Optimization of SV1 Application Codes in a Production Environment

David Turner and Tina Butler
NERSC, Lawrence Berkeley National Laboratory, Berkeley, CA 94720

Mike Stewart and Bob Thurman
Cray Inc., Lawrence Berkeley National Laboratory, Berkeley, CA 94720

Abstract

NERSC's users have migrated PVP code from C90 to J90 to SV1 processors over the last couple of years. Most user codes have not yet realized the full performance potential of the cache-based SV1 processors. Furthermore, NERSC has not configured any MSP processors, out of concern for the effect it would have on overall system throughput. The purpose of this work is to investigate whether sustained high performance can coexist with high system throughput.

We first detail the selection of user codes for optimization. These codes are then used to investigate techniques for single-processor optimization multiprocessor optimization, and finally MSP optimization. Included will be a consideration of the implementation of MSP processors in the NERSC production environment and an investigation of the effectiveness of scheduling work in this mixed MSP/SSP environment.

Introduction

The National Energy Research Scientific Computing Center (NERSC), located at Lawrence Berkeley Laboratory (LBL) in Berkeley, California, has been a major user of Cray high-performance computers for more than 20 years. Currently, NERSC operates a cluster of three Cray SV1 parallel vector processor (PVP) machines with a total of 64 vector processors, a 692-processor Cray T3E-900, and several other platforms from various vendors.

The Department of Energy's (DOE) Office of Science evaluates proposals from researchers around the country, and allocates NERSC resources to a wide variety of research groups. These resources take the form of "computer time" (defined differently for different architectures) and archival storage space. The Principal Investigator (PI) of each research group is responsible for managing the allocation for their group.

The user population includes researchers that have used NERSC resources since the "early days" of vector computing in the late 1970s, and have extensive knowledge of optimization techniques appropriate to that architecture. Historically, once a code had been optimized for a particular Cray vector architecture, very little optimization was necessary to achieve acceptable performance on subsequent Cray vector models. However, with the introduction of the SV1, some codes have needed additional restructuring to use the vector cache memory effectively.

One proposed path to higher performance on the SV1 is the utilization of its multi-streaming processor (MSP) capability. For a multi-user, multi-discipline facility such as NERSC, this raises many issues concerning system throughput and fair scheduling.

This paper will investigate two aspects of NERSC's 24x7 production environment:

- The impact on NERSC users trying to optimize applications for the SV1; and
- Its relationship to MSP utilization.

The NERSC PVP Cluster

The NERSC PVP cluster consists of three SV1 computers with the following characteristics:

Name	Processors	Memory	Purpose
Killeen	16	1 GW	Interactive
Seymour	24	1 GW	Batch, jobs up to 512MW
Bhaskara	24	1 GW	Batch, jobs up to 512MW

Each machine has about 350GB of fast RAID disk. There is an additional 140GB of disk for home directories on Killeen. These volumes are exported (via NFS) to the two batch machine, providing batch jobs access to home directories.

Killeen provides an interactive environment for code development and debugging. Interactive limits are set high enough (80MW/10 CPU-hours) to allow for some "interactive production" computing. Killeen also runs a small number of batch jobs (of up to 256MW), which can be checkpointed if interactive response suffers.

The batch system is Cray's Network Queuing Environment (NQE). Users submit jobs on Killeen, where they are held in a "pending" state until one of the two batch machines reports that it has available resources (based mainly on free memory and CPU load average). Once a job is dispatched to a batch machine, it normally begins execution immediately in one of three queues (80MW, 256MW, or 512MW). These queues all have time limits of 120 CPU-hours and disk limits of 80GB. Larger limits are occasionally granted on a case-by-case basis.

Working with Cray, NERSC has implemented a *class of service* batch priority job scheduling and charging system. When submitting a batch job, the user specifies one of three batch priority classes: Premium, Regular, or Low. The priority class primarily affects how long a job remains in the pending state. For example, in February 2001, the average pending time for Premium jobs was about 2.5 hours, for Regular jobs, about 14 hours, and for Low jobs, around 78.5 hours. Upon completion, jobs are charged according to their priority class: Premium at 2.0, Regular at 1.0, and Low at 0.5.

It should be pointed out that Unix "`nice`" values no longer have much effect on turnaround of batch jobs. The priority classes do correspond to different `nice` values (Premium=5, Regular=6, Low=7), but these values are so close together that they don't significantly impact runtimes. On Killeen, interactive processes run at a `nice` value of 0, and thus have a measurable advantage over the small number of batch jobs that might also be present.

All three machines are typically fully loaded (0% idle) and the CPUs oversubscribed. Killeen usually has between 24 and 40 runable processes, whereas the batch machines average between 40 and 60 processes. Occasionally, some idle time is incurred if one or more large-memory, single-processes job(s) run. In this case, the system runs out of memory before it runs out of processors, and the CPUs are undersubscribed. Due to the high overhead of swapping, memory oversubscription is generally avoided.

The Benchmarks

Three codes were donated by some of NERSC's largest PVP users (measured by either awarded allocation or actual CPU time used). The following program "descriptions" were extracted from comments in the codes.

`t743lin1`

Toroidal nonlinear 3D-MHD equations. Solves a set of eight nonlinear coupled equations for the pressure, mass density, momenta, and the magnetic field. The diffusive terms are advanced explicitly. A fourth order scheme with hyperviscosity advances the convective terms. Time stepping is second order with trapezoidal-leapfrog

`xqcd_hot`

Lattice QCD. Hybrid (microcanonical/langevin) method for su(3). Noisy fermions (langevin); exponential updating for gauge fields. Optimal for any number of fermion flavours less than or equal to 4. Modifications of gottleib-liu-toussaint-renken-sugar (phys. rev. d35, 2531 (1987)) to original duane-kogut algorithm used to keep errors O(dt^2).

`classic`

Core collapse supernova simulation. A discrete ordinates neutrino transport code coupled to a one-dimensional, adaptive mesh hydrodynamics code. The radiation transport requires the solution of a large sparse linear system at each time step.

Initial Observations

Evaluating performance issues on heavily loaded PVP production systems poses a number of obstacles. Among these are non-deterministic behavior of autotasked codes, and more importantly, the choice of a performance metric. For most users, the single most important measure of performance is elapsed time (turnaround time). However, when running on machines whose CPUs are heavily oversubscribed, evaluating code performance based on elapsed time is not possible. The typical user will therefore use total CPU time as the measure. Usually, several runs are used to evaluate a single optimization, to eliminate minor variations due to the operating system's timesharing behavior. If these variations are larger than the savings due to the optimization, the user may have a very difficult time determining the effectiveness of the optimization.

In trying to establish baseline performance figures for the selected user codes, we quickly encountered some perplexing results. No matter which code we ran, the CPU charges on the interactive machine Killeen were about 20% less than those on either of the batch machines. This was independent of the level of optimization used. The following table shows an example of the data collected while running the fusion code `t743lin1` on a single CPU (compiled with `-O3`) of both Killeen and Seymour. In this table, `nts` is the number of time steps for which this time-dependent code was executed.

Machine	User	Sys	Total	nts	Ratio
Killeen	815.6	9.8	825.4	250	
Seymour	1013.4	13.6	1027.0	250	1.244
Killeen	2334.7	111.8	2446.5	750	
Seymour	2870.7	53.9	2924.6	750	1.195

During the `nts`=250 runs, HPM Group 2 (memory activity) statistics were gathered for the user process. On Killeen, 25% of the memory references encountered conflicts; on Seymour, 41% of the memory references were delayed due to conflicts.

To investigate this further, we took advantage of a scheduled maintenance period during which network connectivity was unavailable. Because most batch jobs depend on access to home directories via NFS and archival storage via HIPPI, all running jobs were checkpointed. During the resulting dedicated time, we ran the following experiment on Killeen and Seymour.

A Fortran program was written to simulate a controlled production workload to "pound" on memory. Each pounder process was eight-way autotasked and vectorized, referencing 150MW randomly.

During testing, one, two, or three copies of the pounder were run, along with the user code (`t743lin1` with `nts`=100) being timed. The table below shows the User CPU time for the user code under various system loads (number of parallel threads and total amount of memory in use under "Pounders").

Pounders	Killeen		Seymour	
	User	**Conflicts/Ref**	**User**	**Conflicts/Ref**
None	253.7	0.16	254.0	0.16
8/150	273.3	0.20	273.0	0.20
16/300	342.4	0.46	320.8	0.31
24/450	335.6	0.43	429.7	0.66

The results show that system load (or more precisely, amount of memory traffic) can have a drastic influence on the CPU time charged to individual user processes. The pounders are obviously not a realistic work load, but long-term monitoring of system-wide HPM Group 2 data (using `/etc/hpmall`) show Killeen averages about 0.4 conflicts/memory reference, while Seymour and Bhaskara average between 0.6 and 0.7 conflicts/memory reference.

Thus, depending on the current job mix, users who are measuring CPU time to judge the effectiveness of their code optimization efforts can easily have their results masked by this memory contention. Results are mostly consistent when multiple runs are made one after another; however, comparing results from runs made on different days is often fruitless. Furthermore, users must be careful not to compare timing results from the interactive machine against those from the batch machines. These will always differ, due the higher number of processors (and thus the greater demands on the memory subsystem) on the batch machines.

The Multi-Streaming Processor

The Cray SV1 includes a novel feature wherein four vector processors can be combined to form a single virtual processor. While similar (from an end-user perspective) to autotasking, multi-streaming allows more efficient synchronization and communication between processors, and hence higher performance.

When first delivered to NERSC, the SV1s required a system reboot to configure MSPs, and once configured, the MSPs could only execute programs compiled specifically for MSP execution. The result would be an unacceptable amount of idle time, given NERSC's commitment to DOE to deliver a certain number of CPU hours to our users.

However, recent releases of Unicos have relaxed these restrictions. MSPs can be configured "on the fly", and once configured, can execute non-MSP programs (if no MSP-specific programs are available). We decided to investigate the performance of some typical user codes compiled for MSP execution.

Eight SV1 CPUs were configured into two MSP processors on Bhaskara. We have observed no impact of this on our "normal" job mix. Autotasking performance was compared to MSP performance using three user codes. In all the tables that follow, **N** refers to the number of SV1 processors involved in the particular run. It should be noted that multi-streaming currently cannot be mixed with autotasking in the same program. Thus the following tests only scale to four SV1 processors (one MSP).

The first set of runs used the program `t743lin1`. In addition to either `-Otask3` or `-Ostream3`, it was compiled with `-Oaggress,inline4,vector3`. For this set of runs, it executed 500 times steps.

N	User	Sys	Total	Elapsed	Opt
1	1928.7	17.7	1946.4	2980.5	`task3`
4	2659.5	102.6	2762.1	1285.7	`task3`
4	2767.2	2.9	2770.1	696.6	`stream3`

The Total CPU times used by the `-Otask3` and `-Ostream3` versions are essentially the same. However, the elapsed time for the MSP job is significantly less. Monitoring the jobs with `/etc/top` showed that once the MSP job started running, it received 100% of the available processor time until it completed; the operating system never descheduled it. The autotasked job, by comparison, was sharing its CPU resources with other jobs. Changing the Unix `nice` values had the predictable effect on the autotasked job; it appears to have absolutely no effect on the MSP job.

However, when four MSP jobs were submitted concurrently, the operating system did timeshare the MSP processors between the four jobs:

N	User	Sys	Total	Elapsed	Opt
4	3135.5	3.8	3139.3	1558.4	`stream3`
4	3083.0	4.6	3087.6	1544.8	`stream3`
4	3103.3	3.6	3106.9	1557.4	`stream3`
4	3123.6	3.6	3127.2	1560.4	`stream3`

The next test involved the lattice gauge program `xqcd_hot`. This program belongs to a user who, along with his collaborators, account for almost 15% of the total annual PVP allocation. Again, the code was compiled with `-Oaggress,inline4,vector3` and either `-Otask3` or `-Ostream3`.

N	User	Sys	Total	Elapsed	Opt
1	4594.8	54.8	4649.6	9637.8	`task3`
4	5141.4	352.4	5493.8	3204.3	`task3`
4	5787.3	7.1	5794.4	1778.9	`stream3`

As with the previous results, CPU times are comparable, while elapsed times are significantly different due to operating system scheduling (i.e., not due to code performance). Of more interest to the owner of this code is reproducibility. When running in autotasked mode, the code produces slightly different answers every time it is run, due to the non-deterministic scheduling of the individual process threads. The MSP version, however, produces identical results each time is run.

The final test of Bhaskara's multi-streaming processors utilized the supernova simulation code `classic`. As with the previous codes, this program was compiled with `-Oaggress,inline4,vector3`.

N	User	Sys	Total	Elapsed	Opt
1	710.6	13.2	723.8	1561.1	`task3`
4	1064.1	51.2	1115.3	726.3	`task3`
4	2944.7	5.3	2950.0	1122.5	`stream3`

These results came as a complete surprise, and we have been unable to explain them. We are continuing to investigate why the MSP performed so poorly with this code.

Initial SV1e Results

While the focus of this work has been on NERSC's production environment, and the challenges that this environment creates in trying to evaluate performance, we were also able to get some preliminary results from a 32-processor SV1e that Cray has made available for benchmarking. This new model features a faster processor (500 MHz vs. 300 MHz for the standard SV1), and improved cache performance.

Once again using the fusion code `t743lin1` running 250 time steps, we ran a series of comparisons between Bhaskara and the SV1e. First, results from Bhaskara using the compiler options (`-O3 -G2`) typically selected by the user who provided the code:

N	User	Sys	Total	Elapsed
1	944.6	11.2	955.8	2165.6
2	1236.7	11.8	1248.5	1261.0
4	1313.4	33.7	1347.1	932.0

Next we have Bhaskara results using slightly more aggressive optimization flags (`-Oaggress,inline4,vector3`) and either `-Otask3` or `-Ostream3`:

N	User	Sys	Total	Elapsed	Opt
1	924.6	13.0	937.6	2531.8	`task3`
2	1212.4	14.4	1226.8	1202.0	`task3`
4	1252.9	50.0	1302.9	1002.9	`task3`
4	1339.4	2.1	1341.5	341.1	`stream3`

On the SV1e, we used the same set of compiler options as in the previous results from Bhaskara (`-Oaggress,inline4,vector3`). The first set of runs were performed while the machine was not dedicated, but neither was it being heavily used:

N	User	Sys	Total	Elapsed	Opt
1	600.6	1.4	602.0	613.7	`task3`
2	723.6	0.5	724.1	362.8	`task3`
4	856.5	0.4	856.9	214.9	`task3`
4	877.2	0.4	877.6	219.6	`stream3`

Our final runs were done on the SV1e in a totally dedicated mode, and as expected, show the best performance:

N	Use	Sys	Total	Elapsed	Opt
1	529.8	0.1	529.9	530.3	`task3`
2	628.0	0.1	628.1	314.4	`task3`
4	765.1	0.2	765.3	191.8	`task3`
4	752.1	0.3	752.4	188.3	`stream3`

Conclusions

Our primary intent with this work was to investigate how NERSC's heavily loaded production environment affects users' efforts to optimize codes. A secondary goal was to determine whether allowing NERSC users to exploit the SV1's MSP capability would have a negative impact on overall system throughput.

The largest obstacle we identified trying to characterize performance was the variability of CPU charges based on system load. User codes running on the 16 processor machine regularly use about 20% less CPU time than the same code running on one of the 24 processor machines. Even on a single machine, a given code can exhibit wide variations in CPU charges depending on the particular job mix present at the time.

This variable has other implications as well. In particular, it can interfere with a PI's ability to accurately estimate their resource requirements and to manage their finite annual allocation.

Our experiences with the MSP were not encouraging. From a user's point of view, there were two primary benefits: much-reduced elapsed time, and the ability to get absolutely reproducible results. However, neither of these characteristics is a result of any intrinsic architectural advantage of MSP. Rather, they are related to operating system process scheduling. This behavior is incompatible with NERSC's priority class job scheduling and charging system.

Lastly, the inability to use more than one MSP processor (i.e., to mix tasking and streaming in the same job) means that MSP performance can easily be exceeded by autotasking with more than four SV1 processors.