Performance Characteristics of Gang Scheduling in Multiprogrammed Environments

Morris A. Jette
Lawrence Livermore National Laboratory
Livermore, CA 94550
jette@llnl.gov
http://www-lc.llnl.gov/global_access/dctg/jette/

Abstract:

Gang scheduling provides both space-slicing and time-slicing of computer resources for parallel programs. Each thread of execution from a parallel job is concurrently scheduled on an independent processor in order to achieve an optimal level of program performance. Time-slicing of parallel jobs provides for better overall system responsiveness and utilization than otherwise possible. Lawrence Livermore National Laboratory has deployed three generations of its gang scheduler on a variety of computing platforms. Results indicate the potential benefits of this technology to parallel processing are no less significant than time-sharing was in the 1960's.

Keywords:
gang scheduling, multiprogramming, parallel system, scheduling, space-slicing, time-slicing.

Introduction

Interest in parallel computers has been propelled by both the economics of commodity priced microprocessors and a growth rate in computational requirements exceeding processor speed increases. The symmetric multiprocessor (SMP) and massively parallel processor (MPP) architectures have proven quite popular, yet both suffer significant shortcomings when applied to large scale problems in multiprogrammed environments. The problems must first be recast in a form supporting a high levels of parallelism. Then to achieve the inherent parallelism and performance, it is necessary to concurrently schedule CPU resources to all threads and processes associated with each program. System throughput is frequently used as a metric of success; however, ease of use, good interactivity, and "fair" distribution of resources are of substantial importance to customers in a multiprogrammed environment. Efficiently harnessing the power of a multitude processors while satisfying customer requirements is a difficult proposition for schedulers.

Most MPP computers provide concurrent scheduling through space-slicing schemes. A program is allocated a collection of processors and retains those processors until completion of the program. Scheduling is critical, yet each decision has an unknown impact upon the future: should a job be scheduled at the risk of blocking larger jobs later or should processors be left idle in anticipation of future arrivals? The lack of a time-slicing mechanism precludes good interactivity at high levels of utilization. Gang scheduling solves this dilemma by combining concurrent resource scheduling, space-slicing, and time-slicing. The impact of each scheduling decision is limited to a time-slice rather than the job's entire lifetime. Empirical evidence from gang scheduling on a Cray T3D installed at Lawrence Livermore National Laboratory (LLNL) demonstrates this additional flexibility can improve overall system utilization and responsiveness.

Most SMP computers provide both space-sharing and time-slicing, but schedule each process independently. While good parallelism may be achieved this way on a lightly loaded system, that is a luxury rarely available. The purpose of gang scheduling in this environment is to improve the throughput of parallel jobs by concurrent scheduling, without degrading

either overall system throughput or responsiveness. Moreover, this scheme can be extended to scheduling of parallel jobs across a cluster of computers in order to address larger problems. Gang scheduling of DEC Alpha computers at LLNL is explored in both stand-alone and cluster environments, and shown to fulfill these expectations.

Overview of Gang Scheduling

The term "gang scheduling" refers to all of a program's threads of execution being grouped into a gang and concurrently scheduled on distinct processors. Furthermore, time-slicing is supported through the concurrent preemption and later rescheduling of the gang [4]. These threads of execution are not necessarily POSIX threads, but components of a program which can execute simultaneously. The threads may span multiple computers and/or UNIX processes. Communications between threads may be performed through shared memory, message passing, and/or other means.

Concurrent scheduling of a job's threads has been shown to improve the efficiency of both the individual parallel jobs and the system [3, 13]. The job's perspective is similar to that of a dedicated machine during the time-slices of its execution. Some reduction in I/O bandwidth may be experienced due to interference from other jobs, but CPU and memory resources should be dedicated. Job efficiency improvements results from a reduction in communications latency, promoting fine-grained parallelism. System efficiency can be improved by reductions in context switching, virtual memory paging, and cache refreshing.

The advantages of gang scheduling are similar to those of time-sharing in uniprocessor systems. The ability to preempt jobs permits the scheduler to more efficiently utilize the system in several ways:

- Long running jobs and those with high resource requirements can be executed without monopolizing resources
- Interactive and other high priority jobs can be provided with near real-time response, even jobs with high resource requirements during periods of high system utilization
- Jobs with high processor requirements can be initiated in a timely fashion, without waiting for processors to be made available in a piecemeal fashion as other jobs terminate
- Low priority jobs can be executed, provided with otherwise unused resources, and preempted when higher priority jobs become available
- High system utilization can be sustained under a wide range of workloads

Job preemption does incur some additional overhead. The CPU resources sacrificed for context switching is slight and explored later in the paper. The increase in storage requirements is possibly substantial. All running or preempted jobs must have their storage requirements satisfied simultaneously. Preempted jobs must also vacate memory for other jobs, with the memory contents written to disk. The acquisition of additional disks in order to increase utilization of CPU and memory resources is likely to be cost-effective, but this need must be considered.

Other Approaches to Parallel Scheduling

Most MPP computers use the variable partitioning paradigm. In variable partitioning, the job specifies its processor count requirement at submission time. Processors allocated to a job are retained until its termination. The inability to preempt a job can prevent the timely allocation of resources to interactive or other high priority jobs. A shortage of jobs with small processor counts can result in the incomplete allocation of processors. Conversely, the execution a job with high processor requirements can be delayed by fragmentation, which necessitates the accumulation of processors in a piecemeal fashion as multiple jobs with smaller processor counts terminate. Variable partitioning can result in poor resource utilization due to resource fragmentation [10, 18], processors left idle in anticipation of high priority job arrival [12], or processors left idle in order to accommodate jobs with substantial resource requirements.

Another option is dynamic partitioning, in which the operating system determines the number of processors to be allocated to each job. While dynamic partitioning does possess the allure of high utilization and interactivity [11, 15, 16], it can make program development significantly more difficult. The program is required to operate efficiently without knowledge of processor count until execution time. The variable processor count also causes execution time variability, which may be unacceptable for workloads containing very long running jobs or even moderate size jobs with high priority.

MPP Scheduling

The variable partition paradigm prevalent on MPP architectures and its lack of job preemption makes responsiveness particularly difficult to provide. In order to ensure responsive service, some MPP computers divide resources into "pools" reserved for interactive jobs, batch jobs, or available to any job type. These pools can also be used to reserve portions of the computer for specific customers on some systems. Since jobs can not normally span multiple pools, partitioning the computer in this fashion reduces the maximum problem size which can be addressed while fragmentation reduces scheduling flexibility and system utilization. The optimal configuration places all resources into a single pool available to any job type, assuming that support for resource allocation and interactivity can be provided by other means.

SMP Scheduling

Space-sharing and time-sharing are the norm on SMP computers, providing both good interactivity and utilization. Most SMP computers schedule each process independent, which works well for a workload consisting of many independent processes. However, the solution of large problems is dependent upon the use of parallel jobs, which suffer significant inefficiencies without the benefit of concurrent scheduling [3, 13].

Parallel job development efforts at the National Energy Research Supercomputer Center (NERSC) illustrates difficulties in parallel job scheduling [2]. In order to encourage parallel job development, NERSC provided dedicated time to parallel jobs on a Cray C90 computer. Several of the parallel jobs running in dedicated mode were able to achieve a parallel performance (CPU time/wall time) over 15.5 on this 16 CPU machine and approach 10 GFlops per second of the 16 GFlops per second peak speed. While all batch jobs were suspended during the dedicated period, interactive jobs were initially permitted to execute concurrently with the parallel job. Several instances were observed of a single compute-bound interactive program reducing the parallel job's throughput and system utilization by almost 50 percent. Suspension of interactive jobs was found to be necessary in order to achieve reasonable parallel job performance.

The benefits of SMP computers can be scaled for larger problems by clustering. Efficient execution of parallel jobs in this environment requires coordination of scheduling across the entire cluster. If communications between the threads consisted exclusively of message passing, it is possible to base scheduling upon this message traffic and achieve a high level of parallelism [13, 14]. The LLNL workload makes extensive use of shared memory for communications, preventing us from pursuing this strategy.

LLNL Workload Characterization

LLNL customers have long relied upon interactive supercomputing for program development and rapid throughput of jobs with short to moderate execution times. While some of this work can be performed on smaller computers, most customers use the target platform for reasons of problem size, hardware compatibility, and software compatibility. LLNL began its transition to parallel computing with the acquisition of a BBN TC2000 in 1989. Additional parallel computers at LLNL have included the Cray C90, Cray T3D, Meiko CS-2, and IBM SP2. Many of our problems are of substantial size and have been well parallelized.

The LLNL Cray T3D workload is typical of that on our other parallel computers. The model at LLNL has 256 processors, each with 64 megabytes of DRAM. All processors are configured into a single pool available to any job. The LLNL Cray T3D is configured to permit interactive execution of jobs up to 64 processors and 2 hours execution time. Interactive jobs account for 67 percent of all jobs executed and consume 13 percent of all CPU resources delivered. The interactive workload typically ranges from 64 to 256 processors (25 to 100 percent of the computer's processors) during working hours and drops to a negligible level in the late night hours. Peak interactive workloads reach 320 processors for a 25 percent oversubscription rate. Timely execution of interactive jobs is dependent upon the preemption of batch jobs and, in extreme cases, time-sharing processors among interactive jobs.

Large jobs account for a high percentage of resources utilized on the Cray T3D as shown in Table 1. Memory requirements are quite variable, but most jobs use between 42 and 56 megabytes per processor. Due to Cray T3D hardware constraints, a contiguous block of processors with a specific shape must be made available to execute a job [1], making fragmentation a particularly onerous problem. Gang scheduling permits us to preempt jobs as needed to execute large jobs in a timely fashion with minimal overhead.

Job Size (CPU Count):	2	4	8	16	32	64	128	256
CPU Utilization (%):	0.0	0.0	0.3	0.7	16.9	37.2	38.3	6.6

Table 1: CPU Utilization by Job Size on Cray T3D at LLNL

Most programs use either PVM (Parallel Virtual Machine) or MPI (Message Passing Interface) libraries for communications between the threads. Although the Cray T3D has a distributed memory architecture, it will support a shared memory programming model permitting a job to read and write memory local of another processor assigned to that job. This shared memory programming model has considerably lower overhead than message passing and is widely used. A small number of programs utilize a combination of both paradigms: shared memory within the Cray T3D and message passing to communicate with threads or serial program components executing on a Cray YMP front-end. This multiple paradigm model is expected to be common on our DEC Alpha cluster and IBM SP2 containing SMP nodes.

LLNL Gang Scheduler Design Strategy

The LLNL customers expect rapid response, even on a heavily utilized multiprogrammed computer; however the need for rapid response is not uniform across the entire workload. In order to provide interactivity where required and minimize the overhead of context switching, we divide our workload into six different classes. Each job class has significantly different scheduling characteristics as described below:

- Express jobs have been deemed by management to be mission critical and are given rapid response and optimal throughput.
- Interactive jobs require rapid response time and very good throughput during extended working hours. The jobs' response time and throughput can be reduced at other hours for the sake of improved system utilization and throughput of production jobs.
- Debug jobs require rapid response time during extended working hours. The jobs' response time can be reduced at other hours for the sake of improved system utilization. Debug jobs can not be preempted on the Cray T3D.
- Production jobs do not require rapid response time, but should receive very good throughput at night and on weekends.
- Benchmark jobs do not require rapid response time, but can not be preempted.
- Standby jobs have low priority and are suitable for absorbing otherwise idle compute resources.

The default classes are production for batch jobs, debug for totalview debugger initiated jobs, and interactive for other jobs directly initiated from a user terminal. Jobs can be placed in the express class only by the system administrator. Benchmark and standby classes can be specified by the user at job initiation time.

Each job class has a number of scheduling parameters, including relative priority and processor limit. There are also several system-wide scheduling parameters such as aggregate processor limit for all gang scheduled jobs. The scheduling parameters can be altered in real-time, which permits periodic reconfiguration. LLNL emphasizes interactity during extended work hours (7:30 AM to 10:00 PM) and throughput at other times. The gang scheduler daemon itself can also be updated at any time. Upon receipt of the appropriate signal and completion of any critical tasks, the daemon writes its state to a file and initiates a new daemon program.

Sockets are utilized for user communications, most of which occur at job initiation. Sockets are also used for job state change requests and scheduling parameter changes, which are rare. Job and processor state information is written periodically to a data file, which is globally readable. This file is read by the "gangster" application, which is the users' window into gang scheduling status.

We have delegated the issue of "fair" resource distribution to other systems, including the Centralized User Bank (CUB) [8] and Distributed Production Control System (DPCS) [17]. These systems provide resource allocation and accounting capabilities to manage the entire computing environment with a single tool. Both systems exercise coarse grained control by the scheduling of batch jobs and fine grained control by adjusting nice values of both interactive and batch processes. Our gang scheduler is designed to recognize changes in nice value and schedule accordingly. Jobs of classes which can be

preempted will automatically be changed to standby class when at high nice value and returned to their requested class upon reduction in nice value. This mechanism has proven to function very well in managing resource distribution while minimizing interdependence of the systems.

Significant differences exist in the three LLNL gang scheduler implementations. These differences were largely the result of architectural differences, but some were based upon experiences with previous implementations. Each implementation is described below with results.

BBN TC2000

In order to pioneer parallel computing at LLNL, a BBN TC2000 with 126 processors was acquired in 1989. The TC2000 has a shared memory architecture and originally supported space-sharing only.

The gang scheduler for the TC2000 reserves all resources at system startup and controls all resource scheduling from that time [6, 7]. User programs require no code changes to communicate with the gang scheduler. However, the program must load with a modified version of the mandatory parallel program initiation library. Rather than securing resources directly from the operating system, this library secures resources from the gang scheduler daemon. The program may also increase or decrease its processor count during execution by explicitly notifying the gang scheduler daemon. Otherwise, a job must continue execution on the specific processors originally assigned to it. Time-sharing is performed by the use of sleep and wake signals, issued concurrently to all threads of a job. This mechanism can provide for very rapid context switches, on the order of milliseconds. This implementation did have "fair share" mechanism with the objective of providing each user with access to comparable resources. The gang scheduler would determine resource allocation to be made for each ten second time-slice with the objective of insuring interactivity, equitable distribution of resources, and high utilization.

While the gang scheduler implemented for the TC2000 was able to provide good responsiveness and throughput, some shortcomings should be noted. Despite the shared memory architecture to the TC2000, it was not possible to relocate threads in order to better balance the continuously changing workload. The scheduler could react to workload changes only at time-slice boundaries, which limited responsiveness. The user based fair share mechanism was abandoned in later implementations due to the availability of an independent and more comprehensive resource allocation system.

Cray T3D

The Cray T3D is a massively parallel computer incorporating DEC alpha 21064 microprocessors capable of 150 MFLOPS peak performance. Each processor has its own local memory. The system is configured into nodes, consisting of two processors with their local memory and a network interconnect. The nodes are connected by a bidirectional three-dimensional torus communications network. There are also four synchronization circuits (barrier wires) connected to all processors with a tree shaped interconnect [1].

Without getting into great detail, the T3D severely constrains processor and barrier wire assignments. Jobs must be allocated a processor count which is a power of two, with a minimum of two processors (one node). A job’s can be built to run with any valid processor count, but its processor count can not change after execution begins. The processors allocated to a job must have a specific shape with specific dimensions for a given problem size. For example, an allocation of 32 processors must be made with a contiguous block of eight processors in the X direction, two processors in the Y direction, and two processors in the Z direction. Furthermore, the possible locations of the processor assignments are restricted. These very specific shapes and locations for processor assignment are the result of the barrier wire structure. Jobs must be allocated one of the four barrier wires when initiated. The barrier wire assignment to a job can not change if the job is relocated and, under some circumstances, two jobs sharing a single barrier wire may not be located adjacent to each other.

Prior to the installation of a gang scheduler on our Cray T3D, we were forced to make several significant sacrifices in order to satisfy our customers’ need for interactivity [5, 9]. The execution time of batch jobs was restricted to insure "reasonable" responsiveness to jobs with large processor requirements and interactive jobs, although one may argue that delays on the order of hours may not be reasonable. Most batch jobs were limited to four hours. One batch queue permitted execution times up to 19 hours, but this queue was enabled during only brief periods on weekends and holidays. Since jobs could not be preempted, our batch workload was dramatically reduced at 4:00 AM. As batch jobs completed, their released resources might remain unused by any interactive job for many hours. At times of heavy interactive use, the initiation of an interactive job had to wait for other jobs to terminate and release resources. The processor allocation restrictions also made for severe

fragmentation problems. While interactivity was generally good, the processor utilization rate of 33 percent was considered unacceptable.

The Cray T3D gang scheduler implementation has a several significant differences from that of the BBN TC2000. Since the initiation of all parallel jobs on the T3D is conducted by a single application program, we placed a wrapper around this to communicate with the gang scheduler daemon. No changes to the user application or scripts were required. The fixed time-slice period was replaced with an event driven scheduler with the ability to react instantly to changes in workload. The allocation of specific processors and barrier wires can dramatically effect performance on the T3D, so substantial effort was placed in developing software to optimize these selections. Processor selection criterion includes best-fit packing, placement of jobs with similar scheduling characteristics in close proximity, and minimizing contention of the barrier wire circuits.

Context switching logic also required substantial modification. The BBN TC2000 supported memory paging, while the Cray T3D lacks support for paging. Cray T3D job preemption results in the entire memory image of the preempted job being written to disk before the processors can be reassigned, which requires about one second per preempted processor. In the case of a 64 processor jobs being context switched, about one minute is required to store the preempted job's context and another minute to load the state of another job of similar size. The T3D gang scheduler calculates a value for each job including its processor count, job type, location (disk or memory) to make job preemption and initiation decisions. Additional information associated with each job class and used in this calculation include: maximum wait time, processor limit, and do-not-disturb time multiplier. The minimum time-slice for a job is the product of the job's processor count and its class' do-not-disturb time multiplier. While the minimum time-slice mechanism does reduce responsiveness, it prevents the costly thrashing of jobs between memory and disk and is critical for maintaining a high utilization level.

The Cray T3D gang scheduler has been in operation since March 1996. We were able to dramatically modify the batch environment to fully subscribe the machine during the day and oversubscribe it at night by as much as 100 percent. The normal mode of operation in the daytime is for interactive class jobs to preempt production class jobs shortly after initiation. The interactive jobs then usually continue execution until completion without preemption. At night, the processors are oversubscribed temporarily and only for the express purpose of executing larger jobs (128 or 256 processor). This strategy results in only about eight percent of all jobs ever being preempted.

The batch queue for long running jobs has been substantially reconfigured: its time limit has been increased from 19 to 40 hours, its maximum processor allocation (for all running jobs in the queue) increased from 64 to 128 processors, and is enabled at all times. System utilization increased substantially and weekly CPU utilization rates over 96 percent have been sustained. Figure 1 shows monthly CPU utilization for a period of 15 months. Three different schedulers were utilized over this period. UNICOS MAX is the native Cray T3D operating system from Cray Research. DJM or Distributed Job Manager is a parallel job scheduler originally developed by the Minnesota Supercomputer Center and substantially modified by Cray Research. The LLNL developed gang scheduler is also shown. The CPU utilization reported is that during which a CPU is actually assigned to a program which is memory resident. CPU utilization is reduced by three things:

1. Context switch time, a CPU is unavailable while a program's image is being transferred between disk and memory
2. Sets of processors on which no job can fit, a packing problem
3. Insufficient work, particularly on weekends

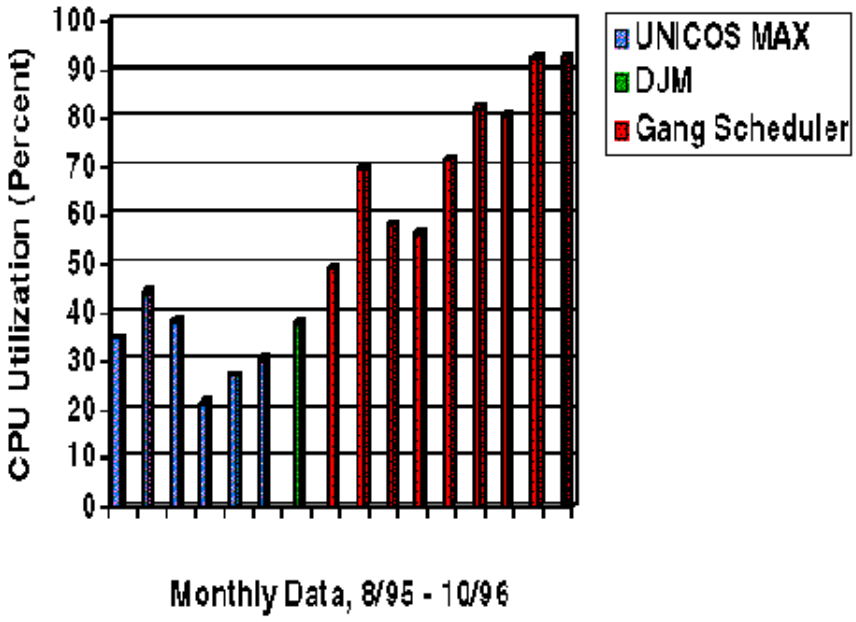

Figure 1: Cray T3D CPU Utilization

Benchmarks run against both UNICOS MAX and an early prototype of the LLNL gang scheduler showed a 21 percent improvement in interactive job throughput with no reduction in aggregate throughput. The cost of moving jobs' state between memory and disk to provide responsiveness was fully compensated for by more efficient packing of the processor torus. The current implementation has additional performance enhancements and should show an aggregate throughput improvement of a few percent.

Figure 2 shows a gangster display of typical Cray T3D daytime utilization. Note that only 12 of the 256 processors (six of 128 nodes) are not assigned to some job. A total of ten interactive and debug class jobs are running and using 148 of the processors. Over half of the batch workload is currently paged out for these interactive jobs. Left side of display shows the contents of each node (two processors). Each letter indicates the job and a period indicates an unused node. The right side of the display describes each job. The W field shows the barrier wire used. The MM:SS field shows the total execution time accumulated. The ST field shows the job's state: i=swapping in, N=new job, not yet assigned nodes or barrier wire, o=swapping out, O=swapped out, R=running, W=awaiting commencement of execution, assigned nodes and barrier wire. The gang scheduler state information is written to disk at 15 second intervals. The gangster display is updated at the same rate.

```
          gangster - 13:25 - 42377
   b  b  n  n  s  s  s  s  CLAS JOB-USER      PID  COMMAND  #PE BASE W ST MM:SS
  b  b  n  n  s  s  s  s   Int  d - skjellum 73843 sendmany   8 604  0 R 119:26
 b  b  n  n  s  s  s  s    Int  g - skjellum 76306 testall    2 716  0 R  92:02
b  b  n  n  s  s  s  s     Int  t - nordlund 77238 parcas    32 000  2 R  84:29
                          Int  e - nordlund 79224 parcas    32 200  0 R  40:21
   b  b  n  n  s  s  s  s  Int  b - germann  80421 therm-dv  32 020  3 R  18:15
  b  b  n  n  s  s  s  s   Int  w - eltgroth 80998 fivept     2 702  2 R   8:42
 b  b  n  n  s  s  s  s    Int  h - eltgroth 81227 pet_test   4 700  0 R   5:24
b  b  n  n  s  s  s  s     Int  f - pcovello 81324 nimrod     2 712  1 R   3:43
                          Int  o - pcovello 81494 nimrod     2 706  3 R   0:00
   t  t  e  e  a  a  .  h
  t  t  e  e  a  a  .  f   Dbug a - susan    81280 aaaa8123  32 400  1 R   4:28
 t  t  e  e  a  a  d  .
t  t  e  e  a  a  d  g     Prod s - mahdi    83818 ge        64 420  0 R1462:43
                          Prod j - tomas    78508 mdter200  32 400  1 O  49:42
   t  t  e  e  a  a  .  h  Prod u - eduardo  79879 new       64  -1 -1 N   0:00
  t  t  e  e  a  a  .  w   Prod m - delarub  79890 new       64  -1 -1 N   0:00
 t  t  e  e  a  a  d  .    Prod n - tomas    80331 mdterrep  32 220  0 R  19:39
t  t  e  e  a  a  d  o
```

Figure 2: Typical daytime gangster display on Cray T3D

While the utilization rate is quite satisfactory, responsiveness is also of great importance. Responsiveness can be quantified by slowdown, the ratio of total time spent in the system to the run time. During the three week period of July 23 through August 12, 2659 interactive jobs were executed using a total of 44.5 million CPU-seconds and 1328 batch jobs were executed using a total of 289.5 million CPU-seconds. The slowdown of the aggregate interactive workload was 18%, which is viewed quite favorably. Further investigation shows a great deal of variation in slowdown. Most longer running interactive jobs enjoy slowdowns of only a few percent. Interactive jobs executed during the daytime typically begin execution within 30 seconds and are not preempted. Interactive jobs executed during late night and early morning hours experienced slowdowns as high as 1371 (a one second job delayed for about 23 minutes). However the computer is configured for high utilization and batch job execution during these hours, so high slowdowns are not unexpected.

DEC Alpha

The Digital Unix 4.0D operating system includes a "class scheduler", which is essentially a very fine grained fair share scheduler. The class scheduler permits processes, process groups, or sessions to be grouped in a class. The class can then be allocated some share of CPU resources. For example, the eight threads of a job could be placed into a single class and allocated 80 percent of the CPU resources on a ten CPU computer. While the class scheduler does not explicitly reserve eight CPUs for the eight threads of this parallel job, the net effect is very close. The operating system also recognizes advantage in keeping a process on a particular CPU to avoid refreshing its cache. We have found the class scheduler actually delivers over 99 percent of the CPU resources as desired for gang scheduling with minimal thread migration between processors. For the job which fails to sustain its target number of runable threads, the class scheduler will allocate the CPU resources to other jobs in order to sustain high overall system utilization.

Since parallel jobs are not normally registered with the Digital UNIX operating system, the gang scheduler relies upon explicit registration through the use of a library. While it is highly desirable that user code modification for gang scheduler be avoided, that is impossible to achieve at this time. Imbedding a few gang scheduler remote procedure calls directly in MPI and PVM libraries would free many users from having to modify their programs. Presently, the gang scheduled application must be modified with a few simple library calls, including:

- Register job with gang scheduler: A global gang scheduler job ID is returned
- Register resource requirements: CPU, memory, and disk space requirements are specified for the job on each computer to be gang scheduled
- Register the processes: Associate a specific process, process group, or session with the gang scheduler job ID

Since it is necessary to coordinate activities across multiple computers, the DEC Alpha gang scheduler returned to the fixed time-slice model used on the BBN TC2000. In order to manage these time-slices across multiple computers, the concept of "tickets" was introduced. These tickets represent specific resource allocations at specific times and are issued only for jobs spanning multiple computers. Jobs which execute only on a single computer are not pre-issued tickets, but are managed by that computer’s gang scheduler daemon which makes scheduling decisions at the start of each time-slice. This design permits each computer to operate as independently as possible, while permitting the gang scheduling of jobs across the cluster as needed. The tickets are managed by the gang scheduler daemon on each computer in the cluster and are associated with a job for its lifetime. A job may be given additional tickets or have some revoked depending upon changes in overall system load. The job is also permitted to alter its resource requirements during execution. A change in the number of CPU required for a job may result in revoked or additional tickets.

The gangster display program was re-written in Tcl/Tk in order to provide a richer environment. Detailed information on system and job status now includes a history of CPU, real memory, and virtual memory use which is updated at time-slice boundaries. This information can be helpful for system administrators and programmers tuning their systems. For example, if a program’s CPU allocation and CPU use are substantially different, the desired level of parallelism is not being achieved and the matter should be investigated. A dramatic changes in real memory use for a program during its execution may indicate substantial paging, which also warrants investigation. Figures 3 and 4 show displays of system and program status.

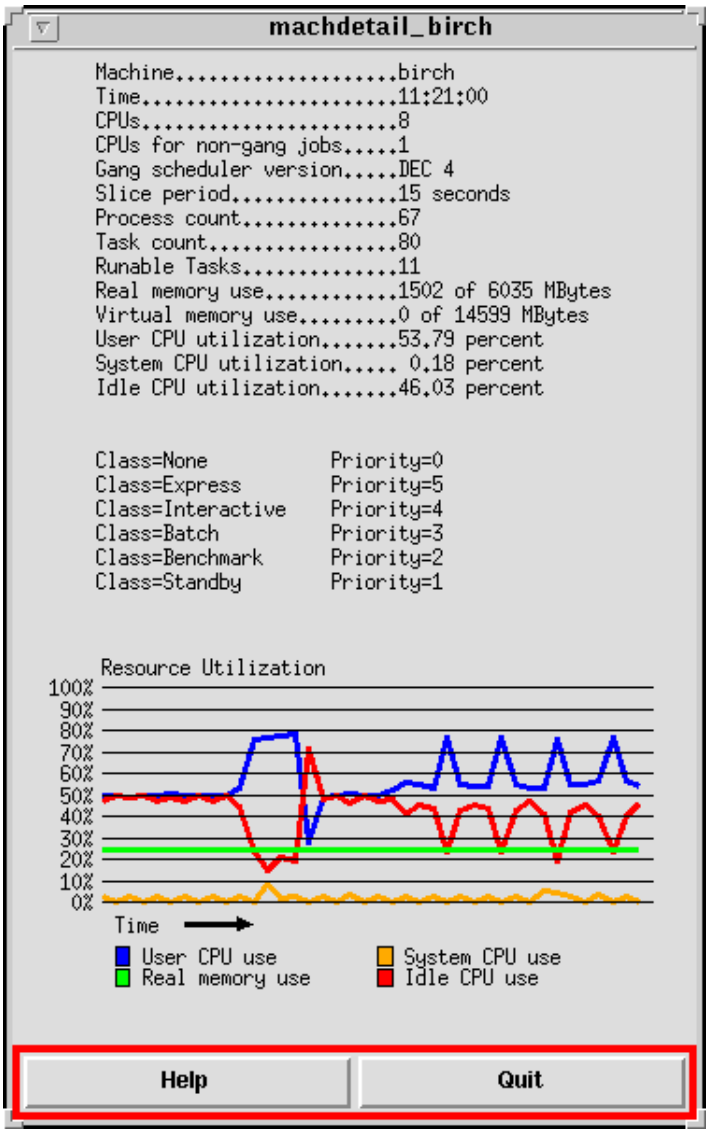

Figure 3: Gangster display of DEC machine status

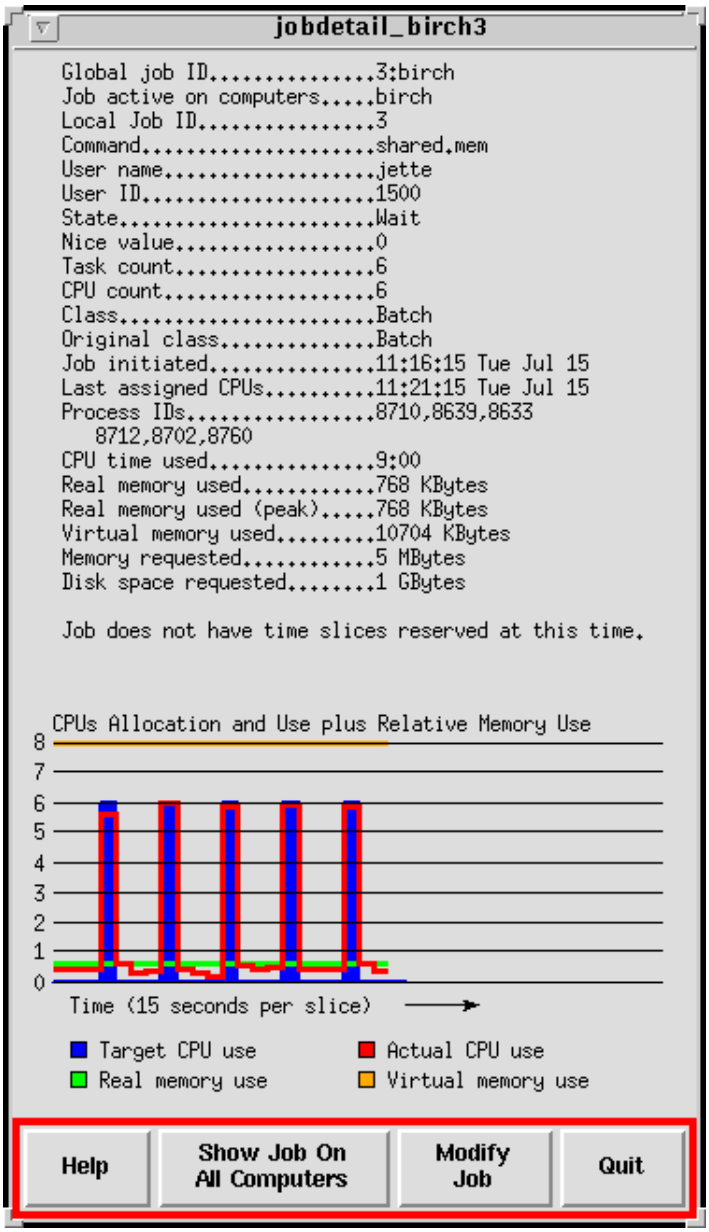

Figure 4: Gangster display of DEC job status

In order to insure that good responsiveness is preserved for all jobs, this gang scheduler permits processors to be reserved for non-gang scheduled jobs. This prevents gang scheduled jobs from reserving all processors on a computer for the entire time-slice, which could prevent logins or other interactions for a significant period. The gang scheduler daemon also reacts to changes in the overall workload, insuring resources are fairly distributed to all active jobs.

Gangster also permits several system-wide operations to a gang scheduled job including: suspend, resume, kill, and change job class. For example, the kill job operation will send kill signals to all processes registered to a gang scheduled job on all computers running that job.

Conclusions

Gang scheduling has been shown to provide an attractive multiprogrammed environment for multiprocessor systems in

several ways:

- Parallel jobs can be provided with access to all required resources simultaneously, providing the illusion of a dedicated environment
- Interactive and other high priority jobs can be provided with rapid response and excellent throughput
- Jobs with large resource requirements can be initiated rapidly, without having to wait for multiple jobs to terminate and release resources
- A high level of utilization can be maintained under a wide range of workloads

Experience on the Cray T3D has been overwhelmingly positive. It can now sustain processor utilization rates over 95 percent through the course of a week while providing the aggregate interactive workload with a slowdown of less than 20 percent. Such performance give this distributed memory MPP a range of performance spanning large-scale parallel applications to general purpose interactive computing.

Experience on the DEC Alpha environment has also been very favorable during our testing period. A very rich and interactive environment is available on these fast SMPs, while true supercomputing class problems can be addressed by harnessing the power of a cluster for parallel jobs. Our plans call for bringing two 80 CPU DEC Alpha 8400 clusters under the control of gang scheduling in the Fall of 1997.

References

1. Cray Research Inc. **Cray T3D System Architecture Overview**. Order number HR-04033, Sept 1993.

2. B. C. Curtis and B. E. Kelly, **Dedicated Computing on a YMP/C916**. Cray User Group, Sep 95.

3. A. C. Dusseau, R. H. Arpaci and D. E. Culler, **Effective Distributed Scheduling of Parallel Workloads**. ACM SIGMETRICS ‘96 Conference on the Measurement and Modeling of Computer Systems, 1996.

4. D. G. Feitelson, **A Survey of Scheduling in Multiprogrammed Parallel Systems**. IBM Research Report RC19790 (87657), Feb 1995.

5. D. G. Feitelson and M. A. Jette, **Improved Utilization and Responsiveness with Gang Scheduling**. IPPS ’97 Workshop on Job Scheduling Strategies for Parallel Processing, Apr 1997.

6. B. Gorda and R. Wolski, **Timesharing massively parallel machines**. International Conference on Parallel Processing, volume II, Aug 1995, pp. 214-217.

7. B. C. Gorda and E. D. Brooks III, **Gang Scheduling a Parallel Machine**. Technical Report UCRL-JC-107020, Lawrence Livermore National Laboratory, Dec 1991.

8. M. Jette and J. Reynolds, **Centralized User Banking and User Administration on UNICOS**. Cray User Group, Mar 1994.

9. M. Jette, D. Storch and E. Yim, **Gang Scheduler - Timesharing the Cray T3D"**. Cray User Group, Mar 1996, pp. 247-252.

10. K. Li and K-H Cheng, **A two-dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system**. Journal of Parallel and Distributed Computers 12(1), May 1991.

11. C. McCann, R. Vaswani, and J. Zahorjan, **A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors**. ACM Transactions on Computer Systems 11(2), May 1993.

12. E. Rosti, E. Smirni, G. Serazzi and L. W. Dowdy, **Analysis of non-work-conserving processor partitioning policies**. Job Scheduling Strategies for Parallel Processing, 1995.

13. P. G. Sobalvarro and W. E. Weihl, **Demand-based Coscheduling of Parallel Jobs on Multiprogrammed**

Multiprocessors. IPPS ’95 Parallel Job Scheduling Workshop, Apr 1995.

14. P. G. Sobalvarro, S. Pakin, W. E. Weihl, and A. A. Chien, **Dynamic Coscheduling on Workstation Clusters**. Symposium on Operating Systems Design and Implementation, Oct 1996.

15. C. Severance and R. Enbody, **Comparing Gang Scheduling with Dynamic Space Sharing on Symmetric Multiprocessors Using Automatic Self-Allocating Threads (ASAT)**, IPPS, Apr 1997.

16. A. Tucker and A. Gupta, **Process control and scheduling issues for multiprogrammed shared-memory multiprocessors**. 12th Symposium on Operating Systems Principles, Dec 1989.

17. R. Wood, **Distributed Production Control System**. Technical Report UCRL-TB-122372, Lawrence Livermore National Laboratory, Nov 1995.

18. Q. Yang and H. Wang, **A new graph approach to minimizing processor fragmentation in hypercube multiprocessors**. IEEE Transactions on Parallel and Distributed Systems 4(10), Oct 1993.

Work performed under the auspices of the U.S. DOE by LLNL under contract W-7405-ENG-48.

Document Number UCRL-MI-127313.

Disclaimer