Profiling Methodology and Performance Tuning of the Met Office Unified Model for Weather and Climate Simulations

Peter E. Strazdins*, Margaret Kahn†, Joerg Henrichs‡, Tim Pugh§, Mike Rezny¶
* *School of Computer Science, The Australian National University,*
Email: Peter.Strazdins@cs.anu.edu.au
† *NCI National Facility, Australia,*
Email: Margaret.Kahn@anu.edu.au
‡ *Oracle,*
Email: joerg.henrichs@oracle.com
§ *Bureau of Meteorology, Australia*
Email: T.Pugh@bom.gov.au
¶ *Monash Weather and Climate, Monash University,*
Email: Michael.Rezny@monash.edu

***Abstract*—Global weather and climate modelling is a compute-intensive task that is mission-critical to government departments concerned with meteorology and climate change. The dominant component of these models is a global atmosphere model. One such model, the Met Office Unified Model (MetUM), is widely used in both Europe and Australia for this purpose.**

This paper describes our experiences in developing an efficient profiling methodology and scalability analysis of the MetUM version 7.5 at both low scale and high scale atmosphere grid resolutions. Variability within the execution of the MetUM and variability of the run-time of identical jobs on a highly shared cluster are taken into account. The methodology uses a lightweight profiler internal to the MetUM which we have enhanced to have minimal overhead and enables accurate profiling with only a relatively modest usage of processor time.

At high-scale resolution, the MetUM scaled to core counts of 2048, with load imbalance accounting a significant fraction the loss from ideal performance. Recent patches have removed two relatively small sources of inefficiency.

Internal segment size parameters gave a modest performance improvement at low-scale resolution (such as are used in climate simulation); this however was not significant a higher scales. Near-square process grid configurations tended to give the best performance. Byte-swapping optimizations vastly improved I/O performance, which has in turn a large impact on performance in operational runs.

***Keywords*-weather prediction, climate modelling, parallel computing, performance analysis, high performance computing.**

I. Introduction

The Met Office Unified Model (MetUM, hereafter referred to as simply 'UM') [1] is a global atmospheric model developed by the UK Met Office (MetO), which has been in operational use since the early 1990s. Its name signifies that it can be used for both weather and climate prediction. As well as in the UK, the UM is used for both of these purposes in South Korea, India, South Africa, and New Zealand; recently it has also been adopted by the Bureau of Meteorology (BoM) in Australia [2].

When configured for modelling the whole earth's atmosphere, the UM models the atmosphere using a rectangular grid with variable distance between grid points to account for the curvature of the earth. The N512L70 grid has $1024 \times 769 \times 70$ grid points in the east-west, north-south and vertical dimensions, respectively. This grid and the N320L70 ($640 \times 481 \times 70$) are typically used for weather prediction; smaller grids such as the N96L38 ($192 \times 145 \times 38$) are used for climate prediction. In general, the doubling of the horizontal resolution has an 8-fold effect on the computational work. This is because the timestep also may need to be halved, as a restriction of the model is that mass may not be transported (e.g. through advection) across more than one grid point upon each simulated timestep.

For current operational use in weather prediction, the BoM uses UM codes (version 6.4) on a global earth grid at a resolution of N144L50. For 2011, it is desired to use at first an N320L70 grid and by later an N512L70 grid for greater accuracy (forecast 'skill') [3]; 24 hours of model simulation time in 500 seconds using fewer than 1000 cores on an Intel/Infiniband cluster or less is the operational target. Such a short run time is required for timely delivery of forecasts and 'ensemble' simulations. Ensemble simulations, where for example 24 simulations, each with their input data perturbed, are performed to generate a representative sample of the possible future atmospheric states for probability assessment. It should be noted that the operational simulation time for weather prediction is typically 10 days; a 24 hour target is deemed sufficient to project to this time. The version of UM to be used is 7.5 (released by the UK Met Office in April 2010) [4], which contains the first implementation

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.283

of multi-threaded MPI code [5] and is optimized for IBM Power processors.

The Australian Community Climate and Earth System Simulation (ACCESS) project [6] is a joint venture between CSIRO, BoM and five Australian Universities. The project is founded on the principle that, by using a common model infrastructure, larger climate science questions can be tackled by the national community. ACCESS is scheduled to contribute to the fifth Intergovernmental Panel on Climate Change in 2011. Here an N96L38 grid is typically used, with the computationally dominant component UM being coupled to ocean, land surface and sea ice models. The target core count is 96 for the UM, and 25 cores for the other models. Next generation short-term UK climate models are scheduled to use an N216L85 grid, projected to scale up to an N320L70 grid in 2012 [7].

Thus, large amounts of supercomputer time in sites across Australia and beyond will soon be committed to running the UM. Without an understanding of its performance, users are likely to run the code on 'out-of-the-box' configurations, potentially making poor use of available resources.

This paper reports our experiences in profiling and tuning UM 7.5 code performance at the N96L38, N320L70 and N512L70 grid resolutions, with the motivation of understanding and improving performance in order to meet the above targets. We used the `vayu` cluster at the National Computing Infrastructure (NCI) National Facility [8], which is an Intel/Infiniband cluster.

This paper is organized as follows. Section II describes the Unified Model and its code structure. It also describes its internal timer module, which becomes an important tool in our performance evaluation methodology. Section III gives relevant details of the `vayu` cluster, and the constraints we have for this project on its usage. Section IV describes our preliminary experiences in performance analysis of the UM on the `vayu` cluster. From a clearer understanding of the properties and limitations of both, a performance evaluation methodology is proposed in Section V. Results involving a scaling analysis are given in Section VI and those involving parameter tuning are given in Section VII. Related work is discussed in Section VIII, with concluding remarks from our experiences being given in Section IX.

II. The Unified Model

Part of the Unified Model's widespread appeal is its flexibility, allowing it to be used for both weather and climate prediction (short and long timescales). It can be configured to run over regional and global models, and can be coupled with ocean and other earth system models [9].

A UM benchmark is configured using the Unified Model User Interface (UMUI). This produces a directory containing the source code and data files. According to the configuration, compilation is conditional (using `cpp` directives). Preprocessing is used to select at compile time one from

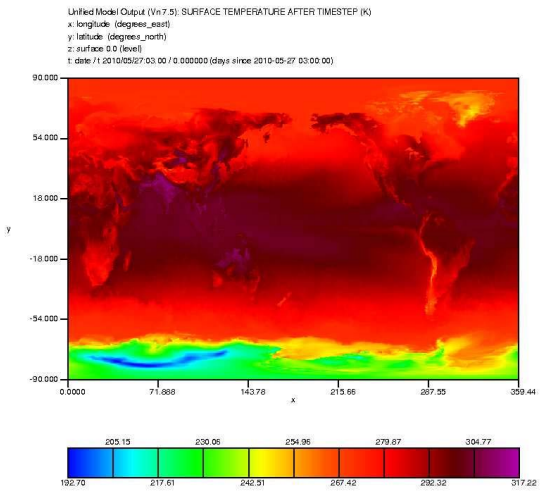

Figure 1. Surface temperature input data of the model (date 27/05/10)

a number of different algorithms for major sections of the model, and to select architectural parameters (e.g. little vs. big-endian). Once built, it is difficult to run the executable for input data substantially different (e.g. a different grid resolution) than what it was configured for.

Like other atmospheric models, many of the components of UM are highly data-dependent; thus, any benchmark must use realistic atmospheric data.

For global atmosphere simulation, the main input data file is a 'dump file' containing a snapshot of all the atmospheric state. Figure 1 gives the surface temperature from the input data file used for the N320L70 grid; this file is approximately 1.5 GB in size. When used in production, the models dump statistics on the atmosphere's state over the simulation; this is handled by Spatial and Temporal Averaging and Storage Handling (STASH) sub-system. Other input files include configurations for the STASH, specification of the short-wave and long-wave radiation parameters, and namelist files for configuring other aspects of the simulation (these last have approximately one thousand parameters).

When running on a grid of $P \times Q$ distributed memory processors, the horizontal dimensions of the grid are divided evenly over the processor grid. As well as its own block column of points, each processor will also have a 'halo' of surrounding grid points.

A. UM Code Structure

The UM 7.5 codes consist of approximately 900,000 lines of Fortran 90 source code. Reflecting the long evolution of the model, predominately Fortran 77 features are used. The code is pre-processed via `cpp`; this is mainly used for code declarations which are often repeated, such as common block declarations and 'standard' sub-lists of parameter names used for subroutine calls.

The top-level control routine for the model is called `um_shell()` [9]. This initializes the parameters used for the simulations (namelist files initializing common blocks and environment variables). The main simulation routine, `u_model()` is then called. This allocates the major arrays, and reads in the data files. It then iterates over the required timesteps, controlling I/O and transfer of data to model couplers (if required). At each iteration, the `atm_step()` routine is called, which undertakes the simulation of the atmosphere (both dynamics and physics) over a single timestep.

For large grids, the computationally dominant part of this process is applying a solver to the Helmholtz pressure-temperature equation for acoustic wave dynamics: this is performed using a Generalized Conjugate Residual (GCR) iterative solver on a tridiagonal 'linear system'. This system is essentially that of the vertical dimension, but replicated across all horizontal mesh points.

The dominant communication patterns in this part of the computation are collective communications (row, column and global), principally barriers and reductions. Other parts have these and also a significant amount of point-to-point communication from the eight neighboring horizontal points (NS, EW and diagonals); these form the so-called 'halo exchanges'.

B. Internal Profiler

UM includes a profiling module which may be activated through namelist configuration [4]. It times parts of the major sub-routines computation using high-resolution wallclock and CPU timer functions. As well as recording the time for a whole subroutine (called an 'inclusive timer'), it also measures parts of the subroutine excluding the calls that it makes which are also timed (called a 'non-inclusive' timer). `u_model()` is the top-level non-inclusive timer; the sum of all non-inclusive timers thus gives the time of the call to `u_model()`, which is effectively the total execution time for the simulation. Approximately one hundred of each kind of timer are profiled. For the timers that are called globally, all processors are synchronized at the start of a timer call.

As well as giving time totals for each such timer (subroutine), it also records the number of calls made. Statistics of the time totals across all processing elements are also gathered; these include the mean, median, maximum and minimum times across each process (PE). A 'parallel speedup' for each subroutine is also provided; this is defined to be the total CPU time divided by the maximum wall time[1]. The output is sorted in terms of wallclock times.

Figure 2 indicates these statistics for a 9 timestep UM simulation on an N512L70 grid; the `read_dump()` subroutine, which reads the dump file, is called only once; most others are called 9 times. It can be seen that execution time is dominated by the Helmholtz solver.

An estimate of load imbalance can be obtained from the differences between the maximum and average of each timer across all processors. This will be an underestimate in the sense that the waiting time in communication due to load imbalance within a function will not be included; nonetheless, it still provides a useful lower bound.

The overhead of the timer routine itself is also recorded (it has been negligible in all our experiments). However, this does not include the synchronization at the start of the timer call; this however does get included in the time of the enclosing ('non-inclusive') timer.

The timer output is given at the end of a run; it is also possible to get timer output on a number of intermediate regions by inserting the appropriate timer routine calls in the source code.

III. The `vayu` Cluster

The `vayu` cluster has 1492 nodes consisting of Sun X6275 blades, each with two quad-core 2.93 GHz Intel X5570 Nehalem processors and cache sizes of 32KB (per core) / 256KB (per core) / 8MB (per socket). The available physical memory per node is 24 Gbytes, excepting 48 nodes which have 48 Gbytes. The interconnect is single plane QDR Infiniband, with a measured MPI latency of 2.0μs and uni-directional bandwidth of 2600 MB/s per node [8]. The nodes are configured with a CentOS 5.4 Linux distribution with the 2.6.32 kernel.

Parallel jobs are launched using a locally modified version of the Portable Batch System (PBS). Maximum walltime and virtual memory limits must be specified upon submission. The scheduler (by default) allocates 8 consecutively numbered MPI processes to each node allocated to the job. Where possible, contiguous sequences of nodes are allocated to a job to minimize interference between jobs. Jobs may be suspended, e.g. to make room for a job requiring a large number of cores. A typical snapshot of the main job queue (taken noon in mid-December) is:

```
1216 running jobs (465 suspended),
280 queued jobs, 11776 cpus in use
```

Run-time files used in our benchmarks such as the dump file and output from runs are on a Lustre file system. Our 'project' was restricted to use no more than 2048 cores and was allocated a few thousand CPU hours of cluster usage.

IV. Preliminary Experiences on the N320L70 Benchmark

Our initial experiments were run with a UM N320L70 benchmark compiled with the Intel Fortran compiler 11.1.072 and using OpenMPI 1.4.1 with processor affinity enabled. In order to simplify profiling, STASH output was disabled. Due to memory constraints, it was not possible to run the benchmarks on fewer than 2 nodes.

[1]This is not meaningful on systems where MPI wait time is included in the CPU time, such as `vayu`.

	ROUTINE	MEAN	MEDIAN	SD	% of mean	MAX	(PE)	MIN	(PE)
1	PE_Helmholtz	206.97	206.98	0.05	0.02%	207.02	(7)	206.85	(48)
2	SL_Full_wind	27.54	24.09	6.91	25.09%	44.78	(3)	23.99	(12)
3	ATM_STEP	36.39	38.53	9.46	25.99%	44.60	(44)	12.35	(2)
4	SL_Thermo	25.38	26.60	3.45	13.58%	31.15	(3)	21.05	(44)
5	READDUMP	24.18	24.36	1.12	4.62%	24.37	(58)	15.76	(0)
6	Convect	16.64	18.14	2.22	13.34%	18.87	(0)	12.48	(63)
7	LW Rad	10.19	10.21	0.20	1.96%	10.65	(14)	9.72	(40)
8	Atmos_Physics2	7.66	6.85	1.92	25.07%	11.72	(53)	5.20	(33)
9	LS Rain	5.23	4.91	1.48	28.22%	9.70	(2)	2.68	(40)
10	SW Rad	5.41	7.11	3.27	60.47%	8.90	(59)	1.09	(0)

Figure 2. Extract of timer output: wallclock time statistics for an N512L70 benchmark with 64 processes for a 1.5 simulated hour run

To profile individual subroutines, a tool such as Oracle Solaris Studio `collect`, in conjunction with MPI profiling and hardware performance counters, would have been ideal. This would in principle enable a performance analysis based on both communication and memory hierarchy issues. However, the Linux kernel available at that time did not support performance counters. While we were able to successfully use `collect` for the N96L38 benchmark for walltime profiling, we were unable to do so for the N320L70, due to the sheer size of the generated data.

The MPI profiling tool IPM [10] was also attempted; this only completed successfully on 256 or fewer cores. For a 5 hour simulation on a 16×16 process grid, IPM reported that 44% of communication time was spent in MPI, dominated by `MPI_Barrier()` (14%), `MPI_Scatterv()` (9% - called only from a subroutine called `q_pos_ctl()`), and `MPI_allReduce()` (7%).

It became evident that the size and complexity of the N320L70 benchmark would require a more lightweight approach. The UM internal profiler, as described in Section II-B, was employed. To conserve usage quotas, the simulations were generally limited to 2 simulated hours (12 timesteps).

A. Variability Analysis

We observed a variability in run times of up to 50% for repeated jobs; however, these were bimodal, with a number of 'fast' runs clustered around 5–10% of each other. Within each run, each iteration of `atm_step()` also showed some variability, with iterations k, where $k\%6 = 0$, taking significantly more than the others, and iteration 0 taking between 2 and 5 times more than them all. The former effect is due to this benchmark having been configured to do extra processing at the start of each simulated hour.

Figure 3 illustrates this phenomenon for two hours simulated time runs. A few runs of up to five hours were attempted; there were no qualitative differences between the second and subsequent hours.

The causes for time variability between runs were almost entirely due to factors affecting the whole of the computation. The most likely causes are to cross-application

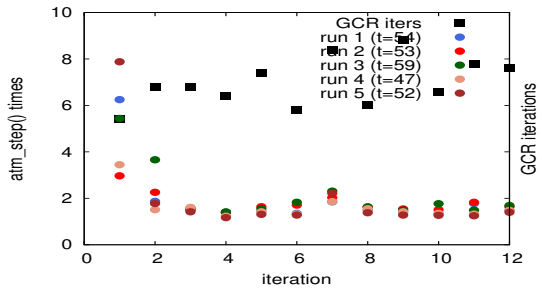

Figure 3. Variation in walltime (s) in consecutive calls to `atm_step()` for two simulated hours of an N320L70 benchmark on a 32×32 grid. Iterations in GCR solver are on scale from 0 to 50

contention [11] (other current jobs heavily communicating the same switch(es) as the UM job) and operating system-related causes (page coloring, NUMA effects and interrupts). Unfortunately, none of these effects was under our control at the time.

The `read_dump()` routine took between 25 and 29 seconds of the total; this routine scales negatively. Within `atm_step()`, the only routine to scale negatively was `q_pos_ctl()`[2], taking between 12 and 15 seconds of the total run-time (at 32 nodes).

The minimal job time for Figure 3 was $\approx 80s$; this can be extrapolated to give $\approx$ 260s for a 24 hour run, corresponding to about 75 CPU hours.

After modifying the UM internal profiler to record on which call to a particular routine was the most expensive, we saw that the first call was generally the most expensive when there was significant variations, and indeed this was the case with `q_pos_ctl()`. From code inspection, this routine used gather and scatter communications (and was the only routine to do so to a significant extent); we conjectured that the first call had extra overhead in setting up OpenMPI connections. The second possible reason was that upon the first iteration, memory-mapped pages in physical memory

[2]This routine corrects the moisture content field, which can become negative from previous calculations. Conservation of the overall level of moisture should be preserved, especially for climate simulations.

from the previous job were being flushed

We repeated the tests, adding the OpenMPI flag `-mca mpi_preconnect_mpi 1` and using the `memhog` utility at the start of the job to flush out out memory-mapped pages before `u_model()` was called. The first iteration was now only within a factor of two of the average, with the impact of `q_pos_ctl()` reduced by a factor of about one half.

Finally, the effect of the variation between non-hourly iterations of `atm_step()` deserves some comment. The most obvious cause is differences in the number of iterations required for the Helmholtz solver. These are listed in the bottom row of Figure 3 (they are the same for every run); they correlate only weakly to the observed times per call.

V. Profiling Methodology

From the above experiences, and taking into account the resource limitations for our project, we designed a profiling methodology based on the following principles:

1) The profiling infrastructure should have minimal effect on runtime.
2) The profiling be as efficient as possible, i.e. consume a minimum of resources while providing an accurate prediction of long-term simulations.
3) The methodology should take into account (as far as possible) the variability of the `vayu` system.

Our methodology is thus to:

- take at least 5 timings for each configuration (process grid and any other tunable parameters). The minimum will generally be taken as the true timing, as it represents the run with the least interference. When varying parameters of the simulation, a single run from each setting is submitted to the cluster and repeat runs do not start until the previous has completed. The rationale behind this is that, if there comes a 'quiet' period, all settings will have a chance to run under those conditions.
- examine the scalability of the major subroutines and estimate the degree of load imbalance.
- run the simulation for a minimal number of timesteps in order to be able to project the performance of the target run-time. This was done by dividing the simulation into an initial period (reading dump file and first 6 iterations) and a 'warmed period' (the next 12 iterations).
  The warmed period should be over an integral number of simulated hours, in order to take into account the hourly variability observed in Section IV. Two hours appear sufficient to smooth out the effects of other sources, such as the variability of the execution time of individual timesteps.
- reduce the overhead of profiling, and measure (or estimate) its extent. This is particularly important over the 'warmed period' which is used to make the projections.
- individually time each iteration within `u_model()`, for later analysis.

Accordingly, we modified the UM internal profiler to measure the warmed and total periods. To reduce overhead (in the warmed period), the global synchronizations invoked when a timer started were limited to those routines which exceeded a certain threshold of the fraction of walltime in the initial period. We chose a threshold of 3%, which resulted in a factor of approximately ten in the reduction of the number of synchronizations. This number was recorded, and at the end of the simulation, a number of consecutive synchronizations were timed in order to get an estimate of the synchronization overhead. The load imbalance estimate for the warmed period was then calculated as the averaged time (over each process) spent in synchronizations during that period, minus the estimated overhead.

As a defensive programming measure, total times for the execution and the warmed period were independently recorded. These correlated to within less than 1% of the sum of the non-inclusive timers.

It turned out that suppressing synchronizations for certain functions was far more complicated to implement efficiently than it at first appeared. This is because some processes invoke timers for subroutines (e.g. polar filtering) that other processes do not. A compromise was to let process 0 decide which timers should be suppressed, and this information was broadcast to other process at the beginning of the warmed period.

VI. Scalability Analysis

This section gives profiling and scalability analysis for the UM on both the N320L70 and N512L30 grids.

`q_pos_ctl()` was identified in Section IV as being a bottleneck; rather than attempting to optimize it, we subsequently obtained from the MetO the recent Parallel Suite 24 (PS24) patches, which had addressed this problem. These were then merged into the UM 7.5 codes. The benchmarks were configured to select an algorithm for `q_pos_ctl()` which conserved the moisture field for the same level; this avoided some of the expensive collective communications in the original code. The codes were built under the conditions described in Section IV.

The original and PS24 codes both used the same dump files for both the N320L70 and N512L70 benchmarks. Each timestep corresponded to 12 simulated minutes in both cases. However, it was observed that the PS24 codes did not have any periodic pattern in the time taken for each timestep. This is unlike the original codes in which every sixth timestep took appreciably longer, as described in Section IV-A.

Figure 4 gives the speedups corresponding to the N512L70 benchmark on the PS24 codes. The runs are over three simulated hours and generate no STASH output. The grid configurations were chosen to be near-square (either a 1:1 aspect ratio, or 1:1.5 where this was not possible). Assuming the degree of communication north-south and east-west is similar, a near square configuration should optimize

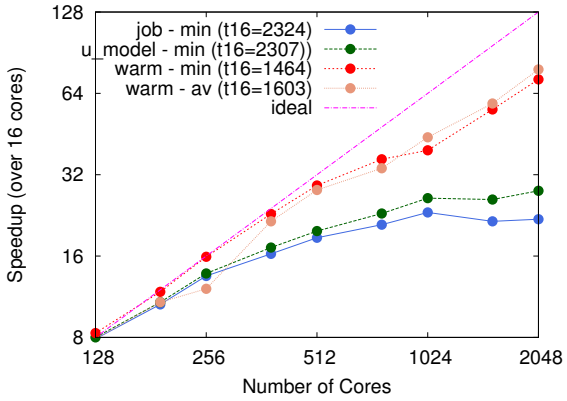

Figure 4. Scaling of UM 7.5 + PS24 patches on the N512L70 benchmark. 't16' is the time in seconds for 16 cores

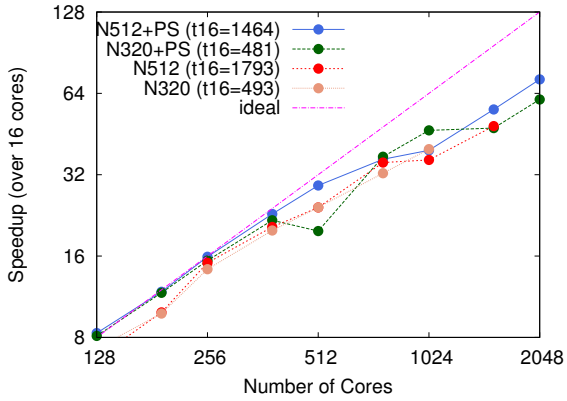

Figure 5. Scaling of the minimum 'warmed' time of UM 7.5 with and without PS24 patches on the N320L70 and N512L70 benchmarks. 't16' is the time in seconds for 16 cores

the surface to volume effects in the grid decomposition, and hence minimize nearest-neighbor communication.

For less than 128 cores, the scaling was linear or very slightly super-linear; to amplify scaling effects at higher cores counts, these results were omitted from Fig 4.

The job times, time for the call to `u_model()` and the warmed period times are given. The latter is given for both the minimum and average times. Despite there being a difference of at least 15%, due to factors described in Section IV-A, the averaged times show similar scaling behavior as the minimum. The time to call `u_model()` scales more poorly, due to the shortness of the simulated time period, and the effect of calling `read_dump()` and the 'warming' of the benchmark.

An analysis of the difference between the job `u_model()` times for 1024 cores (100s vs 87s) yielded the following breakdown: 2s to launch process 0, 4s to launch all processes, 6s to read the namelist files and 1s to exit UM and 'cleanup' standard output files. It can be noted from Figure 4 that this difference increases with increasing core counts.

Figure 5 gives the speedups corresponding to the warmed period for all four benchmarks. While all N320L70 benchmarks are still scaling above 1000 cores, the N512L70 benchmarks scale slightly better [3]. This is not surprising in the sense that the N512L70 grid will have a factor of 1.6 better surface-to-volume ratio than the N320L70. It can be seen particularly from the N320L70 curves that the PS24 codes scale better.

Table I shows the execution profile for the N512L70 benchmarks at 1536 cores for the 'non-inclusive timers' corresponding to the major subroutine. No routine scales perfectly; hence there is potential for worthwhile performance improvement in all. `PE_Helmholtz` is the dominant routine. The parts of `atm_step()` that are not part of the other listed functions also have poor scaling, with significant load imbalance. Considering that 96 is the ideal scaling at this point, the overhead of `SL_Full_wind` appears to be almost entirely in load imbalance. It is also high in `Convect` and `SF_EXPL`; this probably due to latitudinal variations, which is difficult to avoid with a fixed decomposition.

[3]The missing points for the non-PS24 codes above 1500 cores are due to bus errors.

Table I
N512L70 SUBROUTINE WALL-TIME PROFILE WITH $32 \times 48 = 1536$ CORES. 's' IS THE SPEEDUP (OVER 16 CORES), '% t_w' IS % OF WARMED PERIOD TIME, '% IM.' IS FRACTION OF TIME DUE TO LOAD IMBALANCE

function	UM7.5			UM7.5+PS24		
	s	% t_w	% im.	s	% t_w	% im.
`PE_Helmholtz`	59.4	51	0	61.6	48	0
`atm_step`	15.1	11	27	11.4	25	20
`q_pos_ctl`	0.8	12	0	5.4	0	10
`SL_Full_wind`	50.8	10	47	49.1	13	46
`SL_Thermo`	52.7	6	20	53.3	9	19
`Convect`	28.0	9	54	66.2	5	37
`SF_EXPL`	6.7	6	69	35.4	1	75
Atmos_Physics2	12.9	5	71	52.9	1	28
: total:	52.1	100	17	55.8	100	30

`read_dump()` at 1536 cores on the N512L70 grid has an execution time of 24s, corresponding to an inverted speedup of 0.18! It has an absolute execution time of $\approx 25s$. While it is only called once per simulation, it is also a target for optimization.

Overall, load imbalance effects are quite significant (recall that our methodology gives an underestimate of the real load imbalance). It can also be seen that the PS24 patches have significantly better performance in `q_pos_ctl()` and `SF_EXPL`.

Figure 6 shows the load-imbalance distribution of processor nodes over hours 2 and 3 of an N320L70 simulation

```
666666666666666666666666
999999999999999999999999
888999999999999999999999
888988889999999888999999
888888888898887777778888
887777778888777777777788
887777777788777766677778
666666666677666555556667
222222223333322200002222
333322223344222233333344
555555556666666555556655
111122221222222200112222
000011223333333223333322
222223334444444444444443
555555666666666666666666
555555555555555555555555
```

Figure 6. Visualization of load imbalance for an N320L70 simulation on a 16×24 processor grid ('0' = 0%, '9' = 15% of run-time)

using the dump in Figure 1. The data is from time spent in MPI barriers in the updated timer module; thus a large value indicates the process had less work to do. Lighter loads over the high northern latitudes indicate both the effects of extra processing in the equatorial regions due to convection.

A. Methodology Evaluation

To evaluate the profiling methodology of Section V, some 24 hour simulations were performed on the both the N512L70 benchmark with PS24 patches, and on the N320L70 benchmark without the PS24 patches (recall that the later showed more variability in the time to execute a timestep). We chose our 'operational target' core count of 960 cores in a near-square aspect ratio.

Defining t to be the job time, $t_{2:24}$ to be the warmed time (after the 1st simulated hour), and $t_{2:3}$ to be the time for the second and third simulated hours, the formula $t' = t - t_{2:24} + 11.5t_{2:3}$ gives the projected runtime from runs using the above methodology. The following table gives this data for three runs.

run	t	$t_{2:24}$	$t_{2:3}$	t'
N512L70	527	39.12	6.5	524.3
N320L701	224	163.8	13.7	214.8
N320L702	237	174.9	14.9	233.6

For the N320L70 run 1, there was an anomaly at timestep 55, taking 7.9s; on run 2, there was one at timestep 134 of 4.2s (the average timestep is about 1.2s). If these anomalies were removed, the above results shows that the timings for the second and third simulated hours are a quite accurate predictor of the full simulation time.

It was found that the overhead of the profiler (synchronization overhead, gathering data) was less than 1% of the warmed time period in all cases. The printing output phase began to take more than this for over 512 cores; however, this had no effect on the 'warmed time' measurements.

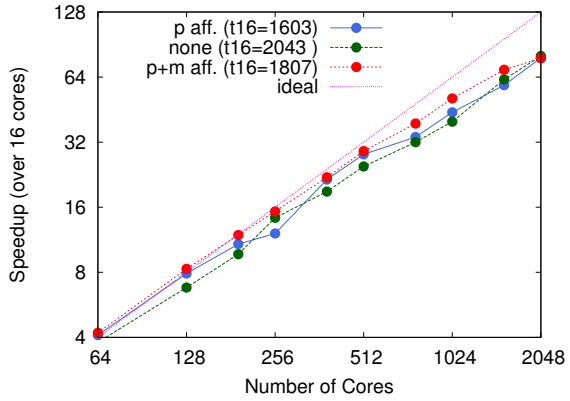

Figure 7. Scaling of averaged 'warmed time' with none, process and process+NUMA affinity for the PS24 codes on the N512L70 benchmark. 't16' is the time in seconds for 16 cores

B. Analysis of Process and NUMA Affinity Effects

The previous results used OpenMPI 1.4.1 with process affinity flags. A locally modified version of OpenMPI 1.4.3, with both process and NUMA affinity available (by default) became available after the data in the previous sections was collected. To analyse the effects of these factors on execution time and its variability, scaling measurements using the methodology of Section V was repeated for OpenMPI 1.4.3 and 1.4.1 (without process affinity). The results are shown in Figure 7.

Both kinds of affinity is overall the fastest; it is 18% faster (in minimum 'warmed' time) than the others at 2048 cores.

Figure 8 indicates the reduction in variability due to affinity, by taking the normalized error from the average $\frac{\Sigma_{i=1}^{n}|t_i - \bar{t}|}{nt}$. While the number of measurements ($n = 5$) are only sufficient to observe general trends, we can see that there is no clear correlation with the error with the number of cores. We can see also that process affinity reduces variability by 20%, but NUMA affinity reduces this further by a factor of 4!

VII. Tuning of the Model

This section describes some experiments in tuning the UM 7.5 codes, based on varying configuration parameters and compilation details.

A. Segment Size Parameters

The performance of the N96L38 benchmark[4] is qualitatively different from the larger benchmarks, since a 48-hour simulation (with each timestep representing 30 minutes) can

[4]For pragmatic reasons, UM version 7.3 was used for this benchmark. However, we expect no significant difference between the behavior of the relevant routines in UM 7.5.

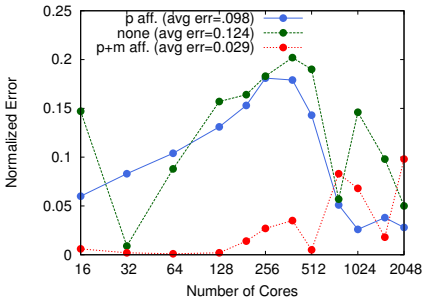

Figure 8. Normalized average error of the 'warmed time' with none, process and process+NUMA affinity for the PS24 codes on the N512L70 benchmark

Table II
SEGMENT SIZE PARAMETER SETTINGS FOR THE N96L38 BENCHMARK

subroutine	setting		improvement	
	def.	opt.	% subr.	% total
short-wave radiation	80	32	14	0.3
long-wave radiation	80	25	25	1.2
convection	80	16	25	2.2

be done in under 5 minutes using only 16 cores. Of more interest is improving computational performance, since this resolution is typical of those used for climate runs which are used for simulations in the order of years.

One of the few performance-related parameters provided by the UM are segment size parameters for three of the atmospheric physics functions, which controls the sizes of segments of atmospheric columns which are processed together. Their values affects memory hierarchy performance and by default are tuned for an IBM Power 6 system.

Table II gives the performance improvement of tuning these parameters for a 2×8 process grid. These are based on the average of 5 runs. These routines originally accounted for 21% of the total run time; the tuning resulted in a relatively modest total reduction of 3.7%. For a 2×4 grid, we obtained identical settings, with similar improvements in walltime.

Figure 9 shows the tuning of these parameters, with again averages taken over 5 runs. Our methodology was to utilize the UM internal profile described in Section II-B, and uses the same segment size for all three parameters on each run. It may be noted that once the parameter size increases over 16, performance is relatively insensitive to the value of these parameters.

For the N320L70 benchmark using the PS24 patches, the effect of all of these routines is however much smaller, accounting for 6–8% of the run time for core counts ranging from 16 to 1536. An experiment on an intermediate core count (384) showed a 1% improvement in the warmed period time with an optimal parameter setting of 16 in all three cases. To conserve machine usage, settings were limited to

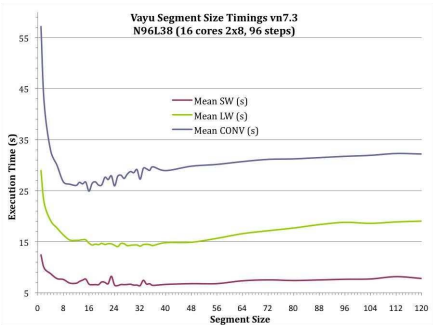

Figure 9. Segment size parameter tuning of the N96L38 benchmark

multiples of 8 up to 80. The minimum of 5 timings (over the warmed period) was used for this result.

B. Process Grid Configuration

The results presented in Section VI used near-square process grid settings. The following table gives the execution time in seconds for various process grids with 480 cores on the PS24 patches on the N512L70 grid:

grid:	10×24	12×20	15×16	20×12	24×10
t (s):	100.1	97.0	97.4	98.0	98.2

and those for 960 cores are given in the following table:

grid:	16×60	20×48	24×40	30×32	40×24	48×20
t (s):	29.6	30.2	28.8	27.6	27.7	28.0

As before, these results were for 3 simulated runs with the minimum timing being given. The above results show that the near-square configurations give slightly better performance.

C. I/O Performance

As noted in Section VI, the `read_dump()` routine took a significant amount of the execution time, even compared with a full simulation run. Part of this inefficiency was traced to the C byte-swapping code which is required for I/O on a little-endian machine (the MetO mainly test the UM on IBM Power machines where this code is not used). The default configuration from UMUI will have this code compiled without optimization.

Simply compiling the system with `icc -O3` rather than the default `gcc` with no optimization reduced `read_dump()` execution time by 42% at 16 cores (the absolute value of the improvement remained constant at higher core counts).

With STASH output on, the endian swapping has a dramatic effect on total benchmark performance. For a 24 hour simulation on 800 cores on the BoM cluster (virtually identical to `vayu` except having a faster Infiniband switch), compiling with `icc -O3`, using a x86-optimized assembler byte-swap routine and avoiding endian conversion altogether

improved total execution time by 42%, 58% and 58% respectively. This brought the simulation time down to 1085s.

VIII. Related Work

One of the few papers in the open literature to analyze the performance of the UM is [12]. However, this describes the very first parallelization schemes, almost two decades ago, and the results do not reflect the current state of the UM. Also, it concentrates on the data assimilation rather than the weather simulation aspects of the UM.

The UM developers at the UK Met Office produce internal reports in the profiling, and scalability of the UM. These are made available to the UM community. A scalability analysis of UM 7.6 with the PS24 patches is made in [4]. The N512L70 benchmark over three simulated days with STASH output off scaled up to 1536 cores, achieving a speedup of 7.7 over 96 cores; this was on an IBM Power 6 cluster. With STASH output on, it scaled to 1024 cores, with a speedup of 5. For climate simulations, with the UM coupled to the NEMO ocean model by the OASIS3 coupler, the N216L85 model scaled to 18 cores, yielding a speedup of 1.5 over 6 cores. A profile of internal subroutines using the notion of 'parallel speedup' as described in Section II-B was also given; these showed `q_pos_ctl()` still scaled the most poorly. However, the results do not show how important to the run-time the poorly scaling subroutines were.

An earlier report studies the effect of using OpenMP within UM 7.4 [5]. Results on an N512L70 benchmark on a 2-core Power 6 system with 2-way hardware threading that utilizing 2 threads per MPI task with two MPI processes per core (to take advantage of hardware threading) gave 10% improvement over pure MPI (one process per core). This was for 2 nodes; the advantage steadily reduced to 2% at 64 nodes.

There are a number of papers on profiling and performance tuning of other atmospheric models. [13] shows linear scaling up to 24 nodes on a dual-socket quad-core Opteron and Infiniband cluster; this is in terms of the number of (large) WRF jobs completed per day. MPI profiles of the distribution of the number of messages per message size intervals (for various core counts up to 64) were given. A comparison between OpenMPI 1.3 and HP MPI 2.2.7 showed linear scaling up to 24 nodes, with OpenMPI having slightly better scaling and generally 10% greater overall performance.

A more detailed scaling analysis of WRF is given in [10]. The benchmarked used a $1500 \times 1201 \times 35$ regional US grid run for 240 timesteps. The performance modelling tool IPM was used to give separate scaling analysis of computation vs communication, with the former scaling super-linearly due to "algorithmic limitations", and the latter scaling sub-linearly. The larger part was due to nearest-neighbor communication in the 'halo exchanges', differing from our result of the majority of the time being spent in barriers and other collectives due to load imbalance.

CAM is a widely used atmospheric model with a similar development history to the UM [14]. It also uses a rectangular grid but has more flexible methods for decomposing data over processes (including decomposing the vertical dimensions), and supports hybrid MPI / OpenMP parallelism. It also has a general 'chunking' scheme for blocks of atmospheric columns, providing a more generic solution to the segment sizes for the UM described in Section VII-A. The runtime selection of a variety of chunk load balancing schemes and communication protocols (e.g. different MPI collective and point-to-point algorithms) is also available. [14] presents a methodology to efficiently traverse the large parameter space and find an optimal set of values. Results are given for a variety of systems using a benchmark with a $256 \times 128 \times 26$ grid. The main result on a IBM p90 cluster represented a speedup of 12 on 512 cores over 16 cores. The metric used was the number of simulated hours simulated per day. Their chunk size results are similar to ours in Section VII-A, finding an optimal chunk size of around 16 for a variety of platforms, and the run-time largely insensitive provided the chunk size is greater than 10. Load balancing scheme selection can improve runtime by 10% in some instances.

However, in all of these works, the results were obtained over long simulation runs, representing a very large amount of compute resources required to generate them.

IX. Conclusions and Future Work

The Unified Model has a highly complex code base with large and expanding user groups around the world. Atmospheric simulations with large resolution grids using the UM is a very large computation. Current profiling tools have difficulty in dealing with these simulations.

Even on a highly timeshared cluster with a high variability between the execution time for identical jobs, it is still possible to accurately and efficiently profile the UM on large benchmarks by using an extended internal subroutine profiler. By reducing synchronizations within the profiler, it is possible to reduce profiler overhead to a negligible extent, while retaining the ability to estimate load imbalance. This is based on the idea of using small parts of the simulation (after 'warming') to project the run-time for a longer simulation. Care has to be taken to ensure the period is representative, due to effects such as extra work being done at periodically-spaced timesteps. Care also has to taken to remove one-time effects, as these lead to a pessimistic projection of the scaling behavior.

The UM 7.5 codes scale well up to 2048 cores for the N512L70 benchmark, with slightly lower scaling for the smaller N320L70 benchmark. The PS24 patches improved two subroutines which were scaling poorly; however, there remains 6 subroutines in which optimization efforts are

worthwhile (potential to improve 2% or more the overall runtime). However, three of these suffer largely from load imbalance, and might require widespread changes, such as dynamic load balancing across the north-south direction. As noted in Section VI, the codes with PS24 patches do not exhibit any periodic behavior per iteration, and it is possible to to use a more efficient methodology (i.e. 1.5 hrs simulated time with a 1 hr simulated time warmed period).

Performance tuning opportunities have largely arisen from the fact that the UM has been tuned for for Power 6 based platforms. Segment size parameters may be tuned to bring modest improvements for the N96L38 data grid, currently used for climate simulations. For larger data grids, the proportion of time spent in routines affected by these parameter is reduced, and the advantage becomes negligible. For moderate to large core counts, near-square processor grids give slightly better performance. I/O performance is significant; optimizing or avoiding the associated byte swap operation gives dramatic performance improvement when STASH output is enabled. OpenMPI process and NUMA affinity parameters have also been shown to have a very significant affect, in both improving performance and reducing the variability of measurements.

Future work includes extending the UM internal profiler to use hardware event performance counters in order to understand better the memory performance of the codes, which is also expected to have a large impact on performance. Obtaining a profiling of MPI subroutines at large core counts is also needed, in order to understand the time spent in communication (as opposed to load imbalance). We plan to combine IPM with our extended internal profiler, using the IPM regions facility to get MPI and hardware event performance counters breakdown across the major subroutines. A more detailed investigation into the constituent components of the dominant Helmholtz solver is also planned.

The results of Section VII-C indicate we are already within a factor of two of the 'operational target' with the N512L70 benchmark. Apart from waiting for a cluster based on the Intel Sandy Bridge processor (for which the target is intended), the most promising opportunity for optimization is in using mixed OpenMP / MPI parallelism, and improving the latter within each nodes. This should reduce MPI communication, due to having fewer MPI processes. The more detailed profiling methodology mentioned above would be helpful in predicting whether this has any benefit.

ACKNOWLEDGMENTS

The authors thank Les Logan for preparing the UM 7.5 benchmarks. We thank Les and Martin Dix for technical advice on the UM. We also thank David Singleton, Robin Humble and Judy Jenkinson for technical support on the `vayu` cluster. In particular, to David Singleton for adding NUMA affinity to the local version of OpenMPI 1.4.3. We thank Paul Selwood for performance advice on the UM and for making available the PS24 patches. We finally thank the NCI National facility for the usage of the `vayu` cluster.

REFERENCES

[1] T. Davies, M. J. P. Cullen, A. J. Malcolm, M. H. Mawson, A. Staniforth, A. A. White, and N. Wood, "A new dynamical core for the Met Offices global and regional modelling of the atmosphere," *Q. J. R. Meteorol. Soc*, vol. 131, pp. 1759–1782, 2005.

[2] The Unified Model Collaboration. [Online]. Available: http://www.metoffice.gov.uk/research/collaboration/um-collaboration

[3] R. Buizza, D. S. Richardson, and T. N. Palmer, "Benefits of increased resolution in the ECMWF ensemble system and comparison with poor-man's ensembles," *Quarterly Journal of the Royal Meteorological Society*, no. 589, pp. 1269–1288, 2003.

[4] A. Malcolm, M. Glover, and P. Selwood, "Scalability of the Unified Model," UK Met Office, Tech. Rep., Jun. 2010.

[5] M. Glover, A. Malcolm, and P. Selwood, "OpenMP scaling tests," UK Met Office, Tech. Rep., Nov. 2009.

[6] (2010) The Australian Community Climate and Earth-System Simulator. [Online]. Available: http://www.accessimulator.org.au/

[7] S. Milton, A. Arribas, M. Bell, and R. Swinbank, "Seamless Forecasting," UK Met Office, Tech. Rep., Nov. 2009.

[8] (2010) The NCI Cluster vayu System Details. [Online]. Available: http://nf.nci.org.au/facilities/vayu/hardware.php

[9] UK Met Office, "Unified Model: Basic User Guide (version 7.3)."

[10] N. J. Wright, W. Pfeiffer, and A. Snavely, "Characterizing Parallel Scaling of Scientific Applications using IPM," in *Proc. of The 10th LCI International Conference on High-Performance Clustered Computing*, Mar. 2009.

[11] D. Skinner and W. Kramer, "Understanding the causes of performance variability in HPC workloads," in *In International Symposium on Workload Characterization*. IEEE, 2005, pp. 137–149.

[12] B. J. N. Wylie and P. D. Surry, "Meteorological Data Assimilation: Migration to Massively Parallel Systems," in *Proc. of Scalable Computing: Cray User Group Conference*. Montreux: Cray Inc, 1993.

[13] G Shainer et al, "Weather Research and Forecast (WRF) Model: Performance and Profiling Analysis on Advanced Multi-core HPC Clusters," in *Proc. of The 10th LCI International Conference on High-Performance Clustered Computing*, Mar. 2009.

[14] P. H. Worley, "Benchmarking using the Community Atmospheric Model," in *Proc of the 2006 SPEC Benchmark Workshop*, Jan. 2006.