J90 Job Scheduling

Roger G Evans
CLRC Rutherford Appleton Laboratory,
Chilton, OX11 0QX, U.K.

r.g.evans@rl.ac.uk
G Timothy Folkes
CLRC Rutherford Appleton Laboratory,
Chilton, OX11 0QX, U.K.

g.t.folkes@rl.ac.uk
Michael Armstrong
SGI/Cray UK

mtja@cray.com

ABSTRACT:

We have reported at earlier CUG's on our needs to schedule a mixture of single CPU and parallel jobs on a J932. The technique using real time process scheduling has worked well in practice with the main problem being the overhead of moving the real time processes into low memory. We have investigated this effect further and believe that given the widespread needs to run job mixes similar to our own there is a need to reduce the system overheads of the present real time scheduling and memory management. The general need for this scheme of job management extends to other shared memory machines with a large number of CPUs and we hope that similar facilities will be available on future SGI/Cray machines with virtual memory management.

KEYWORDS:

Parallel, Scheduling, Batch, J90

Background

CLRC Rutherford Appleton Laboratory is one of two centres (the other being EPCC Edinburgh) providing supercomputing resources to the UK academic community. CLRC operates a J932 with 8 GByte of memory (4 GByte for most of the work reported here), 216 GByte of DS30 disks and 100 GByte of third party SCSI disks. User data is managed by DMF and by locally written Virtual Tape Protocol (VTP) software using respectively a STK4400 and IBM 3494. There are more than 1000 registered users, about half of them active and essentially all are remote from the Rutherford Appleton site.

The workload is varied and ranges from lattice QCD via chemistry and engineering to galactic modelling. The largest single group of users is the environmental modelling community with the UGAMP (Universities Global Atmospheric Modelling Project) being the largest single project.

Chemistry and engineering users are large users of commercial application software but for the majority of our user community the main application is the Fortran compiler! The current distribution of user demand is shown in Figure 1.

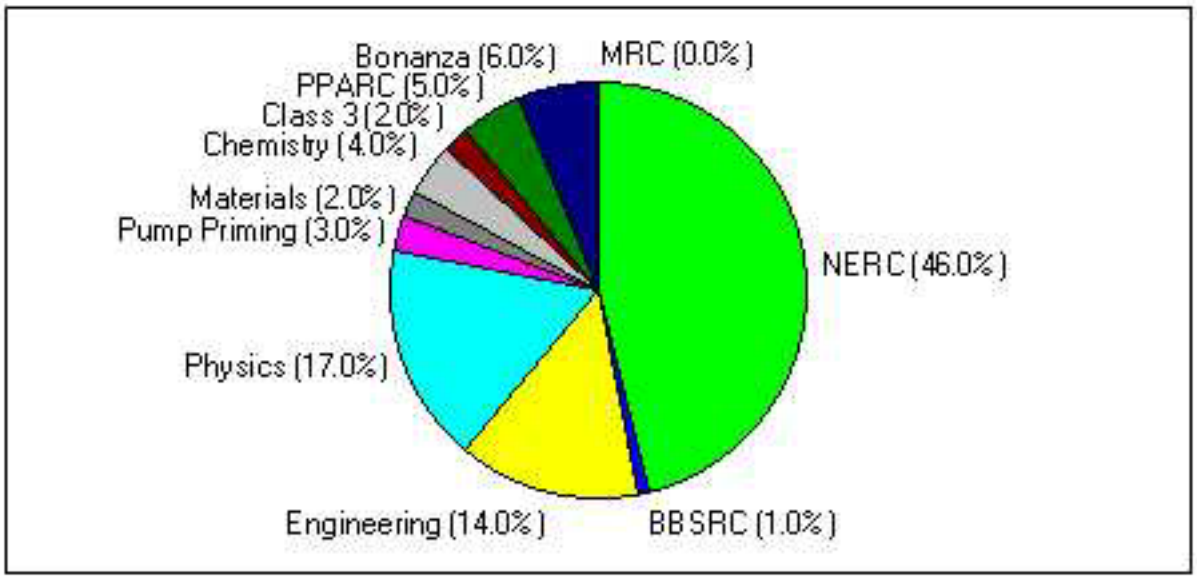

Figure 1 shows the distribution of J90 usage amongst programme areas.

Single Threaded and Parallel Jobs

The companion centre at EPCC Edinburgh operates a 512 node T3D for essentially the same body of academic users and the majority of highly parallel applications will of choice use the EPCC resources. The remaining user community is a mixture of users with modestly parallel Fortran applications (scaling to perhaps 16 CPUs), users of commercial applications such as Gaussian that scale to maybe 8 CPUs and code which is largely single threaded either because it is inherently serial, is transient in nature (eg it is used to analyse bulk data generated elsewhere, or is associated with a modest sized project where the human cost of developing a parallel code is not matched by the savings in computing costs.

For the single threaded jobs, the J90 single CPU performance is often little better than a local workstation and we are seeing a definite fall off in demand for our J90 resources. The response to a recent survey indicates the urgent need to offer computational performance at a national centre which is much greater than that available in UK university departments.

The J90 does become a significant computing resource when operated as a parallel machine and this is further enhanced by our recent memory upgrade from 4 GByte to 8 GByte. Our workload is thus a mix of legacy single threaded applications, applications requiring 4, 8 or 16 cpus and each job class has a spectrum of memory requirements. The user base is small enough that the character of the workload changes dramatically from week to week as groups become active, prepare for conferences and go on vacation.

NQS Scheduling

The standard NQS parameters and complexes manage most aspects of our job mix but we have been forced to write a NQS 'pre-processor' known locally as IQS (Input Queue Scheduler) to restrain the users who insist on submitting 40 jobs to the same queue at one time. Within each queue IQS operates a 'round robin' on each user ID so that a second user will get a job to run after the first job of the 'hog' user rather than after the 40th.

Parallel Jobs and 'gded'

It became particularly clear on our overloaded Y-MP8I that it was pointless to ask users to develop parallel applications to use most of the memory or temporary disk space only to run the job and have it run on one or two CPUs because the other users were busy running small jobs. In conjunction with Cray UK we looked at options for prioritising CPU allocation so that jobs in specifically parallel queues would have a high probability of receiving the number of CPUs actually requested. Initial attempts to do this by re-assigning nice values were not succesful and the

overall requirement is constrained by the lack of access to Unicos source code for J90 sites.

Eventually we developed a wrapper program known locally as *gded* which is an extension of the Unicos *ded* command for running small test jobs on all CPUs at the same time. *gded* runs its argument as a Unicos real time process having examined the NQS queue from whence it came in order to set the number of CPUs appropriately. *gded* operates on queues for 4, 8 and 16 CPUs selected via the *qsub -la multiN* option and is a null command in other NQS queues.

Previous Experience with *gded*

We reported at the last CUG[1] the initial results from *gded* which show that it met our requirements of delivering 90% of the CPU resource requested and operated fairly on a lightly loaded and heavily loaded machine. Our scheduler has run routinely for the last six months with a maximum of two 'multi8' jobs and has assisted in keeping the machine well loaded while delivering excellent application performance to the users. We feel confident in extending the range of multiN job mixes allowed to the users and have not encountered any serious problems in its use.

The limitations of *gded* are few: there is a question on memory management overheads as described in more detail below, and there is a loophole for users who still have old parallel jobs with calls to the multi-tasking (macro-tasking) library. Old multi-tasked code spawned a number of distinct processes and each process is then able to acquire NCPUS as identified within the *gded* environment. An old style macro-tasked code that initiates M processes can obtain M*NCPUS CPUs in an extreme case. We have one such legacy application on the J90 but its owner has been very restrained and has not abused the system. *gded* works equally happily with message passing parallel codes using MPI since these appear to Unicos as auto-tasked binaries.

Further Investigations of *gded* performance

We observed in the early test use of *gded* that when successive Multi8 jobs ran there was a slack period between the two jobs as shown in the CPU usage graph of Figure 1, and an increase in swapping as shown in Figure 2. It appeared that either there was a significant delay in starting a real time job or there were inefficiencies in memory management at job start up and shut down. The crude initial observations showed a period of several minutes of reduced CPU activity between two real time jobs and it was not clear *a-priori* whether this was start up or shutdown overhead or a mixture of both. We also speculated that inappropriate *segldr* settings could result in an excessive number of *sbrk* calls for these large memory programs and looked to a possible revision of *segldr* defaults for jobs in the MultiN queues to alleviate the problem.

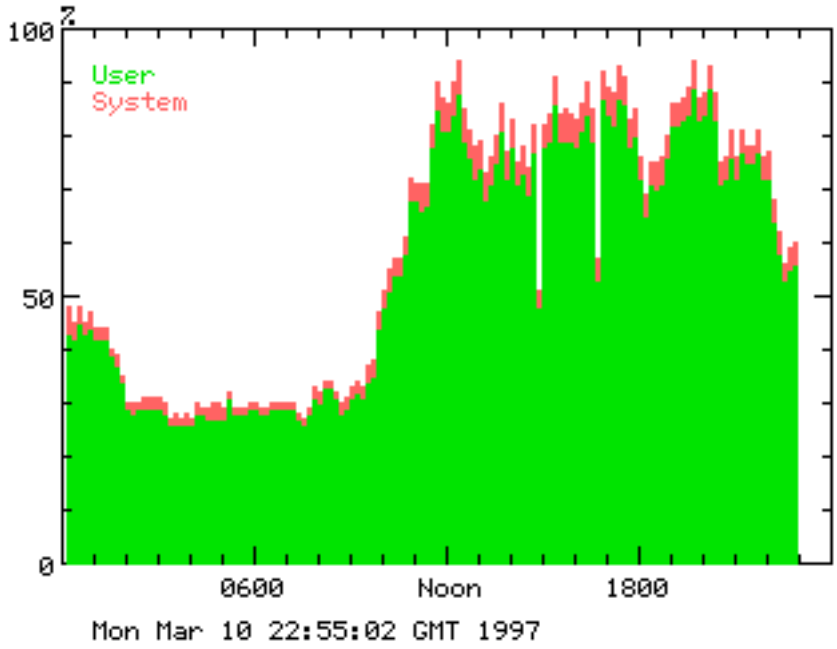

Figure 2 shows the low cpu activity between adjacent parallel jobs around 14.00 and 17.00 GMT

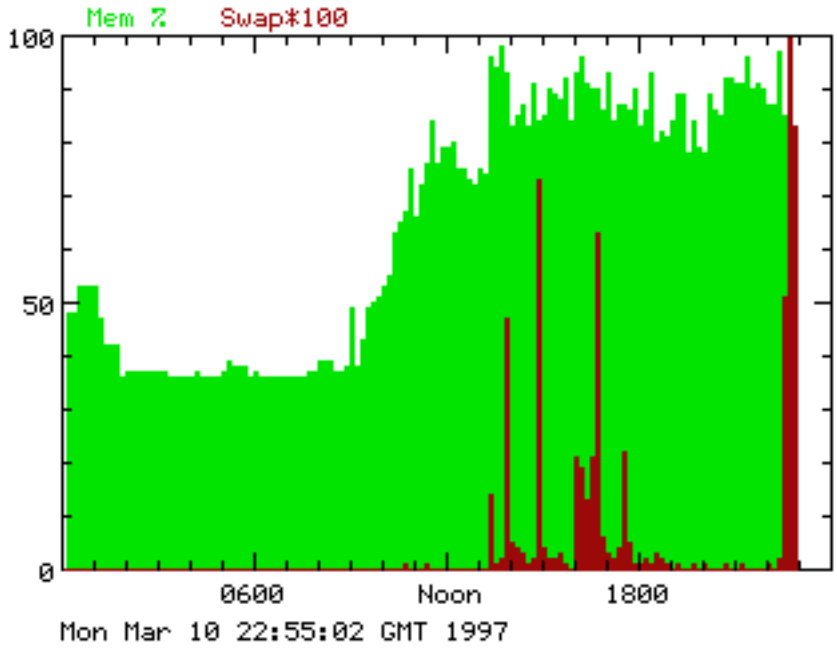

Figure 3 shows the increased swapping activity while the parallel jobs are running.

Since the last CUG we have looked more carefully at these overheads using a mixture of small and large memory jobs and running on a busy and on a near-empty machine. The application that we have used is a small (500 lines of Fortran 77) program that calculates coupled wave propagation in a relativistic plasma. It has the benefit of being well vectorised and parallelised and shows typically 7.8 times speedup on 8 CPUs. Any single threaded work at start up and shutdown is minimal. The memory size is altered simply by changing array sizes in space and frequency and scales from 4 MWord to 150MWord (which is the default limit for our Multi8 queue).

The test jobs are run as part of the normal batch queues and the script starts *sar* monitoring at 10 second intervals just before the executable starts. The benefit of getting the Fortran program to print its precise start and stop times was appreciated only after our J90 had been upgraded to 8GByte of memory and it was not possible to repeat some of the extreme cases.

The normal job accounting records give the effective number of CPUs delivered to the job and the total connect time. *sar -j* run on the accounting file gives the number of 'shuffle' requests for moving real time processes in memory. From the fully instrumented test program we observe that the 'shuffle' requests are coincident with the start and end of the Fortran program within a second or so. Accordingly we have used the *sar -j* results to indicate the system request to start the job and terminate its memory allocation and a comparison of this interval with the *ja* accounting of connect time gives us our measure of overhead.

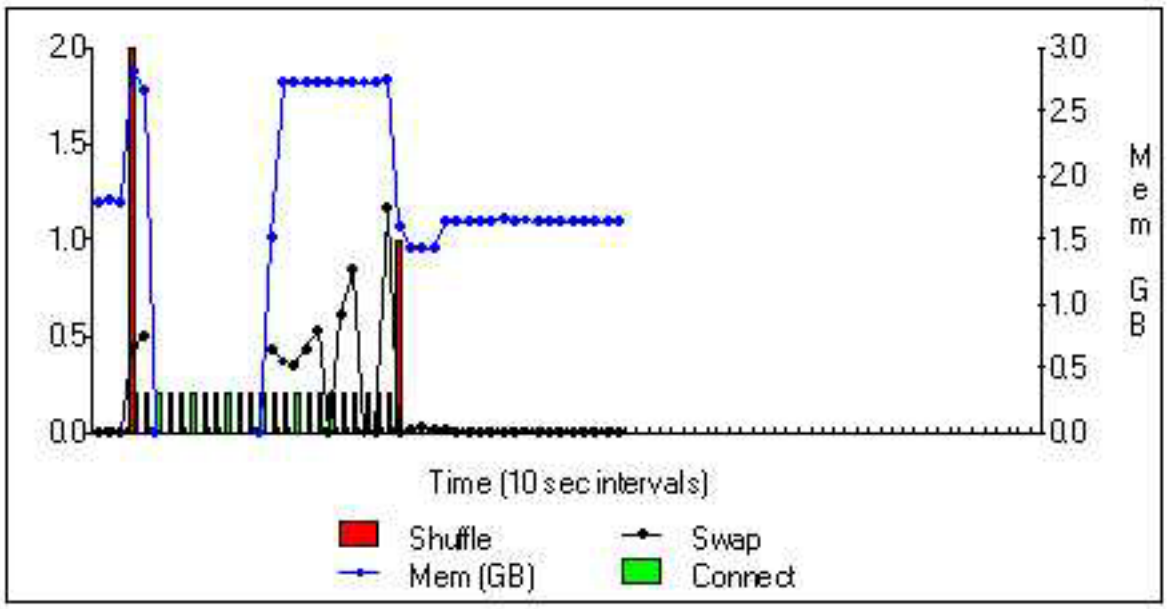

Figure 4 shows details of the *sar* reporting for a 150 MWord Multi8 job on a lightly loaded J90

In Figure 3 and Figure 4 we see the results of plotting the various *sar* options for CPU usage, memory usage and 'shuffles' as a function of time, firstly for a lightly loaded 8 Gbyte machine and secondly for a busy 4 GByte machine. For some reason *sar* fails to gather data during the bulk of the running of the real time process but still gives good data at either end. On the lightly loaded machine the timings are very tight: a shuffle is initiated, the memory allocated increases, the job starts, (*sar* is silent), the job finishes, memory deallocates and the workload returns to its previous level.

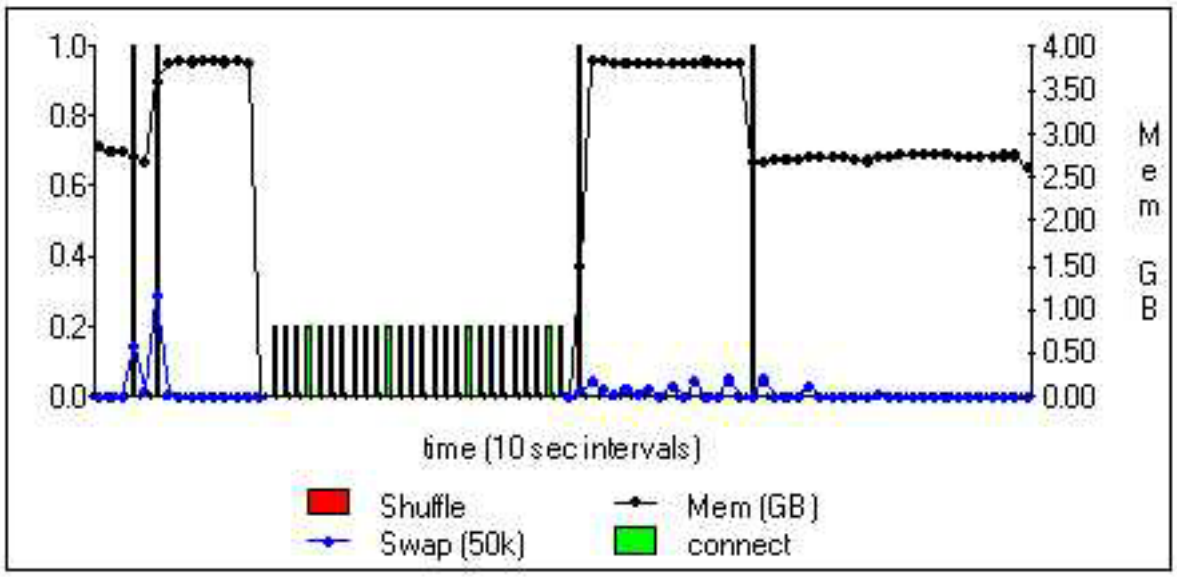

Figure 5 shows details of the *sar* reporting for a 150 MWord Multi8 job on a heavily loaded J90. The duration of 'connect' time is shown but its absolute position is arbitrary.

On the busy machine there are many delays: a first 'shuffle' request begins 20-30 seconds of swapping before the allocated memory increases; a 'shuffle' of unknown origin results in *sar* measurements reappearing; the elapsed time before the allocated memory goes down is much greater than the connect time, and there is a period of low level swapping before the job eventually terminates.

To sumarise the delays that we have observed we have encapsulated the 'overhead' into the difference between the

elapsed time from first to last 'shuffles' and the connect time given by *ja* (which is a measure of genuine cpu time given to the job).

Our results show that there is little overhead on an empty machine, except for the 150 MWord job, on a busy machine the overhead ranges from almost zero for a 4 MW job up to several minutes for the 64 MWord and 150 MWord jobs. The results are shown in Table 1 and show a degree of scatter corresponding to varying workloads. Given the granularity of the *sar* data, overheads of less than 10 seconds are not significant.

Table 1: Timings for several Multi8 Jobs

NQS ID	Memory	Connect	Elapsed	Overhead	Note
99320	4	320	328	8	busy
99325	16	272	300	28	busy
99340	64	247	409	162	busy
99379	150	232	615	383	busy
1446	150	243	594	351	busy
1447	64	260	434	174	busy
1448	16	242	396	154	busy
1449	4	282	283	1	busy
5287	4	296	302	6	quiet 8G
5288	16	255	260	5	quiet 8G
5289	64	258	264	6	quiet 8G
5290	150	242	523*	281	quiet 8G
8648	150	236	480	244	quiet 8G

*** This process showed unusual behaviour on termination and an elapsed time of 403 sec could be an alternative interpretation of the data.**

Five table entries refer to the 8 GByte machine where memory is generally under-allocated and for run 8648 we also had a primary swap partition in memory for very small processes.

There is sufficient scatter to enable almost any conclusion to be drawn but we speculate that the runs 1446 - 1449 form a reasonably self consistent set and we show in Figure 6 that the data is reasonably well fitted by a notional 'bandwidth' of around 2.2 MB/second which compares with a real bandwidth to the swap device of about 20 MByte/second. The most efficient swapping algorithm would show half of the peak swap bandwidth corresponding to each process being swapped out and in only once and we conclude that the memory scheduling of real time processes shows some scope for optimisation.

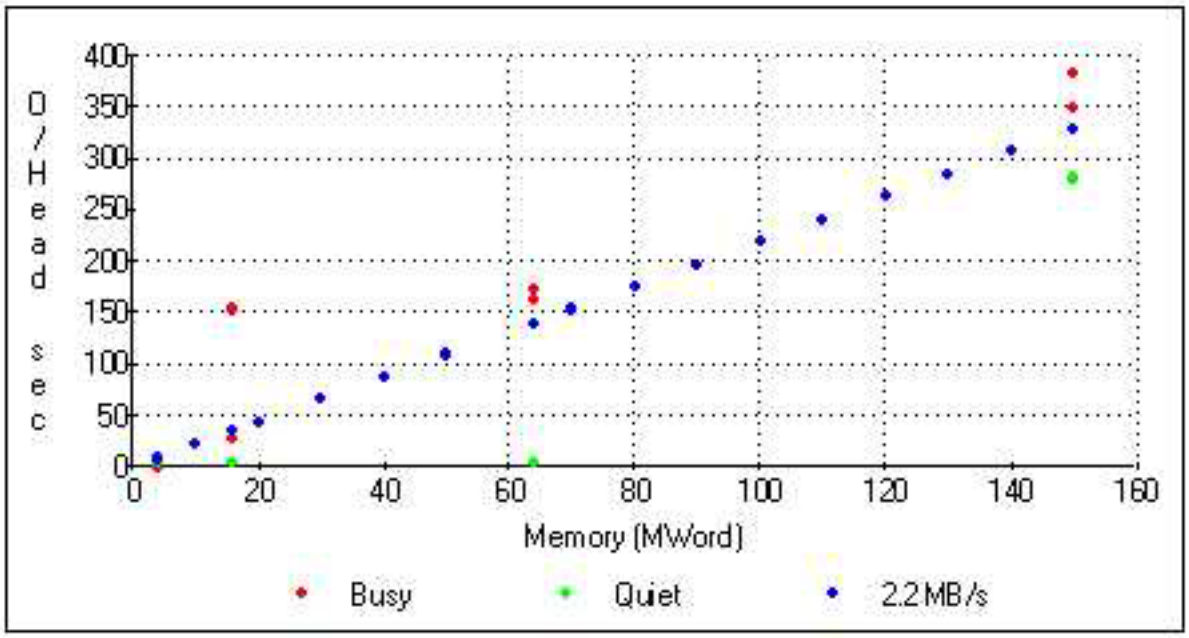

Figure 6 shows how the measured overheads approximate to a 2.2 MByte /second effective bandwidth

If we are correct in attributing these overheads to the time taken to move processes around in memory in order to load the real time process in low memory then this is a significant overhead in using the *gded* real time mechanism to manage the mix of single threaded and parallel work on a busy machine. No doubt the reasoning in requiring that real time processes load into low memory is sound but we wonder if there may be some relaxation allowed in the context of this requirement which is not strictly real time but rather to realise an identifiable priority processor set.

Conclusions

We are generally well pleased with the results of real time process scheduling and it is rapidly becoming indispensible to our machine management. We encourage SGI/Cray to integrate this requirement into future version of Unicos and cellular Irix since the power of multi-CPU supercomputers is only delivered into parallel jobs, but each job may only be a fraction of the overall machine size. On the virtual memory architecture of say the Origin 2000 the need for genuine movement of processes to low memory disappears and there is the potential for the real time mechanism to be used efficiently as long as it is designed in early enough.

References

[1] Tim Folkes, Roger Evans and Mike Armstrong, **Multi CPU pools in an NQS environment**,Proceedings Fall 1996 CUG, page 155

Author Biographies

Roger Evans manages the J932 and DEC 8400 services at Rutherford Appleton Laboratory
Tim Folkes is the senior systems analyst for the J932
Michael Armstrong is a software analyst with SGI/Cray UK.