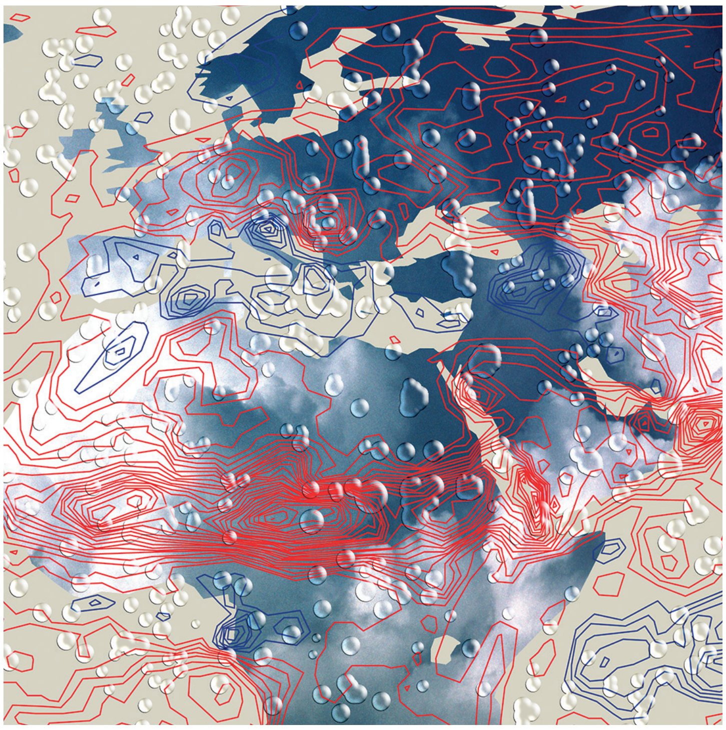
ECMWF Feature article

from Newsletter Number 109 – Autumn 2006

COMPUTING

New features of the Phase 4 HPC facility



www.ecmwf.int/en/about/news-centre/media-resources

doi:10.21957/ck97xpe284

This article appeared in the Computing section of ECMWF Newsletter No. 109 – Autumn 2006, pp. 32–36.

New features of the Phase 4 HPC facility

Neil Storer

A earlier news item in this newsletter entitled *"The new IBM Phase 4 HPC facility"* starting on page 5 described the new IBM Phase 4 system that is currently being installed and commissioned at ECMWF. The aim of the current article is to look in more technical detail at three new features of this system: simultaneous multi-threading, mid-size memory pages and multi-cluster GPFS.

It is important that users of ECMWF's HPC facility are aware of these new features as it will allow them to gain the full benefits of the new system. For example, effective use of Phase 4 will allow a higher percentage of peak performance to be sustained.

Simultaneous multi-threading (SMT)

SMT is probably the most noticeable new feature of the POWER5 micro-architecture; around 20% of the additional transistors on the chip (over and above those on the POWER4 chip) are dedicated to this feature. SMT enables two threads to execute concurrently on a single processor (physical CPU), i.e. in any given CPU cycle it is possible for each of the two threads to execute an instruction on that processor. The effect is to make every processor look as if it were in fact two virtual CPUs. This in turn enables the processor to be kept busy for a larger proportion of the time, thus producing a higher *sustained* performance than would otherwise be the case. However, because only certain parts of the hardware (such as the program counter, instruction buffer and rename registers) are duplicated, but none of the functional units, the peak performance of the processor is not affected by SMT. The processor can intelligently assign priorities to the two threads to dynamically balance access to the shared resources, thus ensuring that one thread does not "hog" these thereby starving the other thread of CPU cycles.

It is difficult to gauge the benefit in performance that SMT brings. Some applications, in particular those that may be limited by being functional-unit bound, or memory-bound, may not see any benefit from using SMT. It is possible to switch the chip into single-threaded mode and by doing so allocate all of the rename-registers that are shared in SMT mode to that thread. In theory this should enable some applications that do not benefit much from SMT to achieve higher levels of performance, but in practice this does not appear to be the case for any of the applications that have been tested in this way at ECMWF.

A p5-575+ server can be run either with SMT functionality enabled or with it disabled. When SMT is disabled, each of the 16 processors can execute only a single thread, and as such, at any instant in time, each thread is guaranteed to run on a processor which is executing no other threads. When SMT is enabled, each of the 16 processors can execute two threads concurrently and as such, at any instant each thread may be sharing a processor with another thread (possibly from a different, unrelated job). In the first instance the operating system has 16 virtual CPUs on which to schedule threads for execution, while in the second instance it has 32 virtual CPUs.

ECMWF has decided that all of the p5-575+ servers in the two Phase 4 clusters will be run with SMT enabled at all times. This simplifies the operation of the clusters and even when there are only 16 threads associated with user jobs in a node, the additional virtual CPUs can be used by the operating system and system daemons that need to execute from time to time; in this way they do not have to interrupt user threads by switching execution context in order to gain access to the processor, reducing the effect of what is termed "operating system jitter".

Use of SMT by parallel applications

Parallel applications can make use of SMT in several ways. Take for instance a parallel application that has 64 threads (MPI tasks, or a mixture of MPI tasks and OpenMP threads) which runs using 64 processors on one of ECMWF's POWER4+ clusters using 2 nodes with 32 threads/node on that cluster. One can envisage several scenarios for running this on the POWER5+ cluster. Consider three scenarios and their impact on running two NWP applications from Member States.

1) ***2 nodes, 32 threads/node.*** If this were to run on the same number of nodes of the POWER5+ cluster it would still be using 64 CPUs, but this time they would be virtual CPUs. That is the application would be making use of SMT (2 virtual CPUs per processor) so that it would only be using 32 processors (not 64 as it did on the POWER4+ system). With half of its threads sharing the processors with the other half, it is likely run slower than on the POWER4+ cluster. However the slowdown may be much less than one would surmise and in fact in some cases it might indeed run faster. We have seen NWP applications run in this way on the POWER5+ system that have taken much less than 10% longer to run than on the POWER4+ system.

2) ***4 nodes, 16 threads/node.*** Another way of running this application on the POWER5+ cluster would be to go back to using 64 processors but not using the SMT feature. This would almost certainly run faster than on the POWER4+ cluster, mainly owing to improvements in the memory access capability of the POWER5+ system and the fewer interruptions to the execution of application's user-threads needed in order to accommodate kernel and daemon access to the processor. Nevertheless, in this instance, no use would be made by the application of the additional virtual CPUs. In this respect the application would be wasting CPU cycles that it could otherwise make use of. The NWP applications ran between 1.4 and 1.6 times faster than on the POWER4+ system.

3) ***4 nodes, 32 threads/node.*** Yet another way of running the application on the POWER5+ cluster would be still to use 64 processors (4 nodes), but this time doubling the number of user-threads, to use all 32 virtual CPUs in each node. The gain in doing this mainly depends on the scalability of the application, but should in general be greater than that using the second paradigm. The NWP applications ran between 1.6 and 1.8 times faster than on the POWER4+ system.

Table 1 summarises these scenarios for the two NWP applications.

The question of how much gain an application will experience using SMT is a difficult one. In general this question can only be answered empirically by running the application twice, once using SMT and once without. For the two NWP applications one can see a gain of over 10% in using scenario (3) over scenario (2); for the IFS model gains in the region of 20% have been seen. Besides the wall-clock time taken to run the application, the user should also take into consideration the relative cost of the two runs (see the next section).

Application	System	Scenario (SMT use)	Nodes used (processors)	Threads per node	Total user-threads	Wall-clock time (seconds)	Gain over POWER4+
X	P4+		4 (128)	32	128	196	
	P5+	1 (yes)	4 (64)	32	128	210	-7%
		2 (no)	8 (128)	16	128	139	1.41x
		3 (yes)	8 (128)	32	256	122	1.61x
Y	P4+		2 (64)	32	64	1950	
	P5+	1 (yes)	2 (32)	32	64	2022	-4%
		2 (no)	4 (64)	16	64	1185	1.65x
		3 (yes)	4 (64)	32	128	1076	1.81x

Table 1 The performance of two specific applications (labelled X and Y) under different scenarios. These applications are NWP models developed by two of ECMWF's Member States.

SMT and job-scheduling

The ways that users can make use of SMT inevitably complicates matters for job-scheduling, particularly since certain features in LoadLeveler (IBM's batch scheduler) have yet to be fully implemented. In order to cater for the various scenarios some ground-rules have been set for the Phase 4 clusters. It is clear that jobs which prevent resources from being fully utilised should pay for those resources, so for the Phase 4 system ECMWF's LoadLeveler job-submit filter and charging algorithm have been modified to take this into consideration. The rules are as follows.

1) Any serial or parallel job that a user submits with the LoadLeveler directive:
   #@ node_usage = not_shared
   will not share the node(s) allocated to it with any other job. Also, regardless of the memory requirement specified by the job, it will be given all of the available memory in the nodes. Furthermore, regardless of the CPU requirement specified by the job, it will be accounted as if it used all of the processors in the node(s), since no other jobs can make use of any CPUs (virtual or physical) that it does not actually use.

2) Any serial job that specifies the LoadLeveler directive:
   #@ node_usage = shared (the default)
   or does not specify:
   #@ node_usage = not_shared
   will run in a node where up to 31 other serial jobs may be executing. This job will use a virtual CPU and may end up from time to time sharing a processor with other serial jobs in the node and consequently may have inconsistent wall-clock times each time it executes.

3) Any parallel job that uses more than one node will be forced to run as if the user had specified the directive
   "#@ node_usage = not_shared"
   and regardless of the memory requirement specified by the job, it will be given all of the available memory in the nodes. In this way, if a user decides to execute only 16 user-threads on a node, the 16 remaining virtual CPUs will not be allocated to any other job and consequently its wall-clock time should be consistent each time it executes. Regardless of the CPU requirement specified by the job, it will be accounted as if it used all of the processors in the nodes, since no other jobs can make use of any CPUs that it does not use.

4) Any parallel job that uses a single node and specifies the ECMWF site-specific pseudo LoadLeveler directive:
   #@ ec_smt = yes (the default)
   or does not specify:
   #@ ec_smt = no
   will run in a node where up to 32 user-threads may execute. Such a job may end up from time to time sharing the processors assigned to it with other parallel jobs in the node and consequently may have inconsistent wall-clock times each time it executes.
   LoadLeveler does not recognise this (or any other) ECMWF site-specific pseudo-directive; it is interpreted by ECMWF's job-submit filter and is not passed on to LoadLeveler itself. Be aware that if users put such ECMWF pseudo-directives in jobs run elsewhere, Loadleveler running at that site will reject the job.

5) Any parallel job that uses a single node and specifies the pseudo LoadLeveler directive:
   #@ ec_smt = no
   will run in a node where at maximum 16 user-threads may execute concurrently on the 16 processors. In this way such a job will not share the processors assigned to it with any other jobs and consequently its wall-clock time should be consistent each time it executes. The job will be accounted at a higher cost than if this directive had been omitted, since virtual CPUs are essentially forced to be idle by this job.

The following rule has been implemented to try to take into account those jobs that ask for so many (but not all) resources (CPUs or memory) in a node that they effectively prevent other jobs from using what is left over.

6) Any job that specifies a requirement for more than 75% of the memory in a node (i.e. more than 18750 MB), or that specifies 29, 30 or 31 (virtual) CPUs in a node (or 15 CPUs if **"#@ ec_smt = no"** is specified) will be forced to run as if the user had specified the LoadLeveler directive **"#@ node_usage = not_shared"** and regardless of the memory requirement specified by the job, it will be given all of the available memory in the node(s).

The following case, which is based on actual runs of a CPU-bound job (a coupled ocean + IFS model), is an example of how accounting differs for the scenarios in rules (4) and (5).

- Four copies of an 8-way parallel job were run on a single node with the default: **"#@ ec_smt = yes"**.
- The same four jobs were then run on two nodes using **"#@ ec_smt = no"**.

The wall-clock times, CPU times and the relative billing units charged are shown in Table 2. It is interesting to note that 25% more CPU time was used in the case of **"ec_smt = no"** than in the case where **"ec_smt = yes"** was used. This is due to the way in which the system itself accounts for use of the processor when two threads are executing on it at any one instant, as opposed to when only a single thread is running on it.

As you can see in Table 2, the user was charged over 40% more for reserving virtual CPUs and not using them. This is because in the one case the four jobs occupied a single node for 62,350 seconds, while in the other case the four jobs occupied two nodes for 44,070 seconds (i.e. 88,070 "node-seconds") and the billing units express this ratio of the "node-seconds" used. The percentage will vary from job to job, depending upon their characteristics, but the message is clear - do not use **"#@ ec_smt = no"** unless it is absolutely necessary. This applies, for example, to time-critical work that would not finish in time if its threads were to share processors.

	Max user-threads per node	Nodes occupied by the 4 jobs	Wall-clock time (seconds)	Total CPU time (seconds per job)	Relative Billing Units
ec_smt = yes	32	1	62,350	247,100	1
ec_smt = no	16	2	44,070	309,200	1.414

Table 2 The extra cost incurred by not using SMT is illustrated in this example of two runs of an ocean model coupled with IFS; the first run uses SMT, while the second (more costly one) does not (though of course SMT was still enabled on the nodes on which it ran).

SMT and processor binding

To make the most out of SMT and memory affinity it is advisable to bind threads to CPUs. There are several ways to accomplish this, which are described in ECMWF's training material. It is difficult to give a general recommendation on the best CPU configuration to bind to, especially for applications using a mixed MPI and OpenMP programming paradigm; it mainly depends on the amount of work performed by each thread. In the same way as ascertaining how much gain an application will experience using SMT, the choice of which binding configuration to use is in general a question that can only be answered empirically by running the application with several configurations to see which best suits that application.

It is important that binding threads to virtual CPUs should only be done for those parallel applications that do not share a node. For applications that share a node it is difficult for users to know which CPUs to choose to bind to, because it is almost impossible to ascertain which CPUs the other applications running on the node are also bound to. For this reason it is essential not to bind threads to virtual CPUs for parallel applications that share a node.Rather let the operating system allocate the CPUs instead, otherwise it may happen that several applications overload the same processors within the node while other processors lie idle.

Table 3 summarises some experiments using a T399 resolution IFS model running a one-day forecast on 3 POWER5+ nodes (48 processors). These were performed early in the lifetime of the POWER5+ system. The IFS model is programmed using a hybrid MPI-OpenMP paradigm. In one set of experiments four OpenMP threads were used in each MPI task, while in the other set OpenMP was not used.

As you can see, the fastest wall-clock times were produced when using SMT with threads bound to virtual CPUs and memory affinity enabled. The best times – using all of the options (**Yes**) – were 26% (using OpenMP) to 33% (not using OpenMP) faster than using none of the options (**No**). Be aware that although the use of processor binding and memory affinity works well for the IFS model, it may not improve the performance of other codes; in fact there can be cases where it degrades performance, so this is another example of "try it and see".

OpenMP threads = 4					OpenMP not used				
MPI tasks	SMT use	Processor binding	Memory Affinity	Wall-clock (seconds)	MPI tasks	SMT use	Processor binding	Memory Affinity	Wall-clock (seconds)
12	**No**	**No**	**No**	**202**	48	**No**	**No**	**No**	**195**
			Yes	195				Yes	190
		Yes	No	193			Yes	No	187
			Yes	182				Yes	188
24	**Yes**	No	No	159	96	**Yes**	No	No	159
			Yes	155				Yes	155
		Yes	No	155			**Yes**	No	160
			Yes	**152**				**Yes**	**155**
Test run with 1 processor per node idle (*) →					90	Yes	No	Yes	163

Table 3 The cumulative effect of SMT, processor binding and memory affinity on the job wall-clock times of two configurations of the IFS model – one using OpenMP, the other not.

(*) *The run using 90 MPI tasks verified that leaving 1 processor idle on each of the 3 nodes was not beneficial for the IFS model, unlike some applications that do seem to benefit from this owing to the subsequent reduction in "operating system jitter".*

doi:10.21957/ck97xpe284

Page size considerations

On the POWER4+ clusters of ECMWF's old Phase 3 system the operating system (AIX 5.2) supports two sizes of pages - the normal default 4 KB small page-size and the 16 MB large page-size. It has been demonstrated that large pages can increase the performance of some applications, while for others there is little or no gain. However, on the Phase 3 system it was necessary to decide upon the number of large pages prior to booting the system, and following the system boot the number was fixed. The practical and operational difficulties in setting up and using large pages meant that for all intents and purposes they were impossible to use at ECMWF, and as such large pages were never made available for use by applications.

The POWER5+ clusters run the AIX 5.3 operating system and this provides better support for large pages. A system utility is available that enables the number of large pages to be altered dynamically while the system is running, rather than having to re-boot it. Even so, although operating system support for large pages is better than in AIX 5.2, it is still very poor (e.g. in many instances this utility can take a very long time to run and often fails to execute properly due to memory fragmentation). This in effect renders large pages unusable by applications at ECMWF yet again.

All is not lost however, because AIX 5.3 (64-bit kernel only) also supports 64 KB mid-size pages. These pages are not handled in the special way that large pages are, but are fully integrated into the operating system in the same way as small pages. Also, the gains that are seen by applications that make use of large pages are by and large seen if mid-size pages are used instead. This means that mid-size pages have all the advantages of large pages, without any of the disadvantages. There are options to the *"ld"* and *"ldedit"* commands to create "mid-size page" binaries, or you can over-ride the default mode of execution using the environment variable:

"LDR_CNTRL=DATAPSIZE=64K@STACKPSIZE=64K".

Barring unforeseen consequences it has been decided to set this by default on the Phase 4 system to ensure that all applications are able take advantage of the possible gains in performance that mid-size pages bring. The IFS model experiences a gain in performance of about a 3% when using mid-size pages, while gains in excess of 10% have been seen using other applications.

Multi-cluster GPFS

The disk space on the old Phase 3 system was split evenly between the two POWER4+ clusters. Each cluster had its own set of disks and filesystems, connected to its own storage area network (SAN). A cluster could only access the data on the other one using techniques such as FTP, RCP, RSYNC and NFS, all of which transfer the data using IP protocols over ECMWF's local-area network (LAN).

On the Phase 4 system this is done differently. All of the disks have been connected to the same SAN, which is also connected to both clusters. The SAN consists of two IBM "directors", which are highly-reliable fibrechannel switches, which enable any disk to be accessed by any I/O server on either cluster. This opens up the possibility of having filesystems that are shared between the two clusters. But rather than have the data transferred through the I/O servers of one cluster over the LAN then through the I/O servers of the other cluster, it can be read directly by the I/O servers of both clusters simply via the fibrechannel SAN. The software that enables this to be done is called the multi-cluster general parallel filesystems (MC-GPFS).

There are many advantages to using MC-GPFS, the main one being speed. Because the disks are connected via a SAN that is directly connected to both clusters, both clusters are able to read the same data at speeds comparable to those associated with disks that are directly connected to the cluster. While techniques such as FTP and such-like are limited to transferring data over the LAN at a few hundred megabytes per second maximum, data can be read via the SAN at a rate of several gigabytes per second. Other advantages include usability and less need for data replication.

With MC-GPFS no longer is it necessary for the user to use IP methods, such as those mentioned earlier, to access data on the other cluster; the data can be accessed as if it were local to the cluster on which his job is running. In fact for all intents and purposes there is no difference in this respect between data that is shared by the clusters in this way and data that is local to the cluster on which the job runs.

On the Phase 3 system certain filesystems needed to be replicated on both clusters so as to ensure that jobs could be submitted to either cluster without having to transfer files from one to the other, such as libraries, binaries and utilities. This replication of data brings its own set of problems. One has to ensure that both copies are kept in-sync and of course twice as much disk space is needed than would be the case for a single copy of the data. These problems vanish with MC-GPFS, because there is only one copy of the data in a single filesystem that is accessible by both clusters.

Having looked at the advantages, it has to be said that there are also some disadvantages. A filesystem that is shared by both systems can be seen as a single point of failure. If something happens to that filesystem, then both clusters are affected. Also, even though a filesystem can be shared by both clusters MC-GPFS has the concept of an "owning cluster". That is one of the clusters is responsible for managing the filesystem metadata, and should that cluster crash then the filesystem would become unavailable to the other one.

To alleviate these problems the MC-GPFS setup at ECMWF is as shown in Figure 1.

A third cluster (HPCI) has been installed, which is the owning cluster for the MC-GPFS filesystems. This cluster comprises four IBM p5-575 servers, and its equipment is set up in a highly redundant fashion. There is little data traffic between HPCI and the filesystems it "owns" and no user work is allowed to run on it. In this way it is expected that this owning cluster will be highly-available and is unlikely to suffer problems that systems running a varied mixture of user work tend to see. This will mean that the filesystems should always be available on both of the production clusters (HPCE and HPCF), and if one of the production clusters should crash, or needs to be taken for a system session, this should have no effect on the other production cluster. It is expected that judicious use of MC-GPFS will increase the productivity of the users of ECMWF's high-performance computing facility.

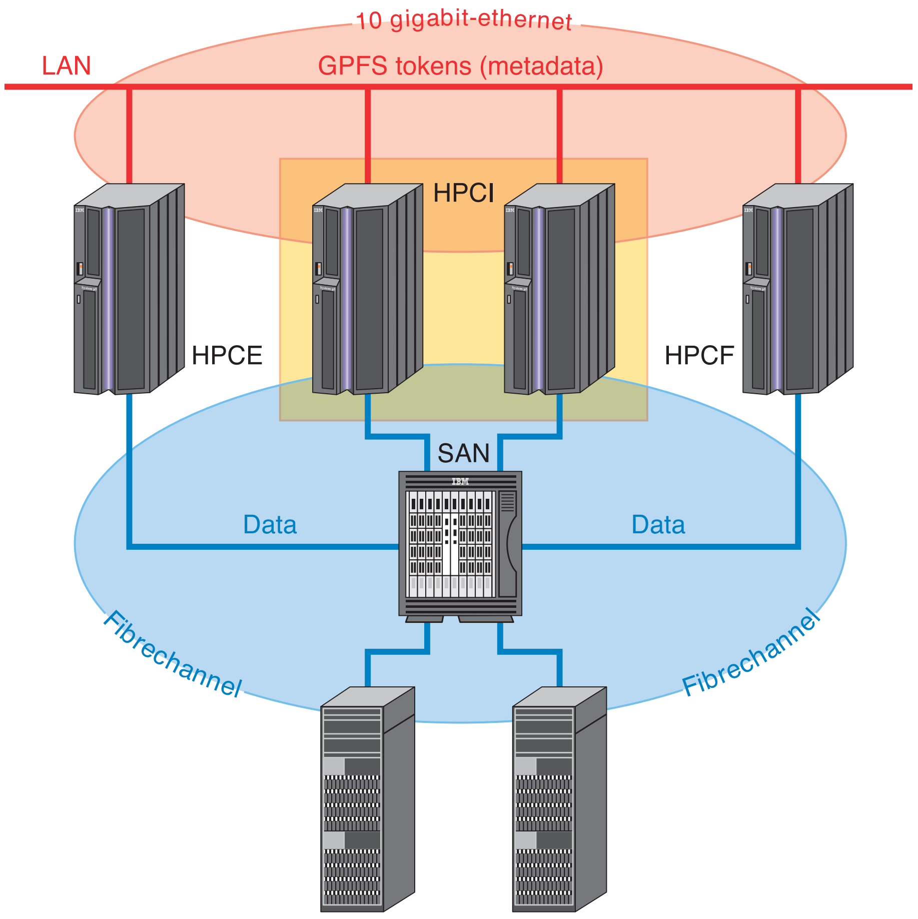


Figure 1 The MC-GPFS configuration on the Phase 4 system, showing the computational clusters (HPCE and HPCF), the "owning" cluster (HPCI), the fibrechannel storage area network (SAN) and the 10 gigabit-ethernet local area network (LAN).

Getting the benefits

There are several features of the Phase 4 system that differentiate it from the Phase 3 system. In particular SMT and mid-size pages will enable the system to sustain a higher percentage of the peak performance on users' codes. For the user there is a small price to pay – the concepts are slightly more complex to understand. However, even though ECMWF has tried to set up the system defaults in such a way as to hide some of the complication, users who take the time to understand the issues should benefit from that understanding.

Understanding the new features of Phase 4 and how to make best use of them has been a joint activity involving George Mozdzynski, Deborah Salmond and Oliver Treiber from ECMWF, John Hague from IBM, and members of the ECMWF User Support Section.

Further reading

For more detailed information on the POWER5 micro- and macro-architecture see "POWER5 and Packaging", the *IBM Journal of Research and Development (Vol 49, number 4/5)* at: http://www.research.ibm.com/journal/rd49-45.html