Assessing the Cost of Context Switch

Angelo Acevedo
Computer Science
Brooklyn College
angeloacevedo9@gmail.com

Nay Ayeyar
Computer Science
Brooklyn College
NAYAYEYAR2230@gmail.com

Luka Nikabadze
Computer Science
Brooklyn College
niqabadzelukal@gmail.com

Hareem Bokhari
Computer Science
Brooklyn College
hbokhari98@gmail.com

ABSTRACT

In this project, we quantify the indirect cost of context switch by replicating the experiment in the research paper, *Quantifying The Cost of Context Switch[1]*.

KEYWORDS

Context switch, Indirect Context Switch, Processor affinity, Cache memory

1) Introduction

We replicate the efforts done in, *Quantifying The Cost of Context Switch,* in which we measure the impact of array data size and access stride on context switch cost. We explore the concepts of context switch, processor affinity and cache memory and their relations to measuring the cost of context switch.

1.1) Context Switch

Context Switch is the switching of the CPU from one process or thread to another. Context switch makes multitasking possible. Additionally, it causes unavoidable system overhead (excess resources that the CPU must expend).
Cost of a context switch can be attributed to multiple factors. It can be categorized into two parts: direct cost and indirect cost.

Direct Cost: The processor registers need to be saved and restored, the OS kernel code (scheduler) must execute, the TLB entries need to be reloaded, and the processor pipeline must be flushed.

Indirect Cost (cache interference): Context switch leads to cache sharing between multiple processes, which may result in performance degradation. This cost varies from different workloads with different memory access behaviors and for different architectures.

1.2) Process Affinity

When a thread has been running on a specific processor, the data most recently accessed by the thread populate the cache for the processor. As a result, successive memory accesses by the thread are often satisfied in cache memory (known as “warm cache”)

If the thread moves to another processor—for example, due to load balancing. The contents of the cache memory must be invalidated for the first processor, and the cache for the second processor must be repopulated. There is a high cost when it comes to invalidating and repopulating caches. Most operating systems with SMP support try to avoid migrating a thread from one processor to another, and instead attempt to keep a thread running on the same processor and take advantage of warm cache.

In order to avoid potential interference from background processes and preemptive scheduling, we use sched_affinity() and sched_setscheduler() to bound the experimental programs to possess their own processor.

This is known as processor affinity—a process has an affinity for the processor on which it is currently running on.

1.3) Cache Memory

Cache Memory is a special very high-speed memory. It is used to speed up and synchronize with high speed CPU. It is costlier than main memory or disk memory but economical than CPU-registers. It acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when called upon.

When a processor needs to read or write to a location in main memory, it first checks for correspond entry in the cache.

Cache hit: Occurs when a processor finds the memory location in the cache and data is read from the cache.

Cache Miss: If the processor doesn’t find the memory location in the cache. The cache then allocates a new entry and copies in the data from main memory, then the request is fulfilled from the contents of the cache.

This is integral to the experiment, because there is cache involvement when the array size is bigger than the cache. Cache invalidating and repopulation causes high cost of the system.

2) Experiment

Following the method in the report [1], we have two programs to measure the total time cost per context switch.

In the first program (s1): we use a single process simulating two processes communicating by sending a single-byte message to itself via pipe. 10,000 simulated round-trip communications.

In the second program (s2): We have two processes repeatedly sending a single-byte message to each other via two pipes. After each process becomes runnable, it will access an array of floating-point numbers. 10,000 round-trip communications

Running on :

Virtual Machine; Debian Operating System
Intel ® Core(TM) i5-5257U CPU @ 2.70 GHz
L2 cache: 512 KB
L3 cache: 3 MB

2.2) Parameters

In measuring the *effect of array size on context switch*—both the simulated and unsimulated communications— we chose a constant stride size of 16 bytes, because in order to get a better understanding of the cost of context switch in regards to the array size, than we need to isolate our stride value. Our array size ranged from 64 kb to 8 MB.

In measuring the *indirect cost of a context switch*, we change both the array size and stride size. The idea of change both parameters give us a better understanding of how different stride sizes can affect the cost of context switching.
The stride size ranges from 8 bytes to 128 bytes. Array size ranges from 32 KB to 2 MB.

As compared to the original experiment by Chuanpeng Li, Chen Ding, and Kai Shen, the parameters of our experiment differ, the rapid advancement of hardware that took place between the original experiment to date contributes to the parameter choice. The original experiment uses maximum of 2048 KB of Array size due to hardware restrictions in 2007[1], but because of the development of larger sized hardware, we have the opportunity to use Array sizes up to 8000KB ~ 8MB in our experiment.

3) Experimental Results

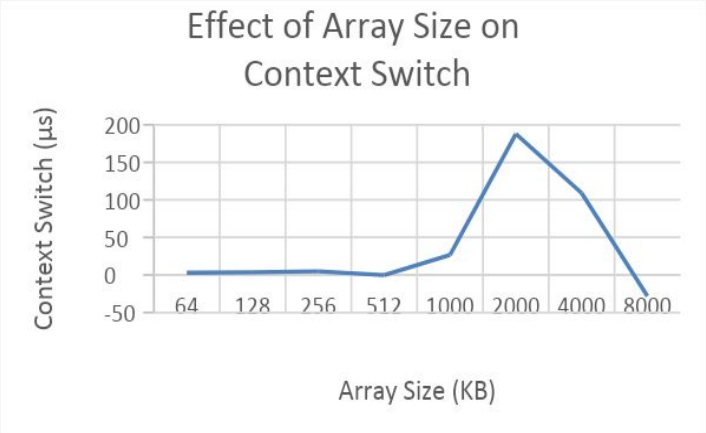

Figure 1: *Results from measuring cost of context switch in relation to array size. Experiments performed on a virtual machine running Debian Operating System, with a 2.7GHz CPU with 512 KB L2 and 3MB L3 and a stride of 16B.*

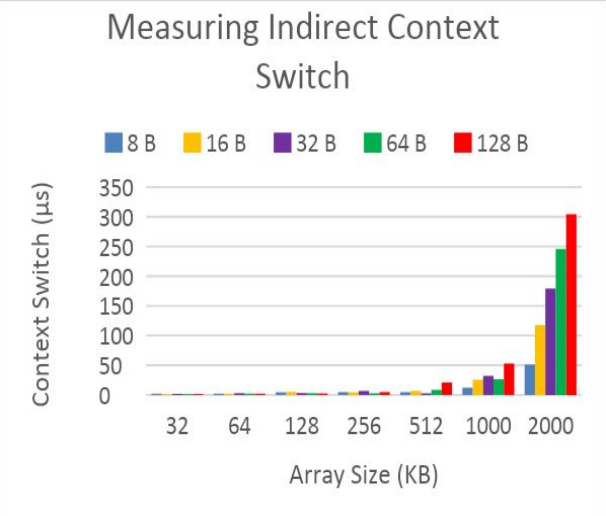

Figure 2: *Results from measuring indirect cost of context switch in relation to varying stride size and array size. Experiments performed on a virtual machine running Debian Operating System, with a 2.7GHz CPU with 512 KB L2 and 3MB L3 and a stride of 16B.*

In Figure 1, using a constant stride size, we observe that when the array size from 64-512 KB the curve is relatively flat, because the data can fit into the L2 cache, and no cache interference is present. From 512KB to 8MB, we notice a significant increase in the cost of context switch. When the data size is larger than the cache size, the overheads of repopulating the L2 cache has significant impact on the cost of context switch. Additionally, we note the presence of cache interference.

In Figure 2, we analyze the indirect cost of context switch with different values of array, and stride, size. The greater the stride, the greater the cost of context switch is present. When the dataset doesn't fit into the cache, the cost of context switch increases dramatically. Stride size affects the cost of cache warm-ups in a similar way it affects program running time.

REFERENCES

[1] Chuanpeng Li, Chen Ding, and Kai Shen. Quantifying the cost of context switch. In Proceedings of the 2007 Workshop on Experimental Computer Science, ExpCS '07, New York, NY, USA, 2007. ACM.