Quantifying The Cost of Context Switch

Chuanpeng Li and Chen Ding and Kai Shen

Computer Science Department
University of Rochester
{cli,cding,kshen}@cs.rochester.edu

1 The Measurement Approach

Based on the traditional method, we used two processes communicating via a pipe. We first measure the direct time cost per context switch (c1) when processes make no memory access. Then we measure the total time cost per context switch (c2) when processes accesses different size arrays with a different amount of computation. The indirect cost is estimated as c2 - c1.

1.1 The direct time cost per context switch (c_1)

We have two processes sending a single-byte message to each other via a pipe. We measure the time cost of 1000 round-trip communications (t_1), which include 2000 context switches between the two processes.

We have a single process simulating two processes' communication by sending a single-byte message to itself. We measure the time cost of 1000 simulated round-trip communications (t_2), which include no context switch. The simulation process will do the same amount of work as the two communicating processes except for context switches. Therefore the time cost per context switch is $c_1 = (t_1 - t_2)/2000$.

Note that a process is forced to switch when it executes a blocking read from the pipe. To avoid the interference from other processes in the system, we use a dual-processor machine exclusively for our experiment. The two communicating processes are assigned to the same processor. We set up the test processes with real-time scheduling policy SCHED_FIFO and give them the maximum priority. Presumably, most kernel processes will run on the other CPU. And no other process can preempt any of our test process as long as our process is runnable. Linux system call *sched_setaffinity()* and *sched_setscheduler()* are used to effect the design.

1.2 The direct and indirect costs per context switch (c_2)

The control flow of this test program is similar to that of 1.1. However, after each process becomes runnable, it will do a certain amount of memory access and computation before it sends the message to the other process and then blocks on the next read operation. We still have a single process simulating the two processes' behavior except for context switches. The simulation process will do the same amount of memory access and computation work as the total of the two communicating processes. Assuming the cost of 1000 round-trip communications between the two test processes is (s_1) and the cost of 1000 simulated round-trip communications (s_2), we get the sum of direct and indirect costs per context switch $c_2 = (s_1 - s_2)/2000$.

We change the following 3 parameters during different runs of our test.

- *Memory size*: the total memory accessed by each process.
- *Access stride*: the size of the strided access.

- *Switch frequency*: the total amount of computation between two context switches. The basic unit includes strided access through the array. The frequency is reduced by running the basic unit multiple times. Inside each basic unit, the number of multiplication on each double data element affect the length of the unit.

1.3 Timer and measurement times

The timer we use is a high resolution timer that relies on a counting register in the CPU. It will report the number of cycles the CPU has gone through since startup. When the length of the measured event is very short, the timer itself may cause some error. Therefore, we measure the cost of a large number of context switches and then use the average cost.

2 Experimental Results

The machine platform we use is IBM eServer with dual 2.0 GHz Intel Pentium Xeon CPUs. Each processor has 16KB L1 data cache and 512KB L2 cache. The operating system is Linux 2.6.17 kernel with Redhat 9. The compiler is gcc 3.2.2. We do not use any optimization option for compilation.

The average direct context switch cost (c_1) in our system is 3.4 microsecond. The results shown below are about the total cost per context switch (c_2). In general, c_2 ranges from several microseconds to hundreds of microseconds. The indirect context switch cost can be estimated as c2 - c1.

We discuss two sets of experiments in detail here. The first tests the effect of memory hierarchy by varying the size of the data used by the two processes. The second tests the effect of the access stride on the cost of context switch.

2.1 Effect of data size

We show the data size and the cost of context switch of 22 experiments in figure 1. In each experiment, each process traverses an array of size d (in 8-byte double numbers) between two context switches. Each process does not do any computation on each array element. The upper graph shows the value of d in each experiment. The lower graph shows the average cost of 2000 context switches measured in each experiment. The numbers form roughly three tiers. The first tier includes the first six experiments, which use two data sizes 8 bytes and 64KB. The time of context switch ranges between $3.7\mu s$ and $4.4\mu s$ with the mean at $4.1\mu s$. Considering that the L1 data cache is 16KB, we conclude that the first tier represents the cost of refilling L1 cache and some of L2 cache. Similarly, the cost of the next tier, between $6.0\mu s$ and $18.6\mu s$ with the mean $12.1\mu s$, represents the cost of refilling L1 and L2 cache. The last tier shows a cost between $18.5\mu s$ and $28.5\mu s$ with the mean $22.4\mu s$.

Looking at the three tiers from left to right, the cost of context switch increases when the data size increases due to the overhead of L2 cache refilling. But within last tier, where the data size is well above the cache, the cost of context switch stop increasing. This is because the data size is too big, cache misses will happen even there is no context switch.

2.2 Effect of access stride

We show the access stride and the cost of context switch of 51 experiments in figure 2. In each experiment, each process access an array of 8-byte double numbers in the following strided pattern. Starting from the first element, it accesses every s element. Then starting from the second element, it accesses every next s element. The process repeats striding until every element of the array is accessed. If s is one, the access pattern is actually sequential. We show array of size between 640KB and 1MB (in 8-byte double numbers). We choose this array size based on the first set of experiments, which show that these array sizes do not affect the cost of context switches differently.

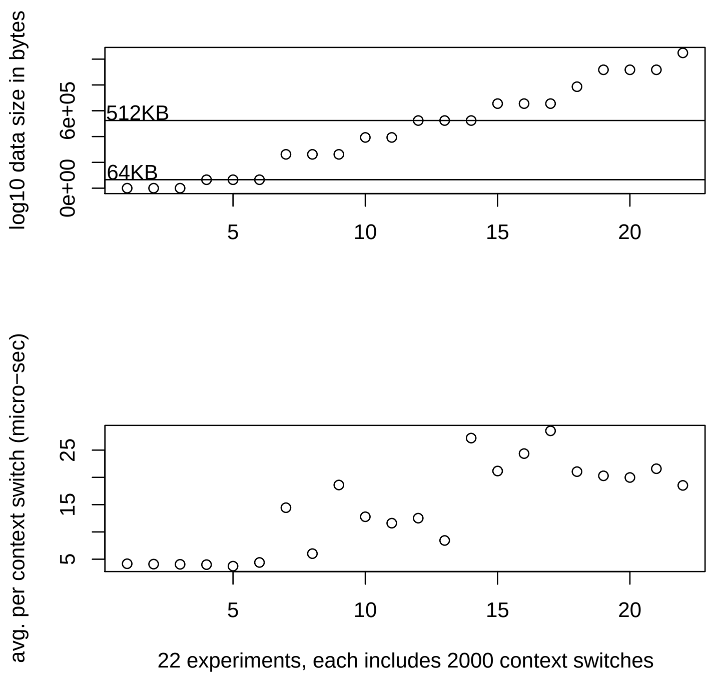

Figure 1: The effect of data size on the cost of the context switch

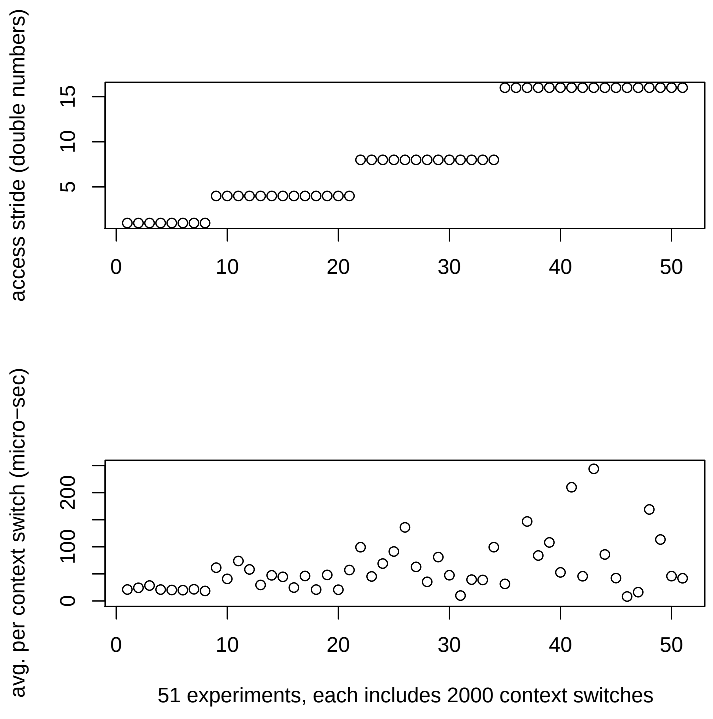

Figure 2: The effect of the access stride on the cost of context switch

The upper graph shows the value of s in each experiment. The lower graph shows the average cost of 2000 context switches measured in each experiment. As the access stride increases from 1, 4, 8, to 16, the average cost becomes greater varied and generally higher. When the access stride is one, the cost ranges between $18.5\mu s$ and $28.5\mu s$ with the mean $21.9\mu s$. When the access stride is four, the cost ranges between $20.8\mu s$ and $73.9\mu s$ with the mean $44.2\mu s$. For the access stride of eight, the cost ranges between $9\mu s$ and $136\mu s$ with the mean $65.8\mu s$. And for the access stride of 16, the cost ranges between $8.2\mu s$ and $455\mu s$ with the mean $112\mu s$.

The progression of the cost increase, from on average $22\mu s$ to $112\mu s$, is significant, and it is caused only by the increase in the access stride. In other words, the way how the cache is filled up can affect the cost of context switch significantly. One likely explanation is that the hardware cache prefetching works well for sequential memory access. The larger the access stride is, the worse the cache prefetching works.

2.3 Effect of switch frequency

After collecting results for thousands of experiments, we found that the cost of context switch grows when the frequency of context switch drops. In other words, the cost grows with the amount of intervening computation between two context switches. Due to the limited time we have, we will assemble these results in the next version of the report.

3 Observations

Through the experiments we make the following observations.

- In general, the indirect cost of context switch ranges from several microseconds to hundreds of microseconds.
- The cost of L1 cache refill after a context switch contributes little or no additional time but the cost of refilling of L2 cache adds on average $8\mu s$ for 512KB cache on our first test.
- The effect of access stride on the cost of context switch is significant. We saw average cost of context switch increases from $22\mu s$ to $112\mu s$ when the access stride increases from one to sixteen in our second test.