Measuring Performance Degradation by CPU

Nicholas P. Cardo, Sterling Software, Inc., Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, M/S 258-6, Moffett Field, CA 94035-1000 USA

ABSTRACT: *Many conditions exist in a systems configuration that can affect overall system performance. This paper discusses an analysis performed where test programs were run on individual processors of a multiprocessor system. The test programs covered 3 areas: memory access, system calls, and disk I/O. Performance timings are tracked for each run across each individual CPU. Attempts to correlate anomalies to the system's configuration will be made.*

With the high costs associated with supercomputing and the ever increasing need to consolidate and reduce costs, it is very important to squeeze the most computing cycles out of purchased systems.

1 Introduction

Balancing system resources to achieve maximum utilization is a difficult task which infringes on all the system's components. An example of this would be the batch scheduler. An inefficient scheduler could fully utilize the processors without effectively utilizing memory. However, understanding the impact of the system's configuration on running processes can provide important insight into an unbalanced system. If the system is balanced properly, process timings would be nearly identical regardless of CPU, filesystem, or size.

2 Performance Test Approach

The general approach is to run programs designed to stress the access to resources. Measurements would be taken to track user, system, and elapsed timings for each program. In order to identify if any effects observed were due to normal system usage, testing was performed under two conditions: normal production and a dedicated system. Each test program was locked to each processor using `/etc/cpus`. Timings were recorded and plotted to identify any anomalous results.

3 Performance Tests

Three programs were developed to stress memory access, system calls, file I/O, and pure computational capacity.

3.1 Memory

The memory test is designed to consume all available memory and perform operations on each word of memory allocated. The systems configuration has 1024mw of memory, of which 1000mw is usable. The test program allocated a 970mw three dimensional array and performed movement of data between elements of neighboring dimensions. Because of the size of this test program, it could only be run on a dedicated system. The objective of this test is to identify if any processor incurs additional overhead when accessing memory.

3.2 I/O and System Calls

An inefficient program would be one that performs large amounts of poorly structured I/O operations or executes large quantities of system calls. This test is designed to be a program with large system overhead by issuing large quantities of I/O, one word in size. Additionally, the program issued eight million system calls. The system calls selected were ones that returned information from the kernel without performing any type of file I/O. The objective of this test is to identify if any processor requires more system time to complete the tasks.

3.3 Computations

In order to achieve maximum performance of a processor, codes need to do little to no I/O and be capable of producing large vector lengths. A test program was developed that performed no file I/O and produced a moderately large vector length. This objective of this test is to identify if a purely computational program is charged any system overhead on each processor.

4 Test Results

Each test was performed three times on each processor. The elapsed, user, and system times for each run were retained and used for analysis. The charts displayed for each test show their results.

4.1 Memory

Since this test required all available memory, the results were obtained on a dedicated system with no corresponding run from normal production.

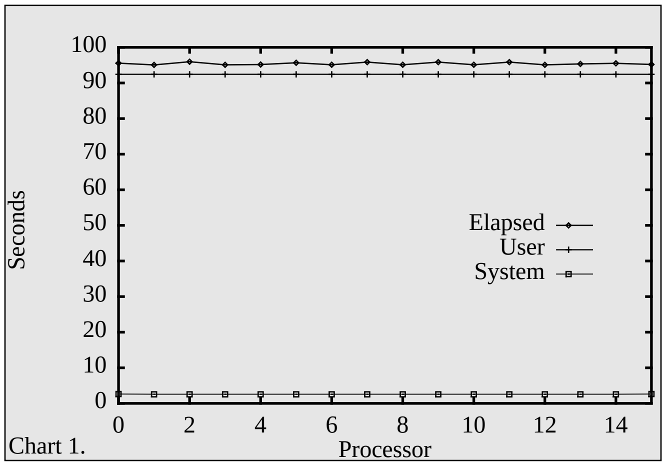

Chart 1.

When testing began, the results initially showed no change in timings between processors. Chart 1 represents a typical run of this test. However, when the length of time for the test was increased, an increase in system time was observed on processors 0 and 15. Chart 2 shows an expanded view of the system time changes on all CPUs. The change becomes more pronounced in longer running tests. Testing showed a 3.5% increase in system time on CPU's 0 and 15. An average increase of 3% was observed on both processors when the testing was run for longer time intervals.

4.2 I/O and System Calls

Initial testing showed a slight increase in system time of approximate 2% on CPU 0. To isolate this further, the I/O was removed from the test. The increase in time was reproducible at an increase of 3% to 3.2%. This increase remained regardless of the length of time for the test. Chart 3 shows representative data for runs during production. The spike in elapsed time for processors 5 and 6 shows contention for the processor. This shows that although the turnaround time for the program on processors 5

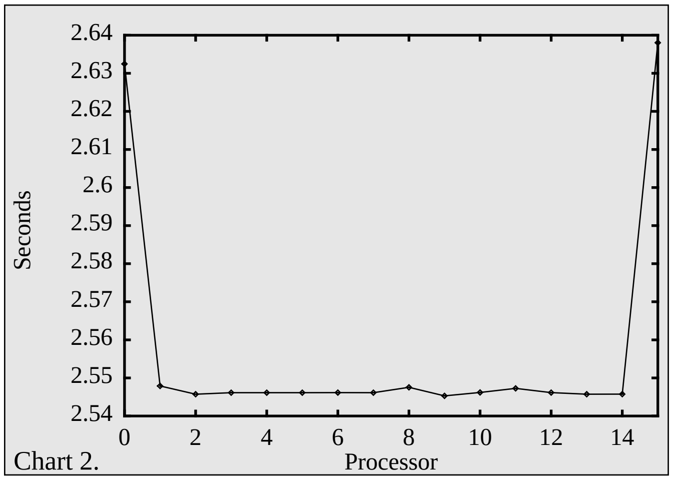

Chart 2.

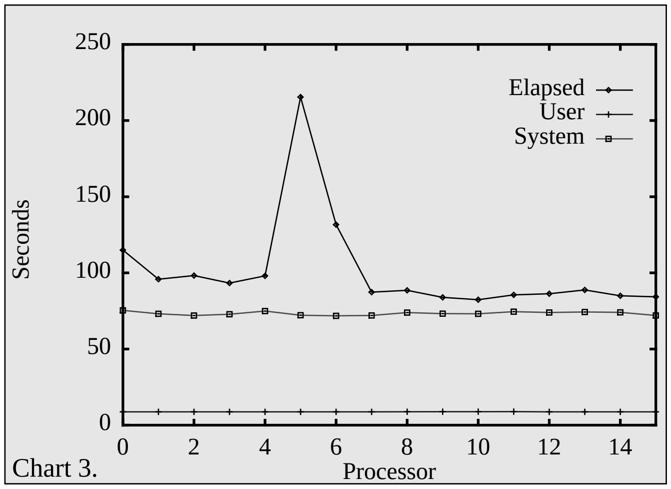

Chart 3.

and 6 was longer, the performance of the program did not change. The user and system times for the program remained as expected.

However, the increase in system time was not observed during dedicated testing. Chart 4 shows results gathered from some dedicated system processing. No anomalous results were encountered.

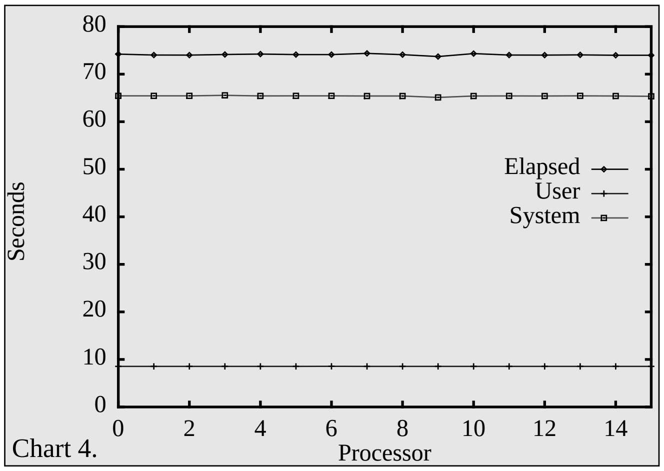

Chart 4.

4.3 Computations

By isolating the test program to only code which utilizes user time, the results clearly show an increase in system time on CPUs 0 and 15. Chart 5 shows the results from testing in normal production. The large elapsed time noticed on some processors is a result of processor contention.

Chart 6 represents only the system time and shows the increase in system time for CPUs 0 and 15. Also evident was that the system time on CPUs 1 through 7 was higher than the results on CPUs 8 through 14. Repeated tests consistently showed an increase in system time of as much as 700% with a mean of 276% over CPUs 8 through 14.

Although not as large, an increase in system time was observed on CPUs 0 and 15 when the test was run on a dedicated system. Chart 7 shows a typical pattern.

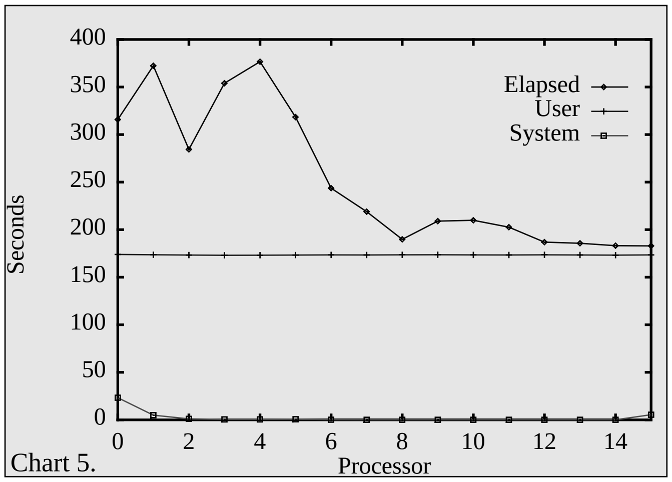

Chart 5.

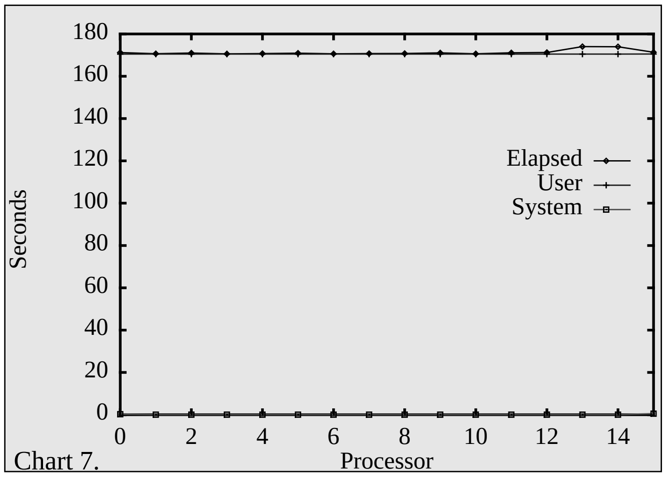

Chart 7.

A plot of the system time observed during a short time run on a dedicated system is shown in Chart 8. This pattern of CPUs 0 and 15 having higher system time was observed irregardless of the length of time the test was run.

5 Analysis and Correlation

The data showed a pattern of higher system time on CPUs 0 and 15. This pattern can be demonstrated on a fully loaded production system or on a dedicated system for running the test codes. This pattern seemed to exist only when the code running required very little system time.

Since I/O had been eliminated from testing, this left two possibilities: other programs and the kernel. To test for impact from other processes, the test programs were run on a dedicated system. The pattern was reproducible, therefore eliminating other processes leaving only the kernel.

This leaves two areas to investigate for the cause of this higher system time. The first area is the kernel itself while the second is hardware.

The tests were performed on a CRAY C90 16/1024. There are 4 vhisp channels connect via processors 0, 1, 2, and 3. The system's hisp channels connected via processors 8 - 15. The system heavily uses ldcache which can be associated with some elevations in system time on processors 0 - 3. However, processor 15 contains a hisp to an IOC and to main memory. Higher system times on processor 15 can be attributed to memory contention or I/O transactions. One possible memory operation that could affect results is memory compaction. However, it would be expected that elapsed time would increase as well.

Processor 0 consistently showed higher system time, some of which is associated with ldcache. When no processors are in the kernel, a usrpci (programmable clock interrupt) will interrupt processor 0. Since this occurs every 1/60th of a second, a large amount of overhead will be accrued on processor 0.

When a program that spends most of its time in the kernel is running, the increase in system time is minimal and not worth noting. One possible explanation for this is that since the processor is already in the kernel, the `usrpci` does not need to interrupt processor 0. The end result is that all processors perform equally.

The testing also shows that individual processes are being charged additional time for work they are not requesting when the program is attached to processor 0. The implication of this is that the kernel has no way to account for or properly charge for work it performs on behalf of itself.

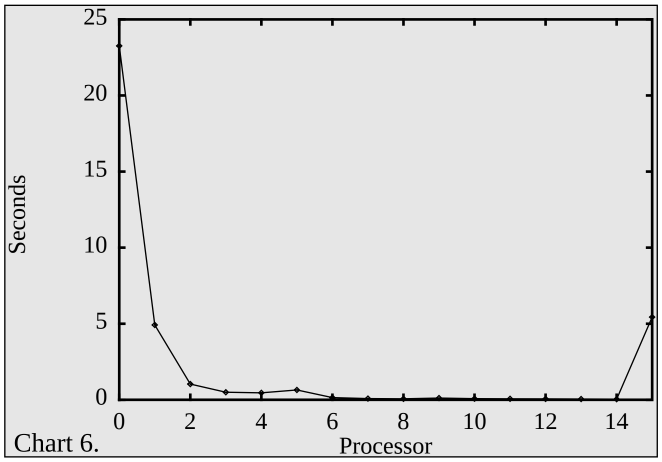

Chart 6.

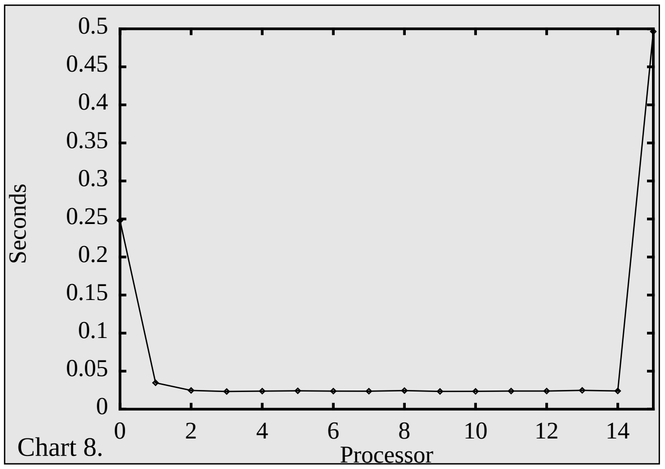

Chart 8.

Ideally, each processor should show exactly the same timings. In most cases, processors 1 through 14 provide similar timings. When differences did occur, they were typically found to be on processors 1 through 3 which can be attributed to ldcache activity.

6 Summary

The information discussed here presents an approach to evaluate the equality of processors as applied to individual processes. The test cases were developed to search out strange behaviors of the individual processors.

Two pieces of information can be concluded from the testing performed. The first is that the kernel inappropriately charges time to processes for work that it initiates. The kernel should be capable of accounting for itself properly by charging this time to possibly the `sched` process and not user processes. Additionally, this behavior applies to process initiated kernel activity. If a process is connected to a processor that is in the kernel, it can be charged for all the kernel activity outstanding when the processor enters the kernel.

The second item is that processor 0 was found to be called upon more often to do work. Interrupts for processor 0 should be capable of being distributed amongst the remaining processors.

United States sites are welcome to visit the Numerical Aerodynamic Simulation (NAS) Facility on the World Wide Web at `http://www.nas.nasa.gov`. The author can be reached at `cardo@nas.nasa.gov` for additional information.