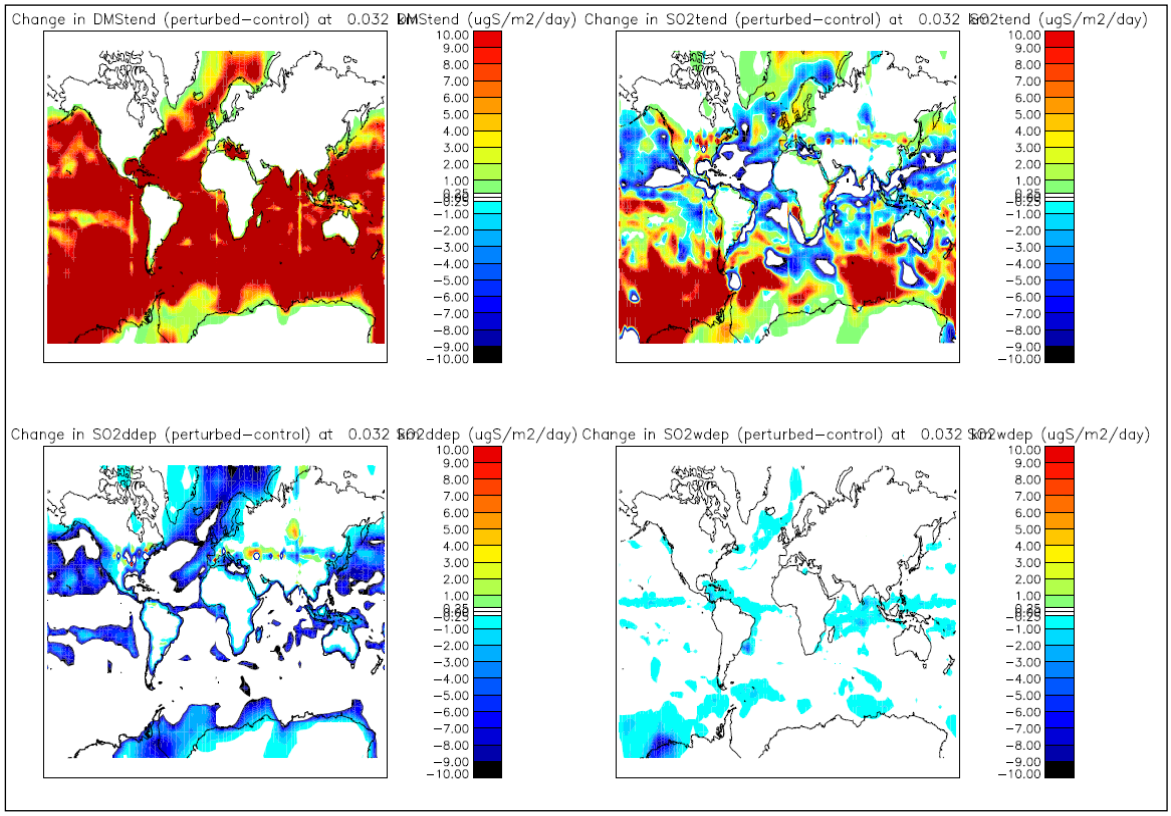


Figure 2: comparison of runs of GM2 on Cray X2; control is a 4PE decomposition; perturbed is a 16PE decomposition

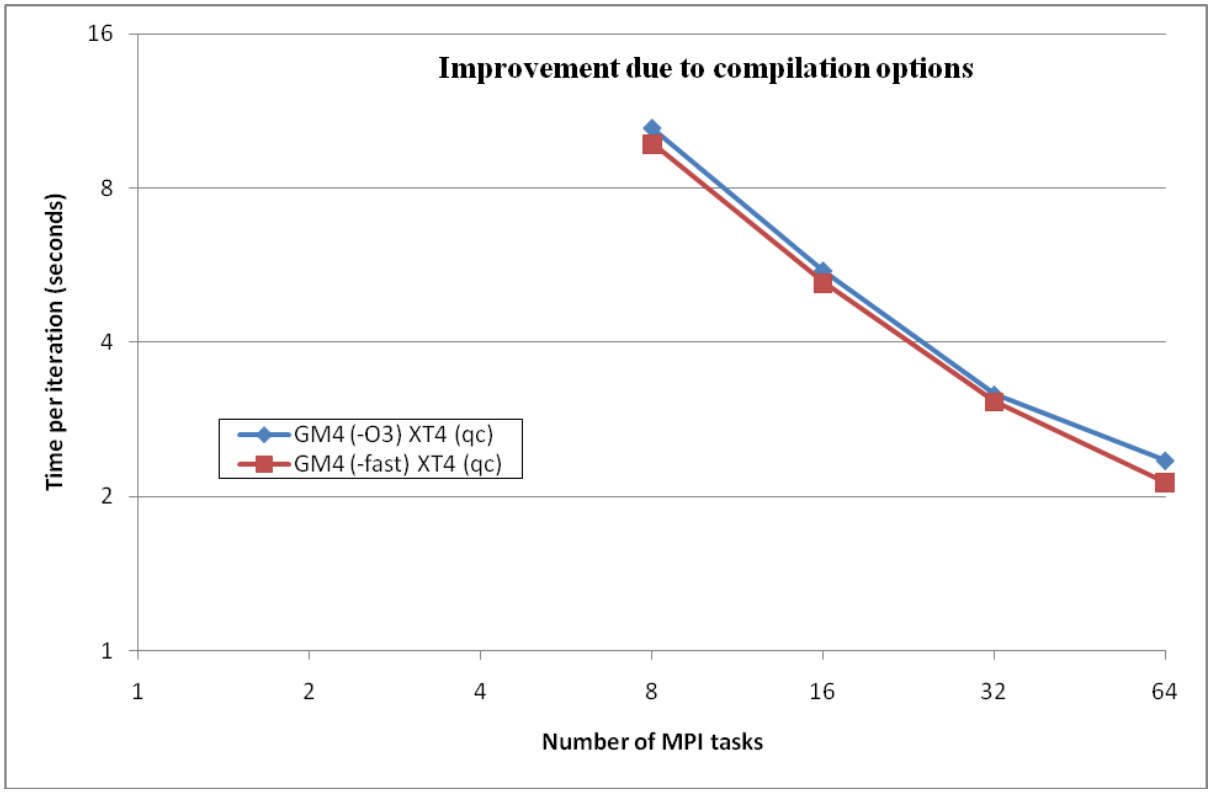


Figure 3: performance of GM4 for two different optimization levels of the PGI Fortran compiler on Cray XT4 with Quad Core nodes.

The sudden change from what can be consider good scaling at 32PEs to poor scaling for 64PEs is not easily explained. The values for NLONMX and NLATMX shown in table 1 indicate that the decomposition is such that only the loop over latitudes is changing and actually reducing. This suggests that some work unrelated to the K loop is beginning to dominate the simulation. The data being handled by the chemistry sub-steps are planes of (32x31) gridboxes and there are almost 200 species being evaluated so it is a significant amount of work. However, these planes are the same size for all decompositions in this study.

Conclusions from Task 1

The extensive range of options could not be used due to issues with the compiler. The "-fast" option gives an improvement over the "-O3" option currently in use. Changing the compilation flag has resulted in reduced time for simulations but has a small affect on scalability. This is shown in figure 3 and detailed in Table 13 of the Overall Summary. The optimisations apply to all the code but will not affect the parallel communication patterns. The parallel communications are reviewed in a later task.

Care must be taken to inspect the result whenever a different optimisation is added to the building of the executable. A basic reference case can be created using the –O0 (zero optimisation) option. The simulation would take longer to perform but the results will stand for a controlled comparison.

Task 2.1: General code optimizations

Performance analysis tools

Cray have supplied performance analysis tools that insert system call "hooks" into the object code and later instrument the executable depending on the type of experiment to be performed. The separation of the GLOMAP simulation process into five parts facilitates the requirement for repeat building of the executable with the Cray PAT tools. The code is built once normally and tested. The environment is changed to activate the Cray PAT applications and then executable must be re-built within this new environment. Different types of experiments can be carried out depending on the method of re-building the executable. There are two main ways of analyzing the performance of the code -- sampling experiments and tracing experiments. A sampling experiment is where the execution of the code is tested at regular intervals (100 milliseconds) and a record is made of which routine is in progress at that sampling point. A tracing experiment targets specific functions and records the accumulated amount of time that the function takes to process. During the run of an instrumented executable, extra output in the form of an experiment file (*name.xf*) is generated. Subsequent processing of the experiment file provides a text report to standard output and some additional files containing information about the performance of the program.

Earlier sampling experiments on HECToR by the researchers at the University of Leeds (see original proposal) had revealed certain subroutines consuming a large proportion of the runtime of the simulation. The Cray PAT analysis was repeated and confirmed those findings as described below.

Sampling reports

The basic sampling experiment reveals that an expected four routines use a lot of the processing time, they are *ADVY2*, *ADVX2*, *ADVZ2* and *CONSOM*. Previous analyses in the long history of the TOMCAT code have also shown these characteristics. However, in this case several routines appear with unexpected high load values, particularly for a low CPU count i.e. *CHIMIE* and some of the "*UKCA*" subroutines. These can be seen in the "USER" section of figure 4. Two of these routines were investigated further by replacing them with modified versions.

The outputs from several sampling experiments were used to identify where the high workload is in the specific subroutines. A typical sampling report has 3 sections: USER, MPI and ETC. figure 4 shows an excerpt (the listings can be very long) with the USER section. The 58 % shows that more than 40% is spent doing other things than the calculations specific to GLOMAP. That information is given in the ETC and MPI sections that will be discussed in other parts of this report. This section indicates which line numbers in the source code consumed the most processing time.

```
Cray PAT sampling report for 32 PEs excerpts from GM3
experiments to enhance UKCA_COAGWITHNUCL().
Table 1:  Profile by Function

 Samp % |  Samp |    Imb. |   Imb. |Group
        |       |    Samp | Samp % | Function
        |       |         |        |  PE='HIDE'
 100.0% | 30997 |      -- |     -- |Total
|-----------------------------------------
|  58.5% | 18143 |      -- |     -- |USER
||----------------------------------------
||   8.3% |  2579 |  112.91 |   4.3% |advy2_
||   7.5% |  2314 |   44.16 |   1.9% |chimie_
||   5.5% |  1696 |   81.25 |   4.7% |ukca_coagwithnucl_
||   4.1% |  1260 |   27.44 |   2.2% |advz2_
>snip<
Table 2:  Profile by Group, Function, and Line
 Samp % |  Samp |    Imb. |   Imb. |Group
        |       |    Samp | Samp % | Function
        |       |         |        |  Source
        |       |         |        |   Line
        |       |         |        |    PE='HIDE'
 100.0% | 30923 |      -- |     -- |Total
|-----------------------------------------------
|  58.5% | 18090 |      -- |     -- |USER
||----------------------------------------------
||   8.3% |  2577 |      -- |     -- |advy2_
3|       |       |         |        |
gm3_spunup/CoagWithNucl/src_32e/prog.f
||||--------------------------------------------
4|||   1.2% |   385 |   47.56 |  11.3% |line.13203
4|||   3.4% |  1047 |   78.59 |   7.2% |line.13370
4|||   3.6% |  1110 |   74.16 |   6.5% |line.13459
||||============================================
||   7.5% |  2310 |      -- |     -- |chimie_
3|       |       |         |        |
gm3_spunup/CoagWithNucl/src_32e/prog.f
||||--------------------------------------------
4|||   1.3% |   408 |   26.62 |   6.3% |line.32966
4|||   3.6% |  1101 |   11.38 |   1.1% |line.33056
4|||   1.4% |   419 |   50.06 |  11.0% |line.33300
||||============================================
||   5.5% |  1694 |      -- |     -- |ukca_coagwithnucl_
3|       |       |         |        |
gm3_spunup/CoagWithNucl/src_32e/GLOMAP.f90
||||--------------------------------------------
4|||   1.5% |   466 |   30.66 |   6.4% |line.3422
||||============================================
```

Figure 4: Example of part of a sampling report for GLOMAP

The investigations into CHIMIE and UKCA_COAGWITHNUCL were separated to be clear where any gains originated. The main aim is to try and reduce the time spent in CHIME and UKCA_COAGWITHNUCL.

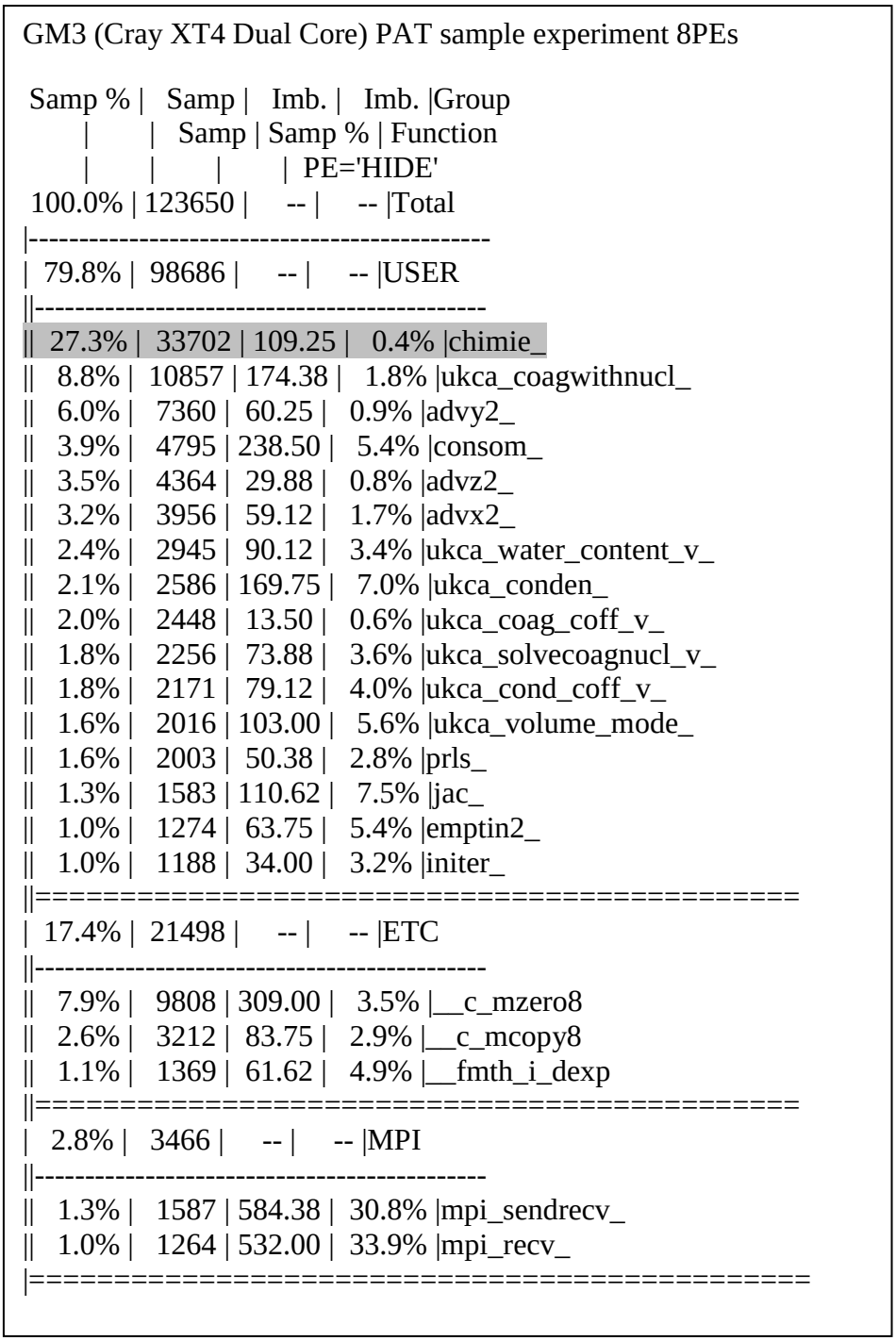
A subsequent code review (through inspecting the source files) revealed that, in some cases, there was unusual organisation of loop order. For example, the TOMCAT part of this simulation code deals with rectangular Cartesian coordinates and thus has a pattern of loops in many places that follow a cycle as follows:

```
DO L = 1, NIV
  DO K= 1, NLATMX
    DO I = 1, NLONMX
      WORK ON ARRAYS WITH INDICES (I,K,L)
    ENDDO
  END DO
END DO
```

This is the best method of accessing the Fortran arrays as it matches the manner of storage in memory. The TOMCAT parts of the code have been long established and optimsed for vector processors. However, in this more recent version of the code some of the high workload lines of source indicated by the sampling experiment were in places where this “natural” loop ordering was not being followed. In some places this was clearly an oversight of programming and was rectified. In other places there is a high workload due to the conversion from data structures organised for use with the advection methodology into data structures that conform to those needed for the FFT solution. There is some analysis of this in Appendix B for SPETRU1, GATHERROW and SCATTERROW.

The samples in figure 5a and 5b show the three groups of information: USER, MPI, ETC. they show the change in workload for the test simulation when run at different decompositons; one for eight cores and one for 64 cores. They also indicate the increase in importance of MPI for a higher PE count.

The function CHIMIE shows as a high workload for low processor counts. It converts the three dimensional data structures into one-dimensional arrays of equivalent size. That information is used in the UKCA_AEROSTEP sub-system. It works on the three dimensional volumes that form the proportion of atmosphere being simulated by a particular MPI task. The whole three-dimensional domain is mapped to a one-dimensional array.



GM3 (Cray XT4 Dual Core) PAT sample experiment 8PEs

```
 Samp % |   Samp |   Imb. |   Imb. |Group
        |        |   Samp | Samp % | Function
        |        |        |        |  PE='HIDE'
 100.0% | 123650 |     -- |     -- |Total
|-----------------------------------------------
|  79.8% |  98686 |     -- |     -- |USER
||----------------------------------------------
||  27.3% |  33702 | 109.25 |   0.4% |chimie_
||   8.8% |  10857 | 174.38 |   1.8% |ukca_coagwithnucl_
||   6.0% |   7360 |  60.25 |   0.9% |advy2_
||   3.9% |   4795 | 238.50 |   5.4% |consom_
||   3.5% |   4364 |  29.88 |   0.8% |advz2_
||   3.2% |   3956 |  59.12 |   1.7% |advx2_
||   2.4% |   2945 |  90.12 |   3.4% |ukca_water_content_v_
||   2.1% |   2586 | 169.75 |   7.0% |ukca_conden_
||   2.0% |   2448 |  13.50 |   0.6% |ukca_coag_coff_v_
||   1.8% |   2256 |  73.88 |   3.6% |ukca_solvecoagnucl_v_
||   1.8% |   2171 |  79.12 |   4.0% |ukca_cond_coff_v_
||   1.6% |   2016 | 103.00 |   5.6% |ukca_volume_mode_
||   1.6% |   2003 |  50.38 |   2.8% |prls_
||   1.3% |   1583 | 110.62 |   7.5% |jac_
||   1.0% |   1274 |  63.75 |   5.4% |emptin2_
||   1.0% |   1188 |  34.00 |   3.2% |initer_
||==============================================
|  17.4% |  21498 |     -- |     -- |ETC
||----------------------------------------------
||   7.9% |   9808 | 309.00 |   3.5% |__c_mzero8
||   2.6% |   3212 |  83.75 |   2.9% |__c_mcopy8
||   1.1% |   1369 |  61.62 |   4.9% |__fmth_i_dexp
||==============================================
|   2.8% |   3466 |     -- |     -- |MPI
||----------------------------------------------
||   1.3% |   1587 | 584.38 |  30.8% |mpi_sendrecv_
||   1.0% |   1264 | 532.00 |  33.9% |mpi_recv_
|==============================================
```

Figure 5a: Comparing sample experiments for 8 and 64 domain decompositions. NOTE the domination of CHIMIE on 8PEs compared with figure 5b that follows.

GM3 (Cray XT4 Dual Core) PAT sample experiment 64PEs

```
 Samp % |  Samp |    Imb. |   Imb. |Group
        |       |    Samp | Samp % | Function
        |       |         |        |  PE='HIDE'
 100.0% | 22117 |      -- |     -- |Total
|-----------------------------------------------
|  39.1% |  8647 |      -- |     -- |USER
||----------------------------------------------
||   5.3% |  1179 |  107.09 |   8.5% |advy2_
||   3.9% |   871 |   38.41 |   4.3% |chimie_
||   3.5% |   781 |   40.91 |   5.1% |ukca_coagwithnucl_
||   2.7% |   601 |   15.22 |   2.5% |advz2_
||   2.7% |   589 |   10.84 |   1.8% |consom_
||   2.3% |   512 |  270.48 |  35.1% |advx2_
||   1.6% |   348 |  112.12 |  24.8% |emptin2_
||   1.4% |   312 |   50.08 |  14.1% |ukca_water_content_v_
||   1.3% |   297 |  160.19 |  35.6% |fillin2_
||   1.3% |   279 |   66.77 |  19.6% |prls_
||   1.1% |   241 |   18.44 |   7.2% |ukca_coag_coff_v_
||   1.0% |   218 |  283.30 |  57.4% |spetru1_
||==============================================
|  32.2% |  7118 |      -- |     -- |MPI
||----------------------------------------------
||  15.8% |  3486 | 2032.61 |  37.4% |mpi_recv_
||  11.9% |  2638 | 2207.33 |  46.3% |mpi_sendrecv_
||   3.8% |   834 |  668.56 |  45.2% |mpi_ssend_
||==============================================
|  28.7% |  6352 |      -- |     -- |ETC
||----------------------------------------------
||   7.2% |  1595 |   90.95 |   5.5% |__c_mzero8
||   7.0% |  1548 |  421.45 |  21.7% |PtlEQPeek
||   1.9% |   429 |   54.33 |  11.4% |__c_mcopy8
||   1.8% |   395 |  139.33 |  26.5% |PtlEQGet
||   1.7% |   372 |  158.47 |  30.4% |PtlEQGet_internal
||   1.0% |   215 |   79.30 |  27.4% |ptl_hndl2nal
|==============================================
```

Figure 5b: Comparing sample experiments for 8 and 64 domain decompositions. NOTE the flatter profile for 64PEs.

The differences between figures 5a and 5b show that the performance of simulation is sensitive to the number of MPI tasks available for calculation. It is highlighted that CHIMIE occurs lower in the sampling table for a higher number of MPI tasks implies that the size of the problem has a direct influence on the workload. The more cores (or MPI tasks) applied to the problem reduce the size of the domain being simulated. The CHIMIE method in use with the GM3 version of GLOMAP mode processed the whole volume of atmosphere in pass. This was modified so that the UKCA_AEROSTEP sub-system works on a single plane of

latitude at a time, this isrefered to as “GM4” in this report. The planes are organised by latitude so the dimensions are the number of longitude points by the number of vertical layers. In the test case used that is 1/4 the quantity of data that was being processed per domain in GM3 for the 64 PE decomposition and 1/32 of the data being processed for an 8 PE domain. This is the method used by the chemistry section of TOMCAT and how earlier versions of GLOMAP processed the data.

Tracing experiments

A baseline source was compiled for PAT sampling using the APA guided option. This generated an “apa” file. The “apa” file contains all the guiding information for instrumenting the executable for the tracing experiment. Repeated builds of the executable for tracing experiments are possible without repeating the sampling experiment. The tracing experiment provides a new experiment file that is processed by the pat_report tool. The report includes names of traced functions and the time spent in the process of executing them.

Cray PAT provides an interface (API) that allows the investigator to insert specific timing points to break up a subroutine tracing information. The timing points are given name-tags and these are used to identify the sections in the post-processed experiment files. This method was used extensively to investigate the UKCA_COAGWITHNUCL function that appears high in the league table for the sample and trace reports. In the figure 6 the case was run with 8 MPI tasks. It is the GM3 code and so CHIMIE continues to show high in the work load. The five labels in bold on the end column are user defined through the API calls to “pat_region_begin”.

```
Cray PAT API "11 loop init mtran" in UKCA_COAGWITHNUCL
      call pat_region_begin(11,'11 loop init mtran',pat_stat)
!
!!!       DO IMODE=1,NMODES
!!!        NDOLD(:,IMODE)=ND(:,IMODE)
!!!        DO ICP=1,NCP
!!!         MDOLD(:,IMODE,ICP)=MD(:,IMODE,ICP)
!!!         DO JMODE=1,NMODES
!!!          MTRAN(:,IMODE,JMODE,ICP)=0.0
!!!         ENDDO
!!!        ENDDO
!!!       ENDDO
!
! replace triple loops above with F90 array syntax (let
compiler decide)
      NDOLD=ND
      MDOLD=MD
      MTRAN=0.0
      call pat_region_end(11,pat_stat)
```

Figure 6: problematic do loops in UKCA_COAGWITHNUCL with alternatives

In figure 6 the alternative coding for the original nested do loop structure is shown. For example, the “*11 loop init mtran*” is a block of code that initialises three arrays to zero.

Several different methods were investigated. These included: using F90 syntax and separating out the individual variables from the triple loop nest. It appears that the compiler was doing a good job and replacing them with the "idiom: `__c_mzero8`" reported when option "-Minfo" is activated.

In the tracing report shown in figure 7, the numbered hash labels (***#11, #5, #2, #10, and #7***) were inserted by the investigator using the PAT API. Actually regions numbered 1 to 11 were inserted but the other sections do not feature high in the workload. This is why the UKCA_COAGWITHNUCL reference is no longer visible.

Unfortunately no clear coding alternatives could be found for all the identified work as the compiler is doing its best with those sections of code.

GM3 Cray PAT **tracing** report with API information from 8 PE run when investigating enhancement of UKCA_COAGWITHNUCL

```
Time % |        Time | Imb. Time |   Imb. |   Calls |Group
       |             |           | Time % |         | Function
       |             |           |        |         |  PE='HIDE'
 100.0% | 2058.854843 |        -- |     -- | 6731515 |Total
|-----------------------------------------------------------------
|  94.5% | 1945.732550 |        -- |     -- | 6557264 |USER
||----------------------------------------------------------------
||  36.9% |  759.508205 |  6.805455 |   1.0% |     144 |chimie_
||   6.9% |  142.135909 |  8.451937 |   6.4% |    1440 |#11.11 loop init mtran
||   6.8% |  140.708321 | 41.589230 |  26.1% |       1 |main
||   6.6% |  136.060100 |  2.901422 |   2.4% |     288 |advy2_
||   4.0% |   82.400176 |  1.758459 |   2.4% |     144 |consom_
||   3.9% |   79.572205 |  0.500504 |   0.7% |     144 |advz2_
||   3.1% |   64.781270 |  2.547494 |   4.3% |     288 |advx2_
||   2.4% |   50.124924 |  1.249039 |   2.8% |    3456 |ukca_water_content_v_
||   2.4% |   48.938390 |  2.616689 |   5.8% |    1440 |ukca_conden_
||   2.3% |   47.166706 |  1.214859 |   2.9% |    4032 |ukca_coag_coff_v_
||   2.1% |   43.071936 |  3.410142 |   8.4% |    1440 |#5.5 imode loop
||   2.0% |   41.345881 |  0.211065 |   0.6% |     864 |ukca_volume_mode_
||   1.9% |   39.174169 |  2.602136 |   7.1% |    7200 |ukca_solvecoagnucl_v_
||   1.8% |   38.046784 |  1.764512 |   5.1% |   14400 |ukca_cond_coff_v_
||   1.8% |   37.836103 |  2.580603 |   7.3% |    1440 |#2.imode loop
||   1.8% |   37.507454 |  1.191105 |   3.5% |   25920 |#10.10 ncp loop
||   1.8% |   36.644806 |  1.473848 |   4.4% |    1440 |#7.7 imode loop
||   1.7% |   34.149579 |  1.831906 |   5.8% |    2617 |prls_
||   1.4% |   28.488018 |  1.145093 |   4.4% |    2617 |jac_
||================================================================
|   3.6% |   73.941021 |        -- |     -- |  155216 |MPI
||----------------------------------------------------------------
||   2.1% |   42.940114 | 10.573601 |  22.6% |   29385 |mpi_sendrecv_
||   1.0% |   20.905757 |  8.770574 |  33.8% |   35596 |mpi_recv_
||================================================================
|   1.9% |   39.181272 |        -- |     -- |   19035 |MPI_SYNC
||----------------------------------------------------------------
||   1.4% |   28.908872 |  4.654048 |  15.8% |    1123 |mpi_bcast_(sync)
|================================================================
```

Figure 7: tracing report example for investigation of "UKCA_COAGWITHNUCL".

The investigation of CHIMIE has resulted in several changes. The outer loop is over latitudes so what were originally transformations three-dimensional to one-dimensional for the whole

domain have now become mapping from three-dimensional to one-dimensional in planes of latitude. The following additional modifications are present in GM4:

> Calculate JLABOVE at same time as JL
> Several "ground level" arrays are now done in a smaller separate loop to avoid a conditional test for mapping 2d to 1d.
> Similarly, mode-radius and mode-number are available only at ground level so these are moved outside a three dimensional loop into a (now) one-dimensional loop.
> Revise JL index calculation and separate S0A, S0G and STG mappings to two dimensional arrays from 4D
> Reset JLABOVE at edge of atmosphere

And for the reverse mappings:

> for SOA, S0G,STG make index counter JL a simpler calculation when mapping from 2D back to 4D

Some other changes were identified although the limitation of time did not allow further investigation. Those sections include: an interpolation method for the "6-hourly" concentrations, further restructuring of conditional statements and some other loops where JL is explicitly calculated rather than being an incremental counter.

Results of changes to code structure

A replacement version of GM3 has been produced (GM4). The main difference is in CHIMIE which processes the data for the UKCA_AEROSTEP in planes of latitude. The data are mapped from multi-dimensional arrays (selected as latitude planes of longitude by altitude) into the one-dimensional arrays used for the aerosol and chemical calculations. The difference in performance between the two versions including the change in optimisation choice determined in Task 1 is shown in table 6 and figure 8.

Number of MPI tasks	**8**	**16**	**32**	**64**	**128**
GM3 seconds per step (dc with -O3)	13.75	6.15	3.18	1.95	1.64
GM4 seconds per step (dc and -fast)	7.91	4.45	2.66	1.77	1.29
Improvement (%)	42.49	27.62	16.27	9.09	21.12

Table 6: Time per step for different decompositions comparing GM3 and GM4 with different optimisations. An indication of the full benefits of changes.

The greatest gains are where the domain sizes are larger. For example, the volume being processed by an 8 task decomposition is 31744 boxes for GM3 method (32x**32**x31) but using the GM4 method the number of boxes is 992 boxes (32x31). The 64 task decomposition is processing 3968 boxes for the GM3 method (32x**4**x31) but is processing 992 boxes for the GM4 method (32x31). The array sizes used in the chemical sub-step (UKCA_AEROSTEP) are significantly smaller when the data is processed by latitude and is indirectly proportional to the number of latitudes being processed on a sub-domain. For the T42 resolution there is 1/64 of the data processed although there is an extra loop for latitudes at a higher level (in the CHIMIE subroutine).

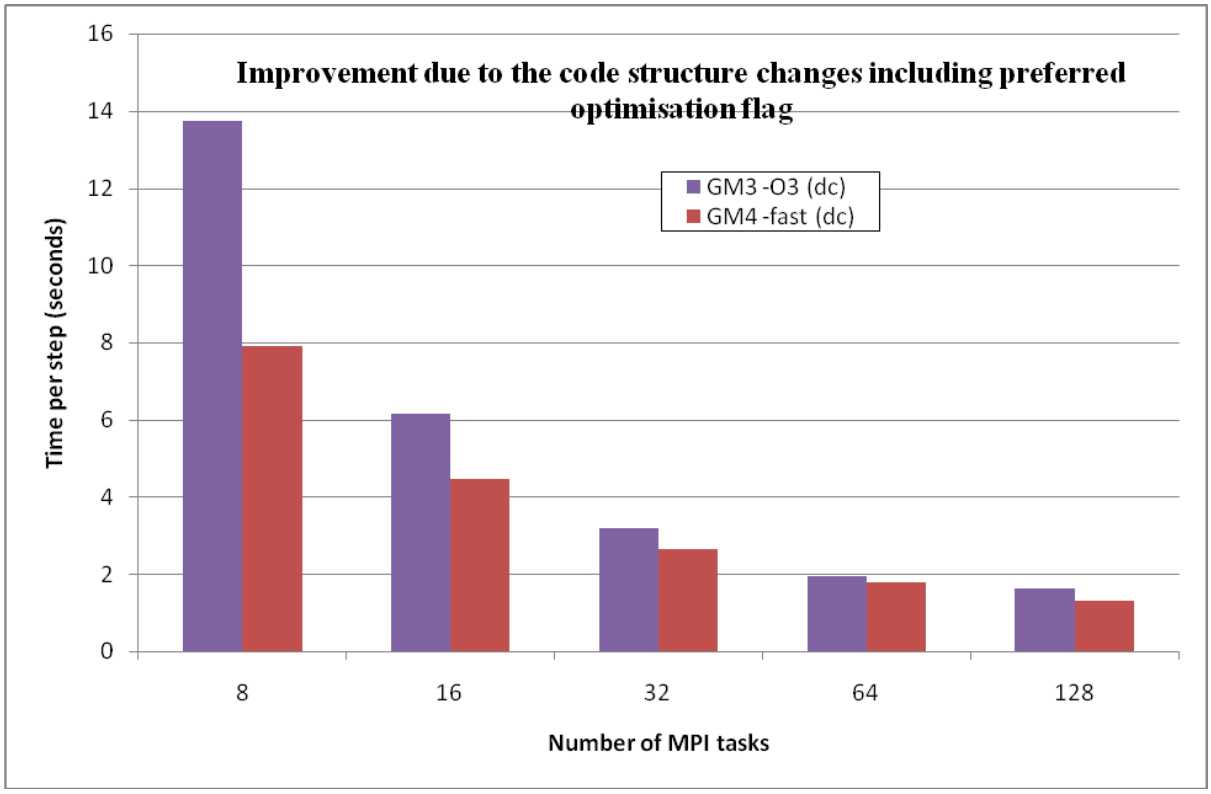


Figure 8: shows a comparison of the performance of GM3 to GM4

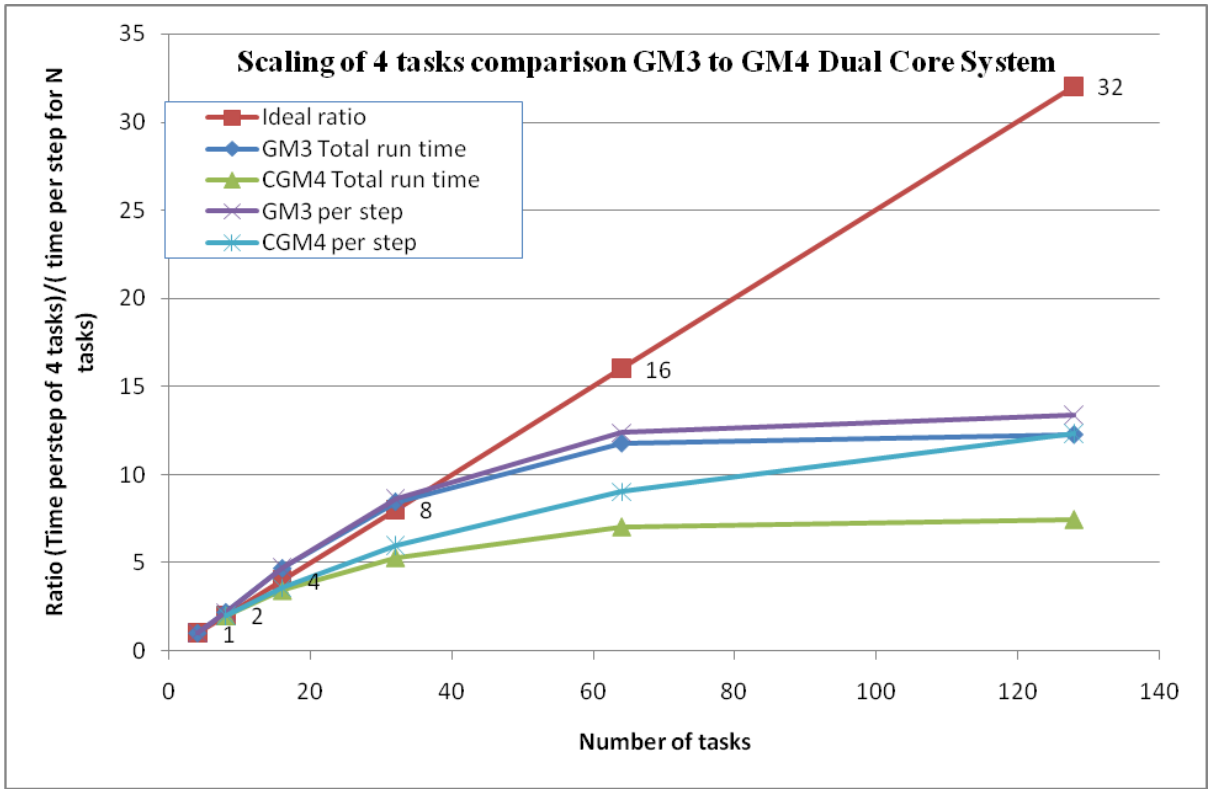


Figure 9: Expressing the time per step in relation to the time for a simulation using 4 MPI tasks.

The curves in figure 9 are an indicator of speed-up. The unfortunate effect of improving the speed of the serial sections is that the speed-up appears to worsen (already the scaling was considered poor for 64 PEs). However, if the first and final iterations are ignored (purple and cyan lines) then the scaling of GM4 has recovered to match that of the GM3 case.

The fact that the percentage improvement reduces for a higher number of PEs is related the earlier observation that there is fixed amount of work per latitude for each decomposition that is also the same for all decompositions.

The code written to process the chemical sub-steps appears to have been written efficiently. The multi-dimensional loops are unrolled into one-dimensional loops over the number of boxes in the plane. This gives the compiler an easier job of optimising. The gains from inlining and vectorising (changing the optimisation from -O3 to -fast) will not be so great for those sections of code. Cache blocking may achieve improvement but that has not been investigated yet.

Another contributing factor to poor scaling is the communications. The imbalances indicated in a PAT sampling experiment show that there are occasions where there are some tasks idle awaiting the other tasks to synchronize. However it is difficult to balance this version of the code as the distributions of the species will require different calculations. Gains can be made by overlapping communications where any work needed to prepare for a communication can be off-loaded on to the communication sub-system (MPI layer). Examples of such instance are places where there is a send and receive exchange. Often this is parcelled into a single MPI_SENDRECV for simpler coding. However, the idea of non-blocking receive can be used to prepare a buffer to receive data from a neighbouring PE before the send actually occurs. This is another line of investigation that is yet to be analysed. Using the analysis of Appendix B it should be possible to design replacement communication points.

Figure 10 shows the sampling experiment for GM4 (essentially modified GM3) and gives a clear indication that the re-organisation of the CHIMIE subroutine has reduced the workload.

GM4 on 8 PEs (XT4 Dual Core) PAT sampling experiment report

```
 Samp % |  Samp |   Imb. |   Imb. |Group
        |       |   Samp | Samp % | Function
        |       |        |        |  PE='HIDE'
 100.0% | 73918 |    -- |    -- |Total
|-----------------------------------------
|  69.5% | 51363 |    -- |    -- |USER
||----------------------------------------
||  10.9% |  8077 |  73.50 |   1.0% |advy2_
||  10.3% |  7580 | 202.38 |   3.0% |ukca_coagwithnucl_
||   6.6% |  4882 |  35.88 |   0.8% |consom_
||   6.4% |  4713 |  11.88 |   0.3% |advz2_
||   5.4% |  4012 |  66.00 |   1.8% |advx2_
||   2.9% |  2131 |  53.50 |   2.8% |ukca_water_content_v_
||   2.5% |  1811 |  97.25 |   5.8% |chimie_
||   2.4% |  1779 |  56.62 |   3.5% |ukca_conden_
||   2.2% |  1611 |  52.88 |   3.6% |ukca_calc_coag_kernel_
||   1.9% |  1418 |  30.38 |   2.4% |ukca_aero_step_
||   1.7% |  1273 |  21.00 |   1.9% |emptin2_
||   1.7% |  1254 | 219.25 |  17.0% |initer_
||   1.5% |  1143 |  17.75 |   1.7% |radabs_
||   1.3% |   939 |  43.00 |   5.0% |ukca_ddepaer_incl_sedi_
||   1.2% |   917 | 170.75 |  17.9% |fillin2_
||   1.2% |   875 |  67.75 |   8.2% |update_1dvars_by_cstep_
||========================================
|  26.5% | 19590 |    -- |    -- |ETC
||----------------------------------------
||  11.1% |  8236 | 166.75 |   2.3% |__c_mzero8
||   3.6% |  2666 |  45.88 |   1.9% |__c_mcopy8
||   1.5% |  1093 | 421.00 |  31.8% |PtlEQPeek
||   1.3% |   937 |  44.50 |   5.2% |__fmth_i_dexp
||   1.0% |   729 |  41.38 |   6.1% |__fvdlog_long
||   1.0% |   715 |  61.25 |   9.0% |munmap
||========================================
|   4.0% |  2965 |    -- |    -- |MPI
||----------------------------------------
||   1.7% |  1223 | 573.88 |  36.5% |mpi_recv_
||   1.6% |  1165 | 246.50 |  20.0% |mpi_sendrecv_
|========================================
```

Figure 10: PAT report for an 8 PE run using GM4. NOTE new lower position for CHIMIE, i.e. 2.5% compared to 27.3% in figure 5a.

The outer loop over K that is the second index to three-dimensional arrays and there is possible further improvement available if those data structures were organised by planes of latitude. For example, swapping the second and third array indices would be achieved by changing the original code shown on the left to the code shown on the right:

```
DO 6 K=1,MYLAT
  DO L=1,NIV
    DO I=1,NLONMX
      JL  =I+(L-1)*NLONMX
      UG(JL)=U3D(I,K,L)
      VG (JL)=V3D(I,K,L)
      TG (JL)=T3D(I,K,L)
```

```
DO 6 K=1,MYLAT
 DO L=1,NIV
   DO I=1,NLONMX
     JL =JL+1
     UG      (JL)=U3D(I,L,K)
     VG      (JL)=V3D(I,L,K)
     TG      (JL)=T3D(I,L,K)
```

However, this would be a very significant change for a large proportion of code throughout TOMCAT and many array definitions would have to be revisited. It is unlikely that this optimisation will be done.

Conclusions from task 2.1

Performance analysis tools have been used to identify areas of code that incur a high workload. Several of these are in two specific subroutines (UKCA_COAGWITHNUCL and CHIMIE) and they were investigated and alternatives were tested. A different approach for the aerosol subsystem was adopted to improve the overall operation of the simulation. The improvement was demonstrated with GM4 as the loop size was significantly smaller because CHIMIE is dealing with planes of data rather than volumes of data.

Some of the identified areas could not be improved. In particular sections where the UKCA_COAGWITHNUCL function initialises arrays at the start of every chemical time step. This is most likely due to the shape of those arrays being 3,4 and 5 dimensional, even though the sizes are small compared to the total number of gid-boxes. It appears that the majority of work in this sub-progam is the initialisation process.

Task 2.2: Optimizations for Parallel Operation

Current parallel implementation

Parallel operation of computer programs raises several issues, one of which is efficient communication between the parallel tasks. A general methodology is to package up local data into local buffers and then pass it to other tasks through the communications layer. At the same time (or nearly simultaneously) the information that is being received from other tasks is stored in additional local buffers and it is transferred to the working arrays just before it is needed. The incoming data is referred to as halo data and the process is commonly known as "halo exchange".

A common implementation seen in this code is to cycle through the entire MPI task IDs on the same row of the topology (using a do loop) and testing if a communication is required.

For example in CALFLU:

```
C     Loop over PEs in row
      DO 91 J=0,NPROCI-1
C
C       Only calculate for PEs in this column
        IF (J.EQ.ICOL) THEN
C
C  b.   initialisation
C
          IF (ICOL.EQ.0) THEN
```

This is usually wasteful because the data locality means that there is a specific sub-set of PEs that will need to communicate with the task in question (its neighbours). A basic optimization is to pre-calculate which tasks will need to communicate and create a sorted list that can then be used to target the communications. This list can be used to bias the loading of the buffer and speed up the unpacking of the information as it arrives. Another optimization is to initiate the receive buffer earlier than when it is needed so that it is available to the other tasks for receiving information.

The strongly structured and unchanging nature of the domain decomposition used by this application assists in the pre-determination of the structures and lists. In fact, there is a section already coded where the references to neighbours are calculated and these are used in the main exchange communications in EXCHUV and DOCOMM2.

Other communications that are present in the GLOMAP code are the global broadcasts, where data on one task is needed by all of the other tasks. These are typically summations, or reduction, operations and MPI has built-in functions to support them. However, it appears that in some parts of GLOMAP there are custom versions of these that are achieved through combinations of send and receive of the data.

The section of the code that solves equations using an FFT method has a different domain decomposition from the rest of the TOMCAT code. Complete lines of latitude are shared

among the PEs the subroutines called by SPEGRD1 work with data structured for the FFT analysis. There is a time penalty for translating the data from one form to another and a memory penalty because of the storage required for the new working arrays.

The Cray PAT analysis provides information about selectable groups of functions. One useful group is that of "MPI" functions. The example sample reports show information about the MPI functions. The workload performing the MPI communications increases with the number of MPI tasks. In general this is expected and acceptable. As a problem is divided up between MPI tasks, it will become smaller when divided between more tasks. There will consequently be more interfaces, or boundaries, where the halo exchanges are needed, thus increasing the ratio of communication to individual task computation.

Summary of manual review of source code

A search of the source code for all MPI communications was carried out and several subroutines were identified as high workload using Cray PAT as well as containing MPI calls. There were several other subroutines and these may warrant investigation at a later stage but those listed here are the major contributors to the computational workload.

A more detailed analysis of each subroutine is presented in Appendix B. the following points list the areas for improving parallel activity, the affected routines and explains the method applied for improving the overall runtime.

Changes for Setting up the Run

SETMPP additional structures were introduced for grouping PEs, creating MPI communicators for the new groups and the creation of the processor topology was moved into this routine from the main program. This routine was moved from INIEXP into the main program.

FFTSETUP This routine was moved from SPEGRD into the main program. It was wasteful to repeat it as it is invariant during the run. Subsequent discussion with the authors suggests this should be moved into CALPHY.

Changes for Communication Pattern

CALFLU, CALSUB These subroutines contain send and receive pairing method that forces a cascade effect on the communications. It was replaced by an alternative oscillatory method that is more efficient (requires less time with PEs waiting to receive/send).

Replace FILLIN2 and EMPTIN2 with Direction-Specific Buffer Packing and Unpacking

DOCOMM2 controls the pattern of the main communication between the uni-directional advection sweeps. It contains a short loop that cycles for two iterations around the following routines:

FILLIN2 This routine is called several times for different variables and contains a conditional statement that decided the manner for copying data from the local data arrays into the communication buffer.

LISTCOMM This routine decides which processor ID in the topology is going to receive the data and initiates the exchange.

EMPTIN2 This routine is called a number of times corresponding to the calls to the FILLIN2 subroutine. It also has a conditional statement that determines how the data is copied out of the communication buffer into local data storage.

The conditional test in EMPTIN2 is at a low level and hence is unnecessarily repeated and introduces inefficiency. The loop was unrolled and the FILLIN2 subroutine was replaced by a call to each of LOAD_NS and LOAD_WE which avoided the need for any test. Similarly EMPTIN2 was replaced by a call to each of UNLOAD_NS and UNLOAD_WE.

EXCHUV routine is similar to DOCOMM2 but has more work to do with velocity calculations. The same modifications as for DOCOMM2 were applied: unroll the loop and replace FILLIN2 and EMPTIN2.

Enhance Buffer Loading and Unloading

GATHERROW, SCATTERROW load and unload communication buffers for the FFT domain decomposition for the PBL scheme (different to the main domain decomposition used by the model). The process of reviewing source code revealed an inefficient loop ordering (which was arbitrarily implemented). The loops I and L were swapped.

Change for Group Reduction Operation

SPETRU1 and **POLCOMM** contain send and receive functions that emulate a global reduction operation. These were replaced by the appropriate MPI reduction functions.

Result of the changes to communication sections

The two versions, baseline GM4 and enhanced GM4 were compared on the new quad-core system, the results are summarised in table 12. The enhanced GM4 is faster than the standard GM4 by 2.56 % on16 PEs, 9.44 % on 32 PEs and 7.61 % on 64 PEs.

The following data (table 7 and table 8) is the result of early experiments of modification of the communication pattern. Table 7 gives the baseline data that is used for comparing improvements gained from compiler options and code restructuring. The reference “corrected GM4” is to the modified X2 library of TOMCAT that has the changes introduced for successful X2 operation. That library is suitable for use on the XT platform also as the changes are in fact improvements to the robustness of the code as revealed during the X2 porting exercise. There are no Cray X2 specific features in the code. The 128 PE run is included to show that the modified software continues to scale at a better rate than is possible with the exisitng production code.

NCPU	8	16	32	64
Tinit	33.74	31.2	36.98	47.81
T143	1231.499	710.2	424.32	294.99
T144	1244.23	720.6	435.24	309.52
Ttotal	1251.98	730.6	449.5	332.91
Ttot-T143	20.481	20.4	25.18	37.92
T143-Tinit	1197.759	679	387.34	247.18
(T143-Tinit)/142	8.43	4.78	2.72	1.74

Table 7: Timings for GM4 using several domain decompositions standard corrected GM4, dual-core Opteron (2.8GHz), PGI 8.0.2, MPT 3.1, fully populated (128PE run not performed).

NCPU	8	16	32	64	128
Tinit	32.74	30.62	33	42.28	66.17
T143	1180.93	656.1	372.6	264.3	261.8
T144	1192.79	665.5	382.3	277.8	284.4
Ttotal	1199.47	674.4	395.4	299.1	321.9
T143-Tinit	1148.19	625.48	372.6	222.02	195.63
(T143-Tini)/142	8.08	4.40	2.62	1.56	1.37
Percentage change from table 7	-4.1%	-7.9%	-3.6%	-10.3%	NA

Table 8: Timings for GM4 with modified communications pattern Enhanced GM4, dual-core, Opteron, Cray OS 2.1 defaults PGI 8.0.2, MPT 3.1, fully populated

The sampling experiments were performed for the 64 PE case for the baseline GM4 and for modified GM4. The two routines EMPTIN2 and FILLIN2 are no longer visible as they have been replaced by four more efficient subroutines that are not visible either. A reduction in time spent on MPI calls is not as great as anticipated as there is a new contribution from the "SETMPP" routine in the form of MPI_CART_CREATE. The overall profile of the USER subroutines is not significantly changed although the imbalance for "CONSOM" is less. A conditional statement was removed from within a do-loop of that subroutine.

```
GM4 on 64 PEs before modifications for communication:
 Samp % |  Samp |    Imb. |   Imb. |Group
        |       |    Samp | Samp % | Function
        |       |         |        |  PE='HIDE'
 100.0% | 23849 |      -- |     -- |Total
|-----------------------------------------------
|  36.8% |  8781 |      -- |     -- |MPI
||----------------------------------------------
||  18.2% |  4334 | 2548.84 |  37.6% |mpi_recv_
||  14.0% |  3347 | 2818.92 |  46.4% |mpi_sendrecv_
||   3.9% |   941 |  831.44 |  47.7% |mpi_ssend_
||==============================================
|  33.8% |  8065 |      -- |     -- |USER
||----------------------------------------------
||   5.7% |  1365 |  142.03 |   9.6% |advy2_
||   3.2% |   756 |  204.92 |  21.7% |consom_
||   3.1% |   737 |   95.91 |  11.7% |ukca_coagwithnucl_
||   2.7% |   637 |   23.19 |   3.6% |advz2_
||   2.2% |   522 |  268.17 |  34.5% |advx2_
||   1.7% |   416 |  288.33 |  41.6% |emptin2_
||   1.4% |   343 |  262.20 |  44.0% |fillin2_
||   1.2% |   283 |   69.45 |  20.0% |chimie_
||   1.0% |   237 |   17.14 |   6.9% |calflu_
||   1.0% |   229 |   51.72 |  18.7% |ukca_coag_coff_v_
||==============================================
|  29.4% |  7003 |      -- |     -- |ETC
||----------------------------------------------
||   5.6% |  1347 |  235.38 |  15.1% |__c_mzero8
||   5.5% |  1309 |  337.02 |  20.8% |PtlEQPeek
||   2.9% |   684 |  204.52 |  23.4% |fast_nal_poll
||   2.2% |   527 |  200.42 |  28.0% |PtlEQGet
||   1.6% |   390 |  104.27 |  21.4% |PtlEQGet_internal
||   1.5% |   362 |  188.81 |  34.8%
|MPIDI_CRAY_smpdev_progress
||   1.4% |   344 |   65.81 |  16.3% |__c_mcopy8
||   1.1% |   273 |   32.80 |  10.9% |__fmth_i_dexp_gh
||   1.1% |   256 |   85.16 |  25.4% |ptl_hndl2nal
|==============================================
```

Figure 11a: PAT report for the 64 PE run using GM4, it is prior to modification to improve communications.

The MPI section in table 11a features high up the table with more than a third of the run time spent in communications. The ETC section is showing a large percentage as this represents the work of the system including “portals” the underlying communication layer beneath MPI.

```
GM4 on 64 PEs after modifications for communication:
 Samp % |  Samp |    Imb. |   Imb. |Group
        |       |    Samp | Samp % | Function
        |       |         |        |  PE='HIDE'
 100.0% | 26158 |      -- |     -- |Total
|-----------------------------------------------
|  34.3% |  8971 |      -- |     -- |MPI
||----------------------------------------------
||  15.0% |  3927 | 2316.62 |  37.7% |mpi_recv_
||  11.7% |  3069 | 2339.02 |  43.9% |mpi_sendrecv_
||   3.5% |   910 |  767.52 |  46.5% |mpi_ssend_
||   3.1% |   812 |  773.25 |  49.6% |mpi_cart_create_
||==============================================
|  33.5% |  8754 |      -- |     -- |ETC
||----------------------------------------------
||   7.1% |  1855 |  333.95 |  15.5% |PtlEQPeek
||   5.5% |  1440 |  204.62 |  12.6% |__c_mzero8
||   3.8% |   984 |  212.80 |  18.1% |fast_nal_poll
||   3.0% |   772 |  181.59 |  19.3% |PtlEQGet
||   2.2% |   571 |  149.44 |  21.1% |PtlEQGet_internal
||   2.0% |   515 |  139.47 |  21.7%
|MPIDI_CRAY_smpdev_progress
||   1.4% |   367 |   43.12 |  10.7% |__c_mcopy8
||   1.4% |   365 |  106.11 |  22.9% |ptl_hndl2nal
||==============================================
|  32.2% |  8433 |      -- |     -- |USER
||----------------------------------------------
||   5.0% |  1298 |  193.70 |  13.2% |advy2_
||   4.9% |  1290 |   19.72 |   1.5% |consom_
||   2.7% |   708 |  105.41 |  13.2% |ukca_coagwithnucl_
||   2.5% |   660 |  162.69 |  20.1% |advx2_
||   2.4% |   620 |   23.67 |   3.7% |advz2_
||   1.2% |   314 |   36.11 |  10.5% |chimie_
|==============================================
```

Figure 11b: PAT report for an 64 PE run using modified GM4. The enhanced communications have increased the relative workload of the “ETC section”. CONSOM appears better balanced and EMPTIN2 and FILLIN2 have been rmoved.

Even though the percentage of the run for MPI time has been reduced it appears to have increased for ETC. this is likely due to more work being done by the system layer. The USER section has been reduced through the steam-lining of buffer loading. The buffer sizes have not been changed so there is the same amount of work being done by the ETC and MPI per iteration.

The primary focus for producing figure 12 is to show the improvements achieved from modifying the commnications it is also possible to show the effect of using only one core per node discussed in detail later in this section. Note that the scales are spaced as per $\log_2$ of the value to emphasise the effects of scaling the speed of the test case.

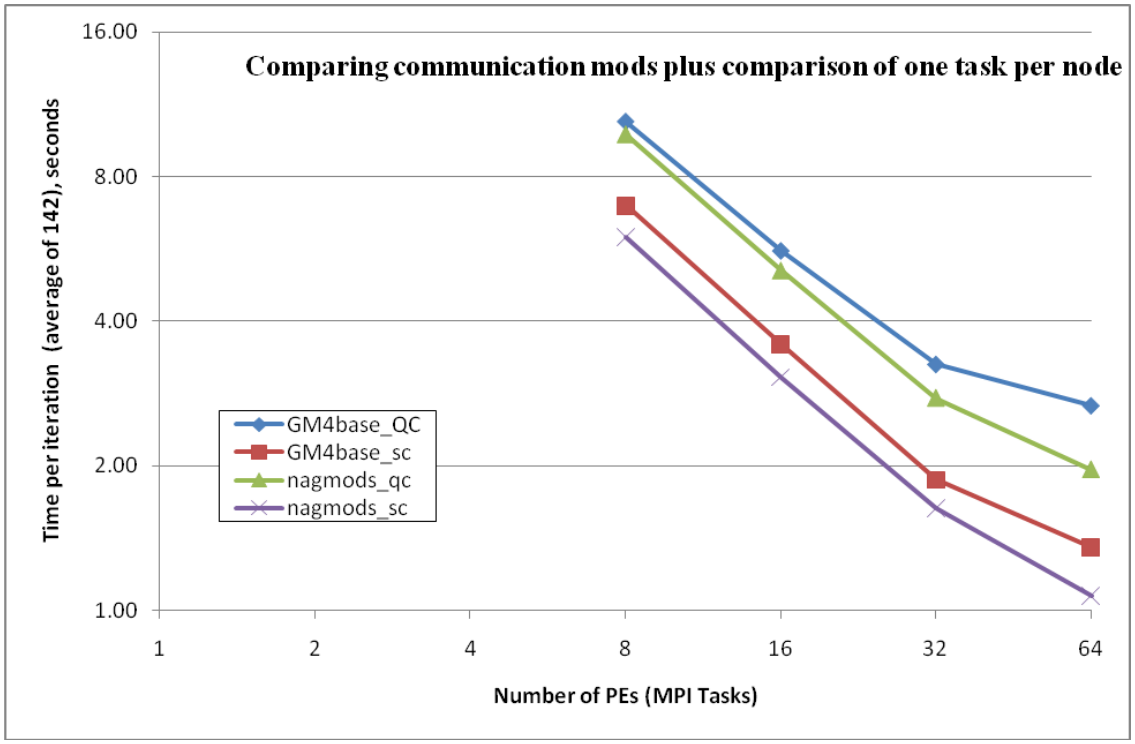


Figure 12: Comparison of performance improvement of GM4 due to communication pattern modifications. The triangle and diamond marked lines are for quad core operation (i.e. fully populated nodes). The cross and square marked lines are for the case where only one core per node is used.

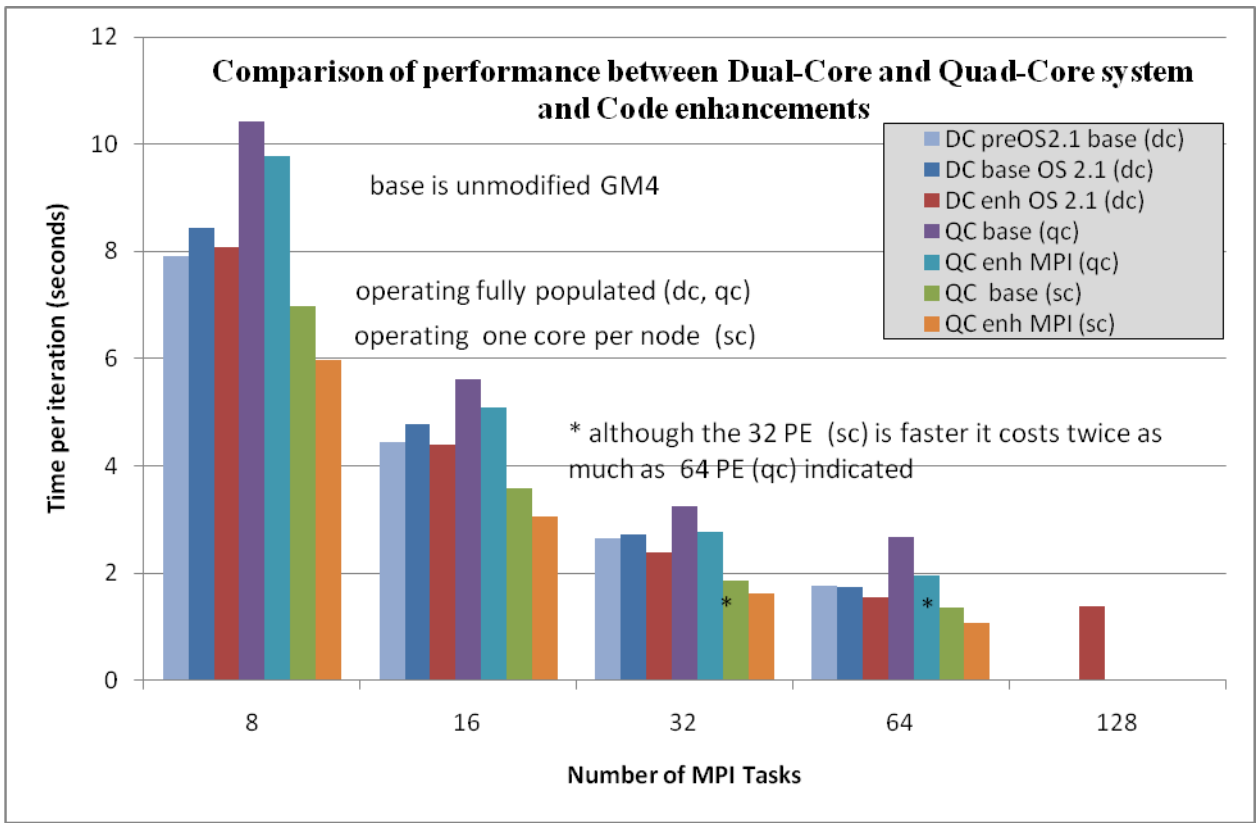


Figure 13: Comparison of performance improvement from modifications to GM4 discussed in the text body below.

Several parameters that affect the performance of GM4 are compared on the single barchart (figure 13) if considered in a chronological manner these are:

- The change from OS 2.0 to OS 2.1 using a baseline code for comparison
- The improvement between a partially modified communication pattern and the baseline
- The difference between the dual core system and the quad core system when all cores are working on the problem
- The difference between operating with four cores per node and one core per node on the quad-core system

The difference between the first two bars of each group is less for a higher task count. It is the difference between one version of the OS (2.0.56) and the newer OS (2.1) required prior to quad-core upgrade. This "slow down" has never been explained and several research groups in very different subject areas had observed this for different applications. It is unlikely that an explanation is forthcoming as the system has since been upgraded to quad-core which is the reason for the interim OS upgrade.

The difference between the second and third bars in each group is the improvement achieved with the partial enhancement of communications prior to the upgrade of the system to quad-core. Those results were encouragement to continue the work and extend the changes as described the Appendix B.

The significant increase in length between the second (dark-blue) and fourth (purple) bars per group indicates that the move to quad-cores has resulted in a large drop in model performance. This is caused by a combination of the lower clock speed and lower memory-per-core on the replacement quad core nodes compared to the original dual-core nodes. The main differences between quad-core and dual-core are shown in table 9.

Platform	**Dual core**	**Quad core**
L2 cache	1 MB/core	512 KB/core
L3 cache	None	2 MB (shared)
Main RAM	6 GB (~3GB/core)	8 GB (~2GB/core)
Clock speed	2.8 GHz	2.3 GHz

Table 9: Notable differences between dual core and quad core processors.

The memory issue is made clearer when running a single MPI task per node (the sixth and seventh bars per group for the modified GM4). The latter situation is four times more expensive (based on original cost model) than the operation with fully populated nodes but is taking just over half the time per step i.e. approximately twice the cost of running fully populated nodes.

This is an indication of the benefit of allowing more memory for the application and making the whole of the L3 cache available. An alternative view is that there are significant gains to be made if more effort is put into the better use of memory and tuning the code to make more effective use of the cache when operating with an instance of the simulation on all cores of the node.

The changes to the communications provide further gains making the cost appear only twice as much for using one quarter of the available resource. This result indicates that mixed mode programming soon to be carried out as part of the follow-up GLOMAP dCSE project is likely to give reasonable improvements. Here the planned configuration will have one MPI task per quad-core node, each with four Open MP threads.

There is still some work to be done to improve the scaling to 64 MPI tasks. Factors than can be addressed include overlapping communication and computation, reducing memory requirement per MPI task and rationalising the global communications per step.

An alternative presentation for the data in figure 13 is a speed-up curve and is shown in figure 14. The data is not continuous across PEs but this sort of data is commonly presented as a curve. Figure 14 demonstrates the scaling compared to 8 PEs. The ideal line is for guidance. For example, a simulation using 64 PEs might be expected to run eight times faster than the simulation using eight PEs.

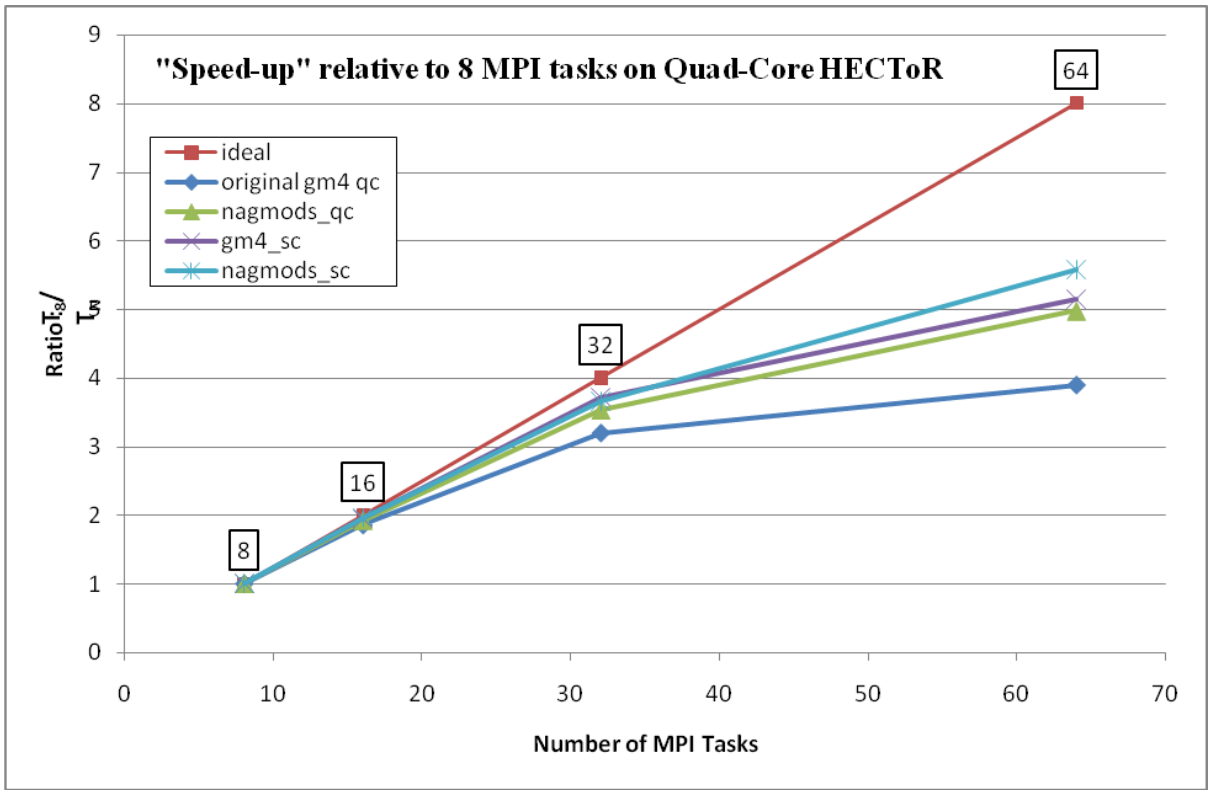


Figure 14: Performance on quad-core nodes (_qc), including under-utilisation, i.e. one core per node (_sc). The enhanced GM4 is indicated by “nagmods”.

Task 2.3: Parallel File Handling, improving reading and writing of data

Overview

The current challenge for software that has been adapted for use on parallel machines is mainly the interaction with file systems. This is where and how the program stores and retrieves data at various stages of the simulation. Generally, codes will more often be started from data generated by a previous run (restart) or need data that controls the progress of the simulation. During the course of the program execution it is likely that data will be written for later analysis or to report on the status of the run. With more powerful parallel systems becoming available scientists are increasing the complexity of their research that leads to an increase in the quantity of data being processed.

Traditionally programmers have used one of the following models to access the files:

a) All MPI tasks read and write their own data to separate files.

This is suited to distributed parallel systems with individual disk systems per processor group (also could be a cluster of SMP systems). The time to write each file is shorter than for the equivalent serial run. It is generally scalable, although this depends on the underlying file system.

The disadvantage is that there have to be ancillary helper applications that separate data files into sizes to match the number of MPI Tasks and helper applications that combine the files into a coherent single entity for post-processing. Coordinating the management of a large number of files can be problematic.

b) An I/O Master is nominated (usually MPI rank 0)

All other MPI tasks transfer their data to the master and it is subsequently written to file (conversely: the master task reads a file and distributes it among all the tasks).

This is often the easiest to manage as the data files remain as they would be arranged for an equivalent serial run. The variation of availability of PEs will not affect the data files and there is no need for extra processing to convert the files prior to or after the simulation. It is portable and will work on any parallel system.

The disadvantage is that the I/O Master will have to store the global data ahead of the write or after a read. Another (perhaps more important?) disadvantage is that the other processors will need to wait for the master to finish reading/writing the data and communicate to them before they can continue with their work. This will show up as detrimental to scaling if the program requires frequent I/O.

c) The I/O Master is the method used by GLOMAP.

All tasks read and write their data to shared files but using MPI specific data structures.

This method does not need any “decomposition or combination” ancillary programs. Potentially it is as fast as (a) but requires the underlying file system to be aware that it is in a

parallel computing environment. There may be limits to the number of tasks allowed to access an individual file and thus limit the scaling to worse than that which could be achieved using (a). A lot of effort has to be expended to design and implement the new data structures required to access the parallel file systems. Each data structure that is written to file has to have a matching data type that describes the pattern of the data within the file.

d) Sub-group for I/O: a sub-group of tasks use the MPI-IO file manipulators with MPI data structures.

This avoids saturating the underlying file system but could be complicated to implement. It is a compromise of (c) and (b) as it requires the parallel data structures to be defined and extra work of determining which tasks are recruited for file handling. While the tasks are doing data handling they cannot contribute to the compute and it is likely that other tasks are idle waiting for the file tasks to complete. There is a one-off set-up penalty during the definition of the parallel topology. Choosing the size of the group may be critical to the improvement over (c). There is a requirement for communication between the I/O group and other tasks. It is possible to design a system that will cope with the imbalance but not within the GLOMAP model.

Analysis of file accesses in case study with existing version

There are 86 files copied into the case directory during the job preparations. Two extra files are generated during the run (fort.9 and fort.31). There are nine symbolic links to other directories created at the outset, some of these are used as shorthand to copy files into the local directory. It is not easy to determine how many files are accessed during a normal simulation as this project uses only the first three days of a run. Some of the data is for updating TOMCAT fields (winds, surface values etc…) other data is for the GLOMAP function with emissions and concentration data.

There are several places where a file has to be read or written. Those occurrences are at set intervals and they are protected by master MPI task tests. During initialisation they are only accessed by the MASTER I/O TASK and it has to store the full extent of the data. As the mode of operation of the whole parallel code is SPMD with static array allocation, all TASKS have declared enough storage space for the global data. A final gather operation is required for writing coherent files such as fort.9, fort.25 or files that are processed elsewhere and by sequential programs.

The distributed memory model of parallel operation implemented in GLOMAP requires that after the master processor (task id 0) has read *all global* data from a file it distributes the content to each task. Some data is universal and the communication is in the form of a broadcast of a large array, which is itself a copy of the global array that stores the information on the master process. Each task then extracts the pieces of information that are to be used on its own process. This is unnecessary and causes two undesirable effects: the memory footprint per process is larger than it needs to be (even the master task memory requirement could be reduced) and there is a high quantity of cross communications (all-to-all) saturating the communication network. It can be replaced with either MPI-IO or alternative scatter

operation with a ROW-MASTER I/O model. In some cases the Master I/O task will broadcast all the data to all other tasks and these then extract their own component. This is only efficient in the fact that the Master I/O does not have to extract each task data and send individually. However, it will saturate the communications layer and thus provide a possible bottle neck.

Another mode of file access is through calls to either PPWRIT or PPREAD. PPREAD reads all the chosen data from a file. It then distributes the data to the other PEs via several calls to MPE_SSEND. It has an extra complication in that data can be structured with halo information (such as the S_{xx} moment array) and sometimes the data has no halo regions. A conditional test is made at a low level to decide how to reference the information during the scattering to other PEs.

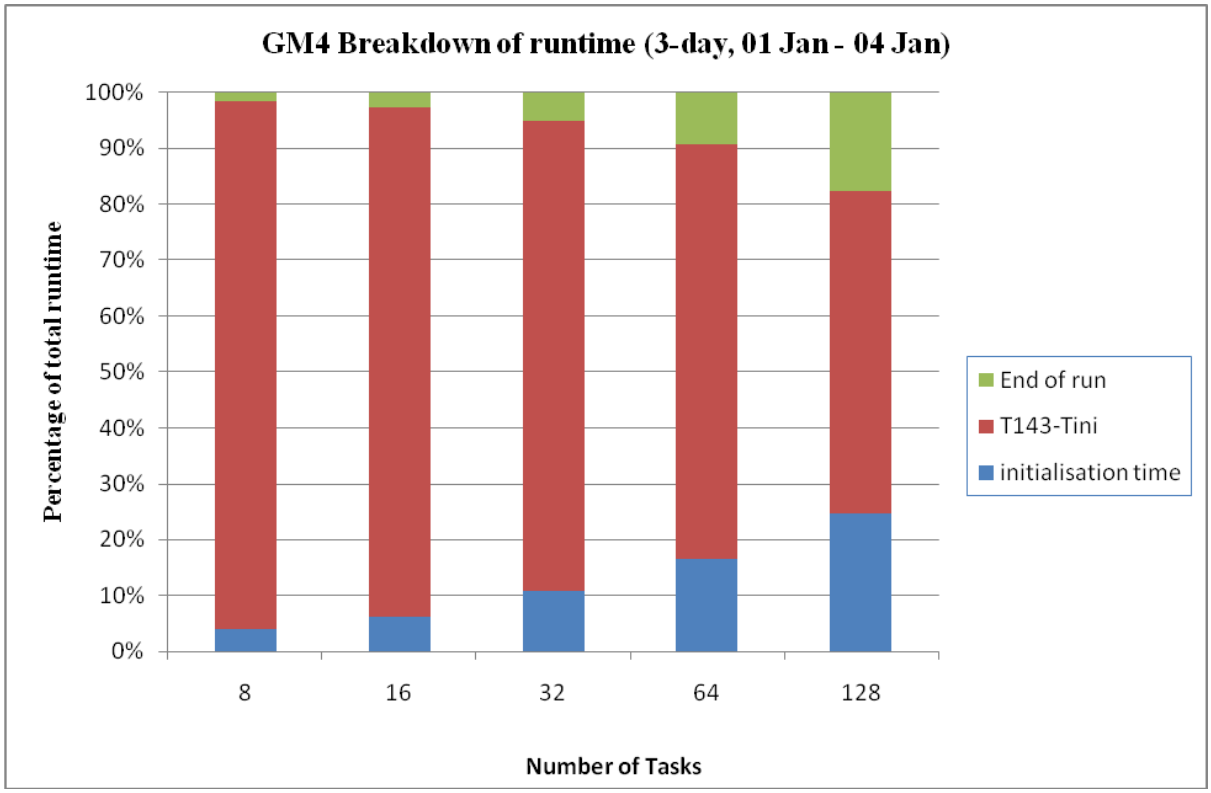


Figure 15: Time for first and final steps shown as a percentage of the full (3-day) simulation.

The timing reveals that the first and final time steps have significant work that is taken up with a Master I/O operation. Varying the number of MPI tasks shows the situation worsening for a larger number of tasks. This is shown in figure 15 where the time for each of the three sections has been identified as a percentage of the full run. It has been noted that the duration of this simulation is only 3 days and this should be taken into account. The standard working practice of simulating 30 days per job would mean a smaller percentage of the runtime will be needed for I/O. A certain amount of the main iterations is also given up to I/O but that has not been investigated in detail in this study. The initial observations indicate that the 6-hourly read for concentrations doubles the time for a time step. This adds up to become $1/13^{th}$ of the inner compute time.

Recommendations to improve file access

To benefit from the capability of the LUSTRE file system on HECToR, it is better to make the reading of data files a parallel job. The preferred route is to use MPI-IO and F90 data structures to allow each MPI-task to read its own section of the input files as they are predominantly read-only. If MPI-IO is unworkable (i.e. if data structures become too complicated or unmanageable) then a fallback position is to use a row-oriented method to read the file using a subset of the MPI tasks and then scatter fragments among the other MPI-tasks that share the same latitudes. Additions to SETMPP are required to declare additional file handles, PE groups and communicators.

Implementation of file data structures

To support the use of MPI-IO there have to be declarations of new data structures, possibly one for each data array written to or read from, a file. These are descriptions of the data and the way that they are stored on hard disc. For example, additional structures for file handling will be MPI data types:

Data that has no halo: data2d_nh_t, data3d_nh_t

Data that has halo information: data2d_wh_t, data3d_wh_t

Communicators and arrays: COMM_PIO will identify the group of processes involved in parallel I/O (actually in first implementation this will be equal to MPI_COMM_WORLD).

The format of the files is clearly defined. Categorising the files will allow specific strategies to be implemented for each category of file. The simplest example is for data that is volume information organised as (longitude by latitude by altitude). Adding communication specific information will allow the removal of some duplication of global data. The trend of the start-up timings will reverse as there is less data per processor.

Categories of files based on their data content: high resolution data (1x1), T42 resolution data (without halo data), T42 resolution data (with halo data), arbitrary data for FFT coefficients. Some data is arranged as plane of longitude by latitude at specific altitudes. Some data is arranged as planes of longitude by altitude at specific latitudes (usually North Pole to South Pole). Each of these categories will need a defined data type to allow parallel access.

Expected results of file handling modifications

The implementation was not completed and thus cannot be tested. However, the timing of the model runs on 32 and 64 PE numbers shows distinct spikes in time-per-time step every 12 time steps (6 hours) where the input meteorological fields are read in from file, some of this is actually due to the processing of the “new” data. However, it is expected that the implementation of parallel I/O will result in significant improvements in run-time on high PE numbers, improving performance and scalability. The start-up costs associated with reading of initialization/restart data files will also be reduced, as will the round-up at the end-of-simulation.

Overall Summary

The GLOMAP/TOMCAT code has been analysed for a specific case of GLOMAP mode MPI and modifications implemented that improve the serial performance. Analysis of the algorithms used for parallel operation has been used to improve the pattern of communication and achieve small gains (the Cray MPI does a reasonable job with the original version). These changes will all translate onto other platforms e.g. "Everest". This work has allowed the use of 64 PEs in a more efficient manner than previously was possible.

An overall summary of changes in runtimes (neglecting first and final step i.e. totals for 142 iterations) is given in tables 10, 11, 12 and 13.

Number of MPI Tasks	**8**	**16**	**32**	**64**
GM4 (QC -O3)	1485	783	449	334
GM4 (QC -fast)	1387	742	434	302
Improvement %	*6.60*	*5.24*	*3.34*	*9.58*

Table 10: Time in seconds showing improvement due to change of compilation option (the difference between -O3 and -fast).

These timings are from tests using four cores per quad-core node and show an improvement for all domain decompositions. The smallest change seen is for 32 PEs. This shows that the current working practice is using an optimal setting that matches the memory usage.

Number of MPI Tasks	**8**	**16**	**32**	**64**
GM3 (DC -O3)	1952	872	451	276
GM4 (DC -O3)	1122	631	377	251
Improvement %	*42.48*	*27.62*	*16.26*	*9.08*
Time in seconds for simulation omitting first and final steps				

Table 11: Improvement due to changes in the code structure.

The numbers are extracted from the work done using two cores per dual-core node. Changes to the code structure have given the largest benefits for a lower number of MPI Tasks. This can be attributed to how well the arrays match the local cache memory management.

Number of MPI Tasks	**8**	**16**	**32**	**64**
GM4 (QC -fast)	1387	742	434	302
GM4 (QC -fast) with MPI enhancement	1389	723	393	279
Improvement over GM4 baseline %	*-0.14*	*2.56*	*9.44*	*7.61*
Time in seconds for simulation omitting first and final steps				

Table 12: Improvement due to changes to the code structure around communication.

The values in table 12 are from the quad-core system with the optimisation level set to "-fast" in all cases. The small gain in performance for a low number of MPI tasks is due to the fewer numbers of communication connections.

Number of MPI Tasks	8	16	32	64
GM4 (QC -O3)	1485	783	449	334
GM4 (QC -fast) MPI enhancement	1389	723	393	279
Improvement over GM4 baseline %	*6.46*	*7.66*	*12.47*	*16.47*
Time in seconds for simulation omitting first and final steps				

Table 13: Shows an overall improvement comparing current production version (GM4 with -O3) with changes for both the compiler optimisation and communications.

The overall improvement is not a simple accumulation as each of the structural changes will be affected by optimisation.

Overall conclusions

Changing the compiler optimisation options and modifying the code structure in key areas has improved the performance of serial parts of the code. Unfortunately, these enhancements have made the scaling performance appear worse even though the total run time is less.

Revising the sections of the code that communicate data between PEs has resulted in shorter runtimes for all PE numbers. Thus, the performance of the code in terms of overall speed has been improved. The scaling to 64 MPI tasks has been improved.

The change from dual core to quad core has impacted the performance and makes further optimisation necessary. The difference between a run using four cores per node and a run using one core per node shows that there is still considerable work on memory use that has not been addressed.

The use of Cray PAT has highlighted sections of the program that incur high computational costs. A review of those sections by the code authors and developers may reveal that they can be replaced with newer techniques that alleviate the burden.

Future work planned

The current project has highlighted benefits of revised code structure and indicated the way to improved file handling. It has also highlighted that GLOMAP will benefit from the availability of Multiple Core processors if it can make the use of Open MP. Since the code was still recently used in pure Open MP and the OMP statements still exist in the MPI version of TOMCAT, it should be straightforward to implement this for significant performance gains on quad-core HECToR. The gains will be proportional to the amount of code that can utilise Open MP.

For example, a single MPI task per node will improve the performance over four tasks sharing 8GB of memory. That task can then create three extra threads for Open MP processing. The success will be limited by the percentage of the simulation that can operate in parallel as there are places where it is inherently serial. The improvement of running a single task per node has shown an improvement of runtime. Combining this with an Open MP implementation may improve the performance to better that of a solely MPI job on fully populated cores.

It has been shown that the communications bottle-neck begins to make itself apparent with 32PEs. Many of the changes made during this project have improved that a small amount with the bottle-neck being shifted to 64 PEs. This was demonstrated in the experiments where only one core per processor was used. Moving to mixed mode will relieve this pattern as within a node there will be multiple threads thus an increase in throughput will be achievable for 128 cores where previously none would benefit. With the improvements for 64 PEs and single core operation then a mixed-mode enable code will potentially give good return for 256 PEs.

The implementation into TOMCAT/GLOMAP of mixed mode Open MP-MPI in this way will be done as part of the 2nd GLOMAP dCSE project which has 4 months funding and will begin in August 2009.

Direction beyond immediate planned work

These recommendations for work beyond the planned projects are aimed at improving the usability and scalability of the application and they include:

- Change the infrastructure to conform to the policies of administrators of HECToR separating out the sequential processes from the parallel jobs.
- Change the method for reading reference files with MPI-IO. The initialization and finalization of each job become more significant as the job is shortened, the current working practice with one-hour runs means that these sections can be as much as 1/30th of the simulation time (~3%).
- Reduce the number of broadcasts of global data. This needs a review of all subroutines where MPE_BCAST is used. Although the profiling did indicate that not a lot of time was spent in the broadcast routines there was some imbalance reported that suggests some "wait-time".
- Reduce the amount of memory that is primarily for the Master I/O model of file handling. This can be achieved by implementing a sensible parallel I/O strategy.
- Make better use of the existing MPI data structures and the regular nature of the discretisation of the computational domain. Approximately half of this project has been analyzing MPI structures with a view to enhancing the parallel performance. Not all sections were completely revised and there is still scope for performance gains.
- Time should be set aside with future projects to review the compilation options and compiler version when new compilers become available. Generally being aware that a new version of the compiler is available may offer improvements in performance.
- Allow for time required for fixing the 64 CPU run on Cray X2, find the bug as it could be beneficial to the XT4 work. Using different compilers and architectures is a useful method for tracking down inconsistencies in the code structures.
- There many other places where the sequential sections of code can be improved. Particularly "in-lining" will be of benefit in several places. However, it should be implemented in a prescribed manner through compiler directives. The work in this area was hindered by a compiler bug.

Appendices

Appendix A: Script examples for DCSE working practice and list of coding errors discovered and fixed

Appendix B: Detail of MPI communications used for 12 subroutines

Appendix C: Example MPI-IO design

Appendix A: Examples of scripts used during DCSE work

The working practice has been described in detail in the main report. It consists of five stages: Create case directory; Create source code; Build executable; Launch simulation; Process result of run. Repeated building of the executable is required when investigating optimisation or using the Cray PAT facility, consequently there is a need for a Makefile and separate submission script to launch the simulation with the new executable.

Example Makefile

```
#
# XT4h Fortran compiler generic name
FC = ftn
# options for pgf90
###DBGFLG= -v -V -g -O0
DBGFLG=
INFO=-Minfo -Mneginfo
###-Wl,-Map,xtgmm.map
#
# investigate compiler options for optimisation
###OPTIM= -Mipa=reshape -Minline -fast -O3
###OPTIM= -Mipa=inline -fast -O3
###OPTIM= -Minline -fast -O3
OPTIM= -fast
###OPTIM= -O3
#
###FFLAGS= $(DBGFLG) $(INFO) -Mextend -byteswapio -r8 $(OPTIM)
FFLAGS= -Mextend -byteswapio -r8 $(OPTIM)

.SUFFIXES: .f90 .f .o

GLOMAP.f90: GLOMAP.f
        ln -s GLOMAP.f GLOMAP.f90

modules.f90: modules.f
        ln -s modules.f modules.f90

GLOMAP.o: GLOMAP.f90 modules.o
        $(FC) $(FFLAGS) -c $<

modules.o:  modules.f90
        $(FC) $(FFLAGS) -c $<

prog.o: prog.f modules.o
        $(FC) $(FFLAGS) -c $<

GM_OBJS=modules.o GLOMAP.o rdsp.o

xtgmm: $(GM_OBJS)  prog.o
        $(FC) $(FFLAGS) -o xtgmm.exe prog.o $(GM_OBJS)

# standalone post processor (Sequential)
pdgc: pdgc.f
        $(FC) -mcmodel=medium -byteswapio $(OPTIM) -o pdgc.exe
pdgc.f

clean:
        rm -f prog.o $(GM_OBJS) *.mod
```

A generic Makefile for the code as supplied was created. This one assumes that prog.f, GLOMAP.f90, modules.f90 and rdsp.f are available. They are usually generated during the job submission in current working practice. This Makefile is for PGI FORTRAN.

The hashes are comments that disable certain lines and there is a tab character on the line following the target dependency line. On any Unix system "man make" will provide further information about its operation.

The DCSE used this Makefile for simpler recompilation during testing of compiler options and Cray PAT analysis. The different options required for building an executable with Cray FORTRAN were implemented using a separate Makefile. They are both similar in layout with the main difference in the FFLAGS value. The XT make is invoked as "make –f make.pgf95 xtgmm" and the Cray X2 compiler is invoked using "make -f make.cftn x2gmm".

PBS job script

The job submission scripts are specific to the number of MPI tasks and also whether the simulation is a standard run (for checking performance), an instrumented run (using Cray PAT facilities) or a debugging exercise. Variation in the parallel density is allowed for in the "-l *mppnppn"* option to qsub. The command to use it is: "qsub GLOMAP64c4.pbs"

```
Example PBS script (GLOMAP64c4.pbs)

#!/bin/ksh
#PBS -N gm4_O3_p64c4
#PBS -l mppwidth=64
#PBS -l mppnppn=4
#PBS -l walltime=0:19:00
#PBS -A n02-chem
#PBS -m e

cd $PBS_O_WORKDIR
export NTASK=`qstat -f $PBS_JOBID | awk '/mppwidth/ {print
$3}'`
export NPPN=`qstat -f $PBS_JOBID | awk '/mppnppn/ {print
$3}'`

date > StartedJob.$$

aprun  -n $NTASK -N $NPPN
${GM4HOME}/SCAL_OS2.1/src_64/GLOMAP.exe

mv fort.9 gm4_O3p64.f09

qsub convert_pdg.pbs

# end of script
```

This script is part of a chain for evaluating the scaling of simulation and is initiated from within the preceding PBS script after the aprun has completed. In this (the penultimate script) there is a submission of a serial job to convert the fort.9 files from double to single precision.

List of coding errors discovered and fixed

The difference between PGF95 and Cray ftn 6.0.0.2 revealed some problems with the code. Some of these were straightforward revealed during the compilation. Additional compilation with the NAG FORTRAN compiler exposed some coding mistakes (i.e. other compilers did not flag an error). Others were highlighted during runtime (with a failure of the code). The following list was discovered during the port to the Cray X2 and subsequently fixed in the central reference library by the code owner. The reference library for use with this project include *UNICATMPP0.91* and *UNIASAD0.2a*. Subsequent corrected libraries were porvided an distinguished by the name: *UNICATMPP0.91X2* and *UNIASAD0.2aX2*.

- *Missing calculation of Cosine in southern hemisphere.*
- *Incorrect index for innermost dimension of multi-dimensional arrays within PBL FFT calculations (SPEGRD1 and SPETRU1). NLON+3 was used where NLON+2 was required.*
- *VDEPH is not initialised when PBLCCM was not being used (IDRY and IVDEP)*
- *DMSCONCGB was uninitialized or indeterminate for time steps between the reading of the reference data from file (this is in emisions.f90 an update file for the main code).*
- *Alignment of data in common blocks mixing real and integer. For this code reals are promoted to double with a compiler option (-r8 for PGf95 ad –s real64 for cftn) thus causing a risk of data corruption.*
- *AERFIELDS4I was introduced to carry MOIS_DMS separate from AERFIELDS4*
- *CMCCTL_R for PEPS and CTS in the ASAD subsystem*
- *DTM moved from COMI to COMR*
- *Several inconsistent parameters in subroutine calling statements were of different type to the implementation. (Real data type instead of integer). For example:*
- *MPE_BCAST was used instead of MPE_BCASTR in many places*
- *MPE_SSEND, MPE_RECV, MPE_SENDRECV have inconsistent data types in some places*
- *Number of arguments to the PPWRITE routine was in error.*
- *Wrong size of array used in MPE_ALLREDUCE in RADIAT*

  CALL MPE_ALLREDUCE(SQ1 ,SQ ,NIV,MPREAL,MPERR)

  Four calls use NIV and should be NIV-1

.

Appendix B: Detail of Routines Requiring Communication

This appendix contains descriptions of the routines that have been reviewed for their affect on the MPI communication within the TOMCAT code. There is very little MPI work within the GLOMAP mode specific code and that is limited to reading or writing data, which uses the parallel Master I/O method for file access.

The routines are listed in alphabetical order with the notes made during a code examination exercise to reveal where there could be improvements. Not every routine that has an MPI construct has been analysed the emphasis is on the MPI_RECV and MPI_SEND that is discussed in section **2.3** of the main report. Several places where MPE_BCAST is used have not been analysed.

Routine	Designed	Changed	Verified
CALFLU	Yes	Yes	Yes
CALPHY	Yes	Yes	Yes
CALSUB	Yes	Yes	Yes
COLLF	Not need	No	No
DISTF	Not need	No	No
DOMAIN	No	No	No
EXCHUV	Yes	Yes	Yes
GATHERROW	Yes	Yes	Yes
LISTCOMM*	Yes	No	No
PHYSICS	Yes	No	No
POLCOM	Yes	No	No
PPREAD	Yes	No	No
PPWRIT	Yes	No	No
SCATTERROW	Yes	Yes	Yes
SETMPP	Yes	Yes	Yes
SPEGRD	Yes	Yes	Yes
SPETRU1	Yes	Yes	Yes
SWAP_NS	No	No	No

Table B1: List of routines discussed in this appendix.

For the majority of the descriptions for the routines the structure is; name, overview, algorithm and recommendations. In some cases extra detail is given to support the descriptions. Some of the recommendations apply to several subroutines where there is a common optimisation identified.

The method used for verification is to compare the output "fort.25". This is a single precision version of "fort.9" and is the usual binary file used by GLOMAP researchers to analyse the simulation. The fort.9 is the double precision result of the simulation and is processed by PDG.exe to make a single precision fort.25. This is a serial job but takes less than minute to process this resolution (T42 data). Even so, it is possible to chain a serial job from the end of the parallel job and this approach has been used in this research.

CALFLU

Overview

This subroutine is called from INIEXP during initialisation and then every 6 hours from INICYCL. It is the main TOMCAT routine for processing meteorological data into tracer mass fluxes. Fluxes are calculated at "grid-box" faces as for a "staggered-grid" method. The domain decomposition is such that the velocities lay on the edge of the domain and so communication is needed in only one direction.

Algorithm

```
Significant computational work
     (6 sections of code)

Loop 790 is followed by
     (Communicate VGRI to south)
A pairing of MPE_RECV and MPE_SSEND

Loop 91 Calculate UGRIA
     (only PEs in a column)
 Then send UGRIA to East
 Receive UGRIA from West

Sum along latitude (1...NLONMX)
Loop over PEs in my row
  Exchange SSUM with all PEs on my row
  Accumulate Zonal Mean in UZON (K, L)
All PEs have the zonal mean
Update UGRI from UGRIA
Send UGRI at MYLON to East
Receive UGRI into "0" from West
```

The following MPE calls are used by this subroutine

	NCYCLT=12 i.e. 6 hours simulated time
MPE_RECV	RBUF is VGRI + VGRIA receive from North
MPE_SSEND	SBUF is copy of VGRI sent to South
MPE_BARRIER	Hold extreme PEs until interior PEs complete exchange
MPE_BARRIER	just before UGRIA transfer to EAST
MPE_SSEND	UGRIA SEND to right SBUFA
MPE_RECV	UGRIA RECV from Left RBUFA
MPE_BARRIER	Hold extreme PEs until interior PEs complete exchange
MPE_SENDRECVR	SBUFA, RBUFA, IP (row): SSUM along latitude rows Zonal mean
MPE_BARRIER	Hold extreme PEs until interior PEs complete exchange
MPE_SENDRECVR	SBUFU, RBUFU, IRGT, ILFT: UGRI east to west exchange

Table B.2: MPE calls are used by subroutine CALFLU

Observations and Suggested changes

There is some inefficiency due to excessive memory usage. There are six buffers that have been allocated locally and need to be big enough for (NLONMX by NIV) elements.

```
C     For sending and receiving VGRI, VGRIA, UGRI, UGRIA
values
      INTEGER IBUF, IBUFU, IBUFA, ICOUNT
      PARAMETER (IBUF =NLONMX*NIV + NLONMX*LEV)
      PARAMETER (IBUFA=NLATMX*LEV,IBUFU=NLATMX*NIV)
      REAL SBUF (IBUF ), RBUF (IBUF)
      REAL SBUFA(IBUFA), RBUFA(IBUFA),SBUFU(IBUFU),
RBUFU(IBUFU)
```

IBUF, IBUFA and IBUFU are calculated before the program is compiled; a single value could be used to define two buffers, one for sending and one for receiving thus removing the memory requirement for the other four buffers. It is better to re-use those than carry excess memory. Care is needed to match the shape of other data structures. The loading of the buffers is not in a contiguous manner. The zonal mean can be calculated with row-collective communicators.

All the PEs that are not on the North Pole attempt to receive from the north and then the PEs that are not on the south polar region attempt to send information to the south. However, the only rows of PEs that are "immediately" ready are those around the North Pole. For example with 8 rows of PEs and 64 latitudes (as for the T42 case) this gives rise to the south to north cascade:

cascade	1	2	3	4	5	6	7
Latitude indices 1-8	S	C	C	C	C	C	C
9-16	N	S					
17-24		N	S				
25-32			N	S			
33-40				N	S		
41-48					N	S	
49-56						N	S
57-64	W	W	W	W	W	W	N

Table B.3: Showing the cascade of communications for existing implementation. S is a communication to the PE to south of this one. N is a communication with PE to north of this one. W is a waiting condition. C is continue with computation.

The west to east communication of UGRI is inefficient because each PE is forced to calculate the ID of its neighbour. This information can be determined at start-up within SETMPP as it is invariant throughout the run.

The algorithm could be made into an odd-even oscillator where the MPI tasks that are topologically on the zeroth row are tagged as even and those on the second row are tagged as odd because the row counting initiates at zero.

	oscillate	1	2
Latitude indices 1-8	Even	S	C
9-16	Odd	N	S
17-24	Even	S	N
25-32	Odd	N	S
33-40	Even	S	N
41-48	Odd	N	S
49-56	Even	S	N
57-64	Odd	N	C

Table B.4: Oscillator communications. S is a communication to the PE to south of this one. N is a communication with PE to north of this one. C is continue with computation.

The resulting communication pattern lets the exchanges occur in less time as several of the PEs are communicating simultaneously. There is no waiting as tasks are in the correct mode on entry to the communications layer.

CALPHY

Overview

This sub-routine is a high level flow control for the boudary layer scheme of this model. It has a section for initialisation of the problem and then calls to the more detailed lower level subprograms. The benefits of changes to this routine will be seen in the reduced MPI workload in broadcast. This will free the code of significant synchronisation overhead. The PPREAD and PPWRITE subroutines are discussed separately.

Algorithm

```
Initialisation section
Read data files
Call spegrd1
Call advance
Parallel read of file unit 79
  Filter for MYPROC=0 to read totemi
  BCAST totemi array
  PPREAD for 7 field variables
OMP parallel do loop over latitudes
   LINEMS()
MPI reduction ztotems
MPI BCAST Total emissions array
```

Observations and Suggested changes

Generally the communication in CALPHYS is mainly through MPE_BCAST after a MASTER I/O section. Replacing PPREADs and PPWRITs with parallel I/O functions will remove the requirement for MPE_BCASTs. There is a final MPI_REDUCE and MPI_BCAST pair that should be replaced by an MPI_ALLREDUCE.

CALSUB

Overview

This sub-program is called from the main TOMCAT program once per six-hour cycle after CALCOND, but is also dependent on IVDIF and ICONV. It is also called once during initialisation. It calculates the convection terms. Written in the header block there is a brief description:

" Calculate cloud ent/detrainment fluxes and vertical diffusion coefficients from model fields"

NOTE, the data WQ, WU, WV are stored (L, I, K) that is very different from the main TOMCAT code or the PBL scheme from NCAR CCM2. This convection code was adapted from an tracer transport model.

Algorithm

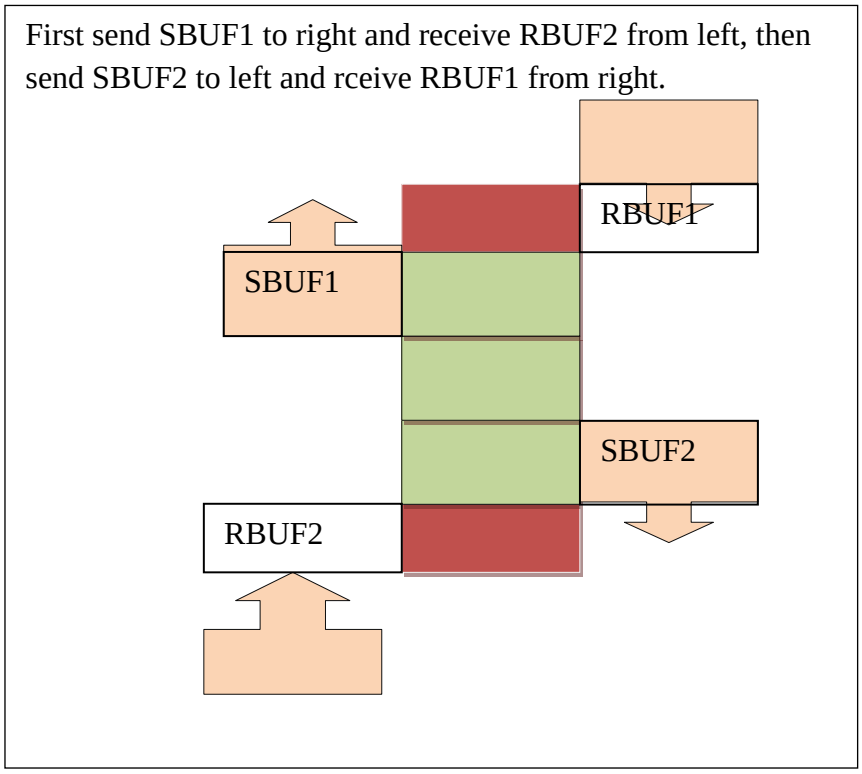
```
PPREADF (IEVAP) EVAP()  ! a formatted file fort.18
EVAP(NLONMX,NLATMX) plane of values
Exch WQ,WU,WV:
  Right-Left (East-West) communications:
  Fill sbuf1 with I=MYLON and sbuf2 with I=1
    Sendrecv (sbuf1, right, rbuf2, left)
    Sendrecv (sbuf2, left, rbuf1, right)
  Unpack buffers rbuf1, rbuf2
  South-North communications
  Fill sbuf2 with K=MYLAT, sbuf1 with K=1, WQ(L,I,K)
    Sendrecv (sbuf1,north, rbuf1, north)
    Sendrecv (sbuf2,south, rbuf2, south)
  Unpack buffers rbuf1, rbuf2 into WQ,WU,WV
     DO I = 1, NLONMX
       DO L = 0,NIV
          WQ(L,I,0) = RBUF1(ICOUNT)
      END DO
    END DO
More work
Read fort.59 SUGRTR (use zsugrtr for temp global storage)
It is then BCAST so each PE picks its own fragment!
RETURN
```

Communication pattern

The existing communications pattern in the north–south direction is not optimal. The PEs on the North Pole row only send south and never send north. The PEs around the South Pole only send north and never send south. This is used to trigger the cascade of communications from poles towards equator. This implies that the interior rows have to wait for the pole communications to complete before they can continue. This is another algorithm that has the inherent cascade of communication triggered from the North Pole. There is a calculation of QAM and QAC that could be done more efficiently.

There is some potential in reviewing the manner in which the buffers are filled and emptied as this looks like non-optimal access of large arrays. Another issue with buffers is that the

"polar" PEs continues to unpack RBUF1 on the North Polar Region and RBUF2 on the South Polar Region that have not been populated due to lack of communication.



EXCHUV

Overview

EXCHUV is called by INIEXP, and then from INICYCL, every 6 hours of simulation. The velocities at faces are exchanged. The buffers are filled differently according to the direction of communication. For WEST-EAST communication there are a varying number of

Algorithm

```
Main work
.
Loop KP =1 and 2
  Fillin2
  Listcomm (the actual MPI communication)
  Emptin2
  Barrier
.
Special  loop for NLONMX=LON
   (special case for a circumferential
hoop of longs)
```

Suggested changes

Make KP explicit and unroll the loop. Replace buffer packing with ***load_ns*** and ***load_we***. Replace buffer unpacking with ***unload_ns*** and ***unload_we***. Make the final special case more sensible in loop limits and reposition the conditional test.

GATHERROW

Overview

This subroutine is part of the boundary layer scheme and is typically invoked with arguments (UUX, ZUU, and PLEV). It will send the FFT parts of the three -dimensional geometric data PFL to other PEs. It will also collect the parts of specific FFT rows that are from other PEs. It will collect the sections into PFG from other PEs sent to this PE. For example PE5 will receive data from PE4, PE6 and PE7 in the 64 PE decomposition. It will also need to send data to PE4, PE6 and PE7.

Algorithm

```
PFG(NLON+2,NIV,NLAT)      !NLAT is the FFT decomposition so
in 64PE case it is 1.
PFL(NLONMX+2,NIV,NLATMX)  ! the field to be collected/sent

RESET PFG to ZERO
Make FFT row copy PFG(IG,LG, KG') = PFL(IL,LL,KL)
Pack SBUF(IPROC) on per processor basis ==PFL(section)

For all PEs  {note this could be very inefficient for HIGH
NPROC)
  IF( MY_PE)
     For all PEs except me
        IF data_to_be_recvd
              RECV()
        END IF
     END DO
  ELSE
    SB1 = SBUF(IP)
    IF (data_to_send)
      SEND SBUF1 to IPROC (other PEs on my row)
    END IF
  ENDIF
END  loop over PEs

Unpack  RBUF(IP) (info from each PE stored separate columns
of the 2-d array)
DO K
  DO I
    DO L   !!! NOTE poor loop order due to data mapping
      K' = K-MYLATREF
      PFG(I,L,K')=RBUF(IC,IP)
    END DO
  END DO
END DO
```

NOTE organisation is longitude, altitude and then latitude and is different to the main TOMCAT CTM where the organisation is longitude, latitude and then altitude.

Suggested changes

The loop over PEs should be replaced with a loop over only PEs on my row of the topology in a cyclic order that will improve communications. This is easily determined at the start of the simulation when the decomposition topology is set. This assumes that the PEs on a row will have common latitudes. Some of the loop ordering is not optimal and should be changed.

LISTCOMM

Overview

LISTCOMM(K) is called from DOCOMM(). The subroutine DOCOMM loads up the two send buffers according to the direction K. That is done by FILLIN2(K). The SW_SNDBUF and NE_SNDBUF have been packed ready for communication; there are corresponding SW_RCVBUF and NE_RCVBUF ready to receive data from neighbour PEs.

Algorithm

The communication is done in two passes of LISTCOMM so the test is for the direction of communication to establish which PEs will be exchanging data.

```
IF the communication is West to East THEN
  (a)Send NE_SNDBUF to East PE
  (b)Recv SW_RCVBUF from West PE
  (c)Send SW_SNDBUF to West PE
  (d)Recv NE_RCVBUF from East PE
ELSE communication is North to South
    IF (southern hemisphere)
      (e) Send NE_SNDBUF
      (f) Recv NE_RCVBUF
      (g) send SW_SNDBUF to South
      (h) recv SW_RCVBBUF from South
    ELSE
      (g) send SW_SNDBUF to South
      (h) recv SW_RCVBBUF from South
      (e) Send NE_SNDBUF
      (f) Recv NE_RCVBUF
    END IF
END IF
```

NOTE the order (e,f,g,h) is changed for northern hemisphere PEs.

Suggested changes

The west to east communication appears satisfactory with the neighbour being in a ready to receive state when the PE is ready to send. The compound SENDRECV function in MPI is used.

The north-south communication is not optimal. There is an attempt to improve it with the switch between north and south hemispheres but that has never been activated as the test is never true. It is likely that there can be gains in parallel performance if this code section is activated. However, if an odd-even pattern is adopted then there should be further improvement as, starting from the North Polar region, the alternate rows of PEs will be classed as “even” and communicate southwards first while the other PEs will be “odd” and

communicate northwards first. This oscillation means that each PE is sending and receiving with the correct corresponding PE rather than having to wait for other unrelated communications to complete. To achieve this pattern the SETMPP routine can be used to prepare the flag that indicates that the row is “odd” or “even”.

Suggested replacement algorithm:

```
IF  (odd)
  Send north (except north pole PEs)
  Recv north
  Send south
  Recv south
ELSE (even)
  Send South (except South Pole PEs)
  Recv South
  Send north
  Recv north
END IF
```

PHYSICS

Overview

The majority of this subroutine is involved with mapping data from the main TOMCAT chemical transport model into a data structure consistent with the PBL scheme from NCAR CCM. The section where communication is significant is considered in this note. It calls sub-programs CALPHY, SWAP_NS and PPWRITE and PPREAD are sub-programs that contain MPI work.

Algorithm

```
Map 8 variables from CTM to CCM 2
(i.e. so that altitude is second index not third )
For example : T3  (I,K,J)   = T3D (I,J,K)
SWAP_NS  for 10 fields dimensioned [(plond*plev) x plat].
Call  CALPHY
Then SWAP_NS for 3 fields  Q3, KVH, PBLHT
Convert these back from CCM2 to CTM
    [Q3, KVH, PBLHT become S0, DC and HTPBL]
PPWRIT every 6 hours to fort.79 using the Master I/O method
  (TOTEMI, IZM,SN,RNFP1,TS,TSSUB,CNT,CNB)
```

Suggested changes

Replace the PPWRIT with user defined MPI-IO wrappers: *PIO_WR_2D_NH() and PIO_WR3D_NH()*. These have only been prototyped and are not fully implemented yet.

The loop order and data structure should be considered to gain better use of cache and vector processing.

POLCOM

Overview

This sub-program is called from ADVY2 for PEs that contain the Polar Regions.

Algorithm

```
copy sm0 into sbuf(0)
copy s00 into sbuf(1..ntra)
work out who is row master (e.g. 56,57,58,59:
decide 56)
decide which PEs are on same row
for each PE in a row (nproci) do a
sendrecv with all  other procs in the row
```

Communications profile

Here is an example row, 15 (i.e. it is row 14 in MPI coordinates)

	PE 56	PE 57	PE 58	PE 59
1	SR57	SR56	W	W
2	SR58	W	SR56	W
3	SR59	SR58	SR57	SR56
4	C	SR59	W	SR57
5	C	C	SR59	SR58

Table B.5: Showing an inherent serialisation of communication and costs 2 "cycles" more than it should. SR is a send-receive exchange pair and W is waiting for a PE, ready to exchange data. C is where the PE has moved onto do main code computations.

Recommendation

Even numbers communicate first time to east and ascending order. Then odd communicate first time to west and cycle in a descending order. Column IDs can be used for odd and even indicators. PE ids are assumed to start at 0 in the first column of the topology.

	PE 56	PE 57	PE 58	PE 59
1	SR57	SR56	SR59	SR58
2	SR58	SR59	SR56	SR57
3	SR59	SR58	SR57	SR56

Table B.6: An alternative communication pattern based on the existing code structures. However, this is better done by GSUM:

```
sbuf(0) = sm0
sbuf(1:ntra) = s00(1:ntra)
call mpi_allreduce (sbuf, rbuf, ntra+1, mpi_double,
mpi_sum, row_comm, ierr)
 sm0 = rbuf(0)
 s00 = rbuf(1:nta)
```

This algorithm requires the row communicator to be pre-set. This is now in SETMPP. SBUF and RBUF are already available. Then let the MPI implementation do the hard work.

PPWRIT

Overview

There are several fields written to disk for one of the following: restart, checkpoint or final result recording. As the fields are spread over all the PEs then there has to be a gathering operation. A "master-I/O" model is the model employed by TOMCAT and thus GLOMAP.

The TOMCAT routine "PPWRIT" has a call to the function "COLLF" where data is gathered from all other PEs and then has a single unformatted write to a binary file (although this is also indirect through a call to "MPE_WR"). There are two methods for collection; one has dimensions set for fields that include halo data and the other has dimensions set for fields that do not have halo data.

The variety of data have a common theme in that even though some are 4 dimensional the first two dimensions represent a layer of atmosphere in the globe i.e. arranged as "lat x longitude", this is used within the collector routine to simplify data accumulation. However, it also means that a certain amount of re-shaping has to be done at the program level and the calls to PPWRIT are embedded in loops over the outer dimensions: i.e. from 1 to NIV and 1 to NTRA or similar. The complication arises if the data has haloes or not. Non-halo data can be dealt with contiguously. Halo data has to be skipped as that is not written to disk. It allows the data file to be independent of the number of PEs in use on at any time. There is a flag to tell **PPWRIT** that the data category is with or without halo and there is a test for the type of data surrounding the call to a single routine "**COLLF**" that does the cyclical gathering.

List of arrays grouped by category: halo or no-halo

Halo	*SM*			---					
No Halo	*ST*	*T3D*	*PV3D*	*PLT*	*Q3D*	*H3D*	*GL3D*	*TSOL*	*PSOL*

Table B.7: List of parameters when called by FINITER, IFSO1 (fort.9)

Halo	SM	U3D*	V3D*									
No Halo	T3D	PLT	W3D	PV3D	H3D	UGRIO*	VGRI	WGRI	DGRI	GL3D	TSOL	PSOL

Table B.8: List of parameters when called by FINITER, IFSO2 (fort.12)

NOTE (*) indicates that these fields are placed within the block of fields of different category. Some of these are processed e.g. ugrio= ugri but shifted as it is face centred i.e. on a staggered grid.

NO HALO	IZM	SN	RNFP1	TS	TSSUB	CNT	CNB

Table B.9: variables supplied when PPWRIT is called by CALPHY (fort.79)

HALO	SM	S0	SX	SY	SZ	SXX	SXY	SXZ	SYY	SYZ	SZZ

Table B.10: variables supplied when PPWRIT is called by FINCYCL, IFRF (fort.31) and repeated in FINEXP

Algorithm of PPWRIT

```
Test for data type
  Call COLLF with dimensions for halo
Else
  Call COLLF with dimension w/o halo
Call MPE_WR (write this plane as one-d array to an
unformatted binary file)
```

Description of COLLF

```
Load a buffer with the local interior data for this plane
IF ( I am iomaster)  THEN
    recv data from each PE into the a single buffer(shifted
by PE id)
ELSE
  send data to iomaster
END IF
Unpack the receive buffer into a global field array
```

The interesting feature is that this model makes no assumption about the data size distribution; it will allow variation between PEs. Unfortunately it adds an overhead that could have been avoided using an MPI_ALLGATHER feature. The corresponding routine for reading from files and distributing to the other PEs is PPREAD using DISTF and MPE_RD.

Recommendation

The first recommendation is to remove the conditional tests for whether the data has halo or not. There should be dedicated routine to halo and non-halo data. The proposal is to replace the planar writing with MPIO-IO data structures. The recommendation is for the routine to be duplicated into a second routine and two new names created:

```
pio_data2d_wr_wh(IFRF, il,iu,kl,ku, pfl)
pio_data2d_wr_ nh(IFRF, ni,nk, pfl)
```

This will avoid using the flag and conditional test nested in the NIV and NTRA loops. Replace the exisitng routine where “TRUE” appears in the original PPWRIT call: i.e.

CALL PPWRIT(IFRF,S0 (NIMN,NKMN,L,JV),.TRUE.,0)

with

CALL PIO_DATA2D_WR_WH (IFRF, NIMN,NIMX,MKMN,NKMX, S0(NIMN,NKMN,L,JV))

However, the planar write is also inefficient. A better method is to replace these completely with three dimensional data volume writes and for 4d data volume species writes. This can be

trialled with the fort.79 and no halo data type. An IO file is connected to the program using *MPI_FILE_SET_VIEW* and the data type parameter is needed to match the data structures that will be written to the file. The type has to be created elsewhere with a call to *MPI_CREATE_TYPE* and must then it should be stored in COMMON (or module) for later use. An additional recommendation is to apply compiler directives to vectorised loops in conjunction with inline directives where there are embedded calls. PPWRIT should be always in-lined.

A third recommendation is related to the communication as this is also not ideal. An MPI_ALL_GATHER can be used as the data is uniform over all PEs and the buffers are loaded similarly. However this still assumes a master I/O model. It will be better to use MPI-I/O structures and write data to the disk from all PEs and if this causes problems then the "row-master I/O "model should be adopted.

SCATTERROW

Overview

This sub-program of the PBL scheme is typically invoked with (UUX,ZUU,PLEV). It will send the off-PE sections of PFG (the FFT solution) to other PEs. It will collect the sections of PFG from other PEs sent to this PE it places those parts into the local working array PFL. It works with "vertical" planes of data, i.e. NLONMX by NIV dimensions.

Algorithm

```
PFG(NLON+2,NIV,NLAT)    !NLAT is the FFT decomposition so in
64PE case it is 1.
PFL(NLONMX+2,NIV,NLATMX)
RESET PFL
Make local copy PFL(IL,LL,KL) = PFG(IG,LG, KG')
Pack SBUF(IPROC) on per processor basis ==PFG(section)
For all PEs [this is a specially arranged cycle of
communication]
  IF( MY_PE)
     For all PEs except me
        IF data_to_be_recvd
              RECV()
        END IF
     END DO
 ELSE
    SB1 = SBUF(IP)
    IF (data_to_send)
      SEND SBUF1 to IP (SSEND!!!)
    END IF
 ENDIF
END DO
UnPack RBUF(IP)
DO K = 1, NLAT  (# FFT LATITUDES ON OTHER PES)
  DO L = ALTITUDES
    DO I = NLONMX   (This was outside L loop in early version)
      K' = K-MYLATREF
     PFL(I,L,K')=RBUF(IC,IP)
END DOs
```

Suggested changes

Improve loop over PEs for optimal communications and avoid conditional statements. The order of I and L loops should be inspected and corrected so that the loop over I is inside the L loop.

SETMPP

Overview

Called by INIEXP, this sub-program determines the rank and number of tasks in the MPI job.

Algorithm

```
IF IMPP
  CALL MPI_COMM_SIZE
  CALL MPI_COMM_RANK
ELSE
  NPROC=1
  MYPROC = 0
END IF
```

This is a very brief initialisation to trigger that the program is running on an MPI capable system. It is an appropriate location to continue all the set up of the parallel topology, groups and communicators.

SPEGRD1

Overview

This subroutine is called from CALPHY to do FFT work for the PBL scheme. Many of the arrays are dimensioned (longitude by altitude by latitude) and thus there has to be a mapping from TOMCAT organisation into the PBL FFT organisation. NOTE in the FFT decomposition NLAT is the number of whole latitudes processed by this PE (must be a complete set of longitudes). All MPI work is done in lower sub-programs.

Algorithm

```
Initialise local working arrays
FFTSETUP (recommend dong this earlier, once only)
GATHERROW is called seven times for different variables
FFT991 repeated for  7 variables
SPETRU1 (ZPP, ZPHIS, ZQU, ZQV, ZTT, ZVORT, ZDIVQ, ZPL, ZPM)
SPETRU1 (ZPP, ZPHIS, ZUU, ZVV, ZTT, ZVORT, ZDIV, ZPL, ZPM)
FFT991 repeated for 10 variables
SCATTERROW is called 12 times for different variables
Scale/convert UUX variables
```

Suggested changes

Move FFTSETUP to CALPHY, the top level PBL subroutine. No other change needed. Changes should happen within lower routines e.g. SPETRU1, SCATTERROW and GATHERROW.

SPETRU1

Overview

This routine is working in the PBL FFT decomposition space so the task arrangement is slightly different to the geometric decomposition. By coincidence 64 PEs and 64 latitudes is a unique combination. Current use is more likely 32 PEs so there will be 2 latitudes per PE. It is called twice from ***SPEGRD1***. The fields in the argument list are prefixed with Z or ZZ so that a local copy can be used for restricted calculations. It appears that processors in the northern hemisphere do a lot of the work (perhaps all of it). There are three SENDRECV pairs where a southern latitude processor packs a buffer and sends the info to the northern hemisphere PE.

Here is the description taken from the comment block at the top of the subroutine:

```
"Spectrally truncate input fields which have already been
transformed into Fourier space.  Some arrays are dimensioned
(2,...), where (1,...) is the  real part of the complex Fourier
coefficient, and (2,...) is the imaginary.  Any array
dimensioned (plond,...) *cannot* be dimensioned (2,plond/2,...)
```

*because plond *may* be (and in fact currently is) odd. In these cases reference to real and imaginary parts is by (2*m-1,...) and (2*m ,...) respectively."*

Map out the topology

For the T42 resolution the TOMCAT advection grid is 128x64x31 (Lon x Lat x Alt). The table B.11 shows the topology for the geometric calculations over 64PEs. The mapping from this to the FFT decomposition is that the northern hemisphere (grey shaded cells of the table) relates the latitude indices (1 to 32) to a processor (0 to 31) and for this arrangement there is a one-to-one correlation i.e. latitude 1 is processed by PE0 and latitude 32 is processed by PE31. There is also a pairing between southern and northern hemisphere e.g. PE 61 pairs with PE 2 because they have the matching (mirrored about the equator) latitude information for the FFT algorithm. The ***GATHERROW*** and ***SCATTERROW*** subroutines ensure that the information from PEs with shared latitudes is transferred to the "owner" for the FFT algorithm e.g. PE4, PE6 and PE7 send their info for Latitude 6 to PE5.

Lat index	LON=1-32	LON=33-64	LON=65-96	LON=97-128
1-4	**0**	**1**	**2**	**3**
5-8	**4**	**5**	**6**	**7**
9-12	**8**	**9**	**10**	**11**
13-16	**12**	**13**	**14**	**15**
17-20	**16**	**17**	**18**	**19**
21-24	**20**	**21**	**22**	**23**
25-28	**24**	**25**	**26**	**27**
29-32	**28**	**29**	**30**	**31**
33-36	32	33	34	35
37--40	36	37	38	39
41-44	40	41	42	43
45-48	44	45	46	47
49-52	48	49	50	51
53-56	52	53	54	55
57-60	56	57	58	59
61-64	60	61	62	63

Table B.11 : Topology for 64 PE decomposition, grey shaded cells are PEs that do the FFT work.

Algorithm

```
INITIALISE arrays ()
  determine IPROC, the partner PE
  IF (NHEMI_FFT)
     create local copy of data.
  END IF
  IF ( NHEMI_FFT) THEN  ! I am a PE in Northern hemisphere
     RECV(RBUF) from IPROC southern partner [e.g. PE61
sending to PE02]
     Unpack RBUF into local copies (shifted by NLAT) ! in
this case NLAT is 1
  ELSE               ! I am a PE in Southern hemisphere
     Pack SBUF (5 VARS)
     SSEND SBUF to northern partner [e.g.  PE61 sends toPE02]
  END IF
  IF (NHEMI_FFT)
    FFT work
    FFT work
  END IF
  IF (NHEMI_FFT)
C Need to sum phi, alps, d, t, vz over northern hemisphere
PEs
Indirect Global SUM (only on northern hemisphere)
    SENDRECV loops over PEs for 9 fields (these should be
ALLREDUCE so need a communicator for Northern Hemisphere)
 END IF

 IF (NHEMI_FFT)
    pack SBUF
    SSEND to southern partner PE (IPROC)
 ELSE  (Southern FFT Hemisphere)
    RECV RBUF from Northern Partner PE (IPROC)
    Unpack RBUF
 END IF
```

NOTE dealing with a plane (LON x NIV) and NLAT is number of latitudes processed by this PE in the FFT decomposition.

Suggested changes

Three SENDRECV sections are used to create global sums which are potentially wasteful. It has been used because there is no “northern hemisphere communicator”. One can be created at the same time as the Cartesian geometry (or in FFT setup if it is more useful) in SETMPP. Then these SENDRECV sets can be simplified into six global summation calls:

```
CALL MPI_ALLREDUCE (SBUF,PHI1,PSP/2,MPI_SUM, COMM_NHEM,IERROR)
CALL MPI_ALLREDUCE (SBUF,PHI2,PSP/2,MPI_SUM, COMM_NHEM,IERROR)
CALL MPI_ALLREDUCE (SBUF,ALPS,PSP,MPI_SUM, COMM_NHEM,IERROR)
CALL MPI_ALLREDUCE (SBUF,D,PSPL,MPI_SUM, COMM_NHEM,IERROR)
CALL MPI_ALLREDUCE (SBUF,T,PSPL,MPI_SUM, COMM_NHEM,IERROR)
CALL MPI_ALLREDUCE (SBUF,VZ,PSPL,MPI_SUM, COMM_NHEM,IERROR)
```

However, to retain consistency with the rest of the code the same can be achieved using a wrapper subroutine, MPE_COMM_RED. As follows:

```
SBUF=0.0
SBUF(1:PSPL)=T(:)
CALL MPE_COMM_RED (SBUF,RBUF,PSPL,MPREAL,MPERR)

DO K=1,PSPL
        T(K)=RBUF(K)
ENDDO
```

This would assume that the communicator is COMM_NHEMI and that the reduction is SUM as shown in the six individual MPI calls detailed above**.**

Swap NS

Overview

This subroutine is called from PHYSICS several times before a call to CALPHY; it is called ten times for different variables. After CALPHY it is called three times for a different set of variables.

Algorithm

```
Some surplus re-calculation of row, column and
corresponding PE ID.
Load a buffer with local field data
IF (second half of the PEs) then
  Be ready to receive information from the first
half of PEs
  Unpack receive buffer into local field storage
in reverse order
  Send a buffer to north partner
ELSE
  PE send buffer to their southern counterparts
  Receive data from southern counterpart
  Unpack buffer into local field in reverse order
of latitude
END IF
```

Recommendations

Move the calculation of the correspondent PE into the SETMPP or FFTSETUP sub-programs. Make the receivers into IRECV (non-blocking) so that there is no overhead or waiting to receive.

APPENDIX C: Parallel File Access and Review of Memory Usage

In task 2.3 the MPI-IO approach has been discussed. This appendix gives an example of a specific implementation that will benefit from a parallel I/O implementation. The example assumes that 16 PEs are being used to execute the simulation. The MPI tasks are distributed over the PEs in a virtual two-dimensional grid aligned with longitude and latitude.

Within the code is a section where zonal means are calculated. This involves summing all the contributions from a specific parameter (or set of variables) around a particular latitude. The result is the variation of the parameter along the median of the globe. The existing method shown diagrammatically in the figure C.1.

All processors on row zero (deep orange) copy the values of the parameter into a communication buffer (pale orange, a three dimensional storage space). This is then sent using MPI "send and receive" to the processor ranked zero in the global communicator topology (generally this includes the median line of longitude). The receiving processor stores the information in "ARR3DTMP1" this is copied into a globally sized three-dimensional array "ARR3DSP1" to free up the temporary storage receptacle. The zonal mean is then calculated on the main processor. Other PEs on other rows also send their three-dimensional information to PE rank 0 which again has to store and calculate zonal means for those latitudes. The result is written to file for later analysis by the researcher. The disadvantage of this method is the number of copies of data and the volume of data that has to be written to file by one PE.

The proposed alternative is for each PE to accumulate the partial zonal mean locally and is shown diagrammatically in figure C.2. The summation has reduced it to a plane of information. The plane of information is communicated using an MPI reduction function (another summation) to the PE in the first column of the topology. All PEs in the first column can then write their result to the results file using parallel I/O. There are several benefits from this; the memory requirement is reduced; the volume of data being communicated is reduced; the number of calculations done on MPI task rank 0 is the same as for all other PEs; and it will arrive at the data writing point at approximately the same time, which should be less than that taken in the original implementation.

A side note on communicators: MPI uses communicators to identify a group of tasks that will communicate. There is a virtual topology associated with the group and this facilitates simpler approach to any communications within the group. The proposed change is to introduce additional groups and communicators that can be determined at start-up and thereby make later calls to the communication layer simpler to understand. This is very easy to set up due the strongly structured nature of the geometry with no variation in the number of longitude, latitude or altitude resolution during a simulation.

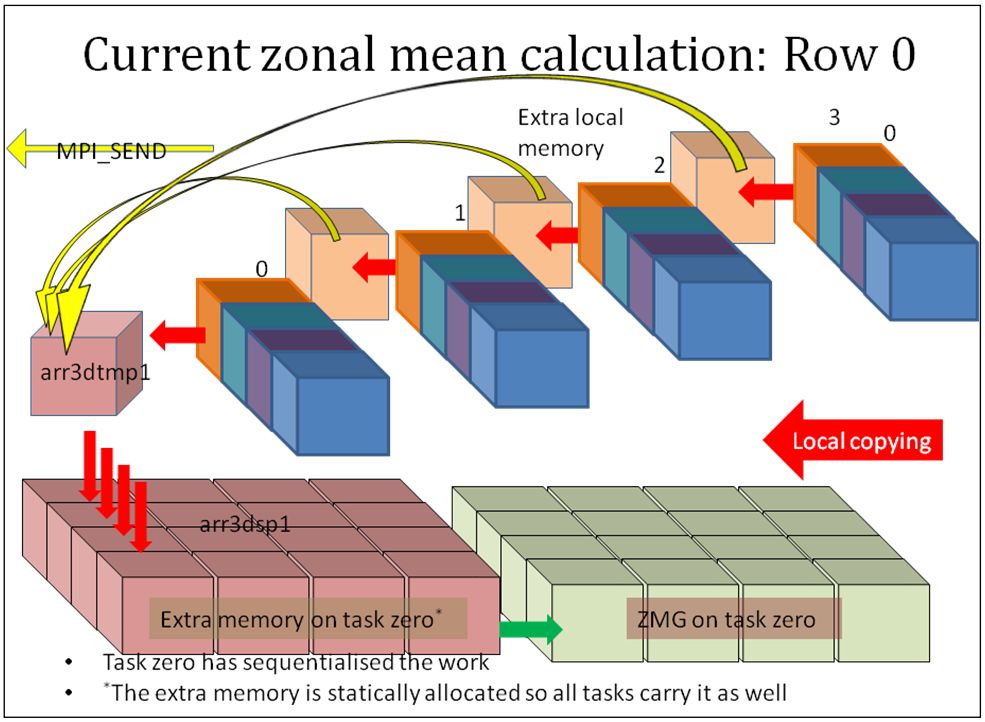


Figure C.1: Each domain makes a copy of the data and passes it to the task 0 process. The task zero process copies the sub-section into a large array and then processes it latitude by latitude.

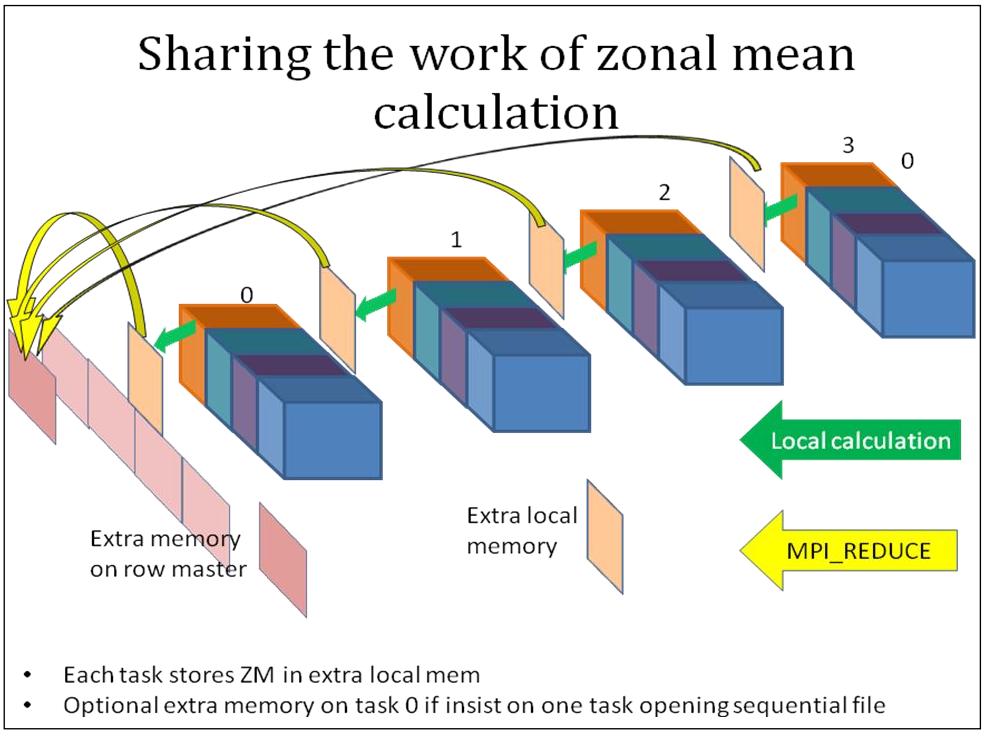


Figure C.2: Each PE will calculate the contribution to the zonal mean from its local data. The data is passed along the row of PEs as a global summation calculation (MPI_REDUCE). The task at the end of the row sends it group of latitudes to the task 0 or it could write to the file using MPI-IO data structures.