Моделирование и анализ информационных систем. Т. 25, № 2 (2018), с. 165–173
Modeling and Analysis of Information Systems. Vol. 25, No 2 (2018), pp. 165–173

DOI: 10.18255/1818-1015-2018-2-165-173

UDC 004.051; 004.451.35; 004.451.46

Measuring Overhead of Concurrency and Virtual Memory

Yauhen Klimiankou

Received November 7, 2017

Abstract. We present the methodology, as well as results of measurements and evaluation of overhead created by concurrency and virtual memory. A special measurement technique and testbed were used to obtain the most accurate data from the experiments. This technique is focused on the measurements of the overall performance degradation that is introduced by concurrency in the form of lightweight user-level threads on IA-32 processors. We have obtained and compared results of the experiments in an environment with and without enabled virtual memory to understand what loss of performance is caused by virtual memory in itself, and how it affects the overhead associated with concurrency. The results showed that overhead of concurrency outweighs virtual memory overhead and that there is a complex dependency between them. The article is published in the author's wording.

Keywords: virtual memory, concurrency, overhead, performance, measurements

For citation: Yauhen Klimiankou, "Measuring Overhead of Concurrency and Virtual Memory ", *Modeling and Analysis of Information Systems*, **25**:2 (2018), 165–173.

On the authors:
Yauhen Klimiankou, orcid.org/0000-0001-7449-7986, MSc, PhD student
Belarusian State University of Informatics and Radioelectronics,
6 P. Brovki Street, Minsk 220013, Belarus, e-mail: klimenkov@bsuir.by

1. Introduction

In this paper we describe a work motivated by the desire to understand the influence of concurrency and virtual memory to the performance of the system. To understand that, we have measured the overhead created by the multithreading in its lightest-weight form called fibers. Fibers are used to provide concurrency on the application level. Additionally, we wanted to find out how the virtual memory affects performance of both concurrent and serialized workloads.

Accurate and fine-grained performance measurements on the low level of the system is challenging and requires taking into account a number of features of system and CPU. We have used a benefit of having of our own research operating system to create the вЂњClean RoomвЂќ environment, in which we have eliminated interference of the measurements with other activities asynchronously going in the system.

2. Related Work

There are a variety of works done to understand a context switch overhead, yet none of them has considered concurrency in general on the modern CISC processors. It is hard to find the published results of actual measurements of virtual memory overhead. Moreover, interdependence between concurrency and virtual memory overheads was not investigated.

The first research in that field was done by Agarwal et al. [1], where it demonstrated that multiprogramming activity significantly degrades cache performance. After that, Mogul advanced and widened original research [8]. In both cases results were achieved through simulation but not actual experiments and concurrency were considered in the heavyweight form – multiprocessing. Additionally, CPUs have significantly advanced during the last two decades. There is a concern that both works do not reflect the behavior of the modern CPUs.

McVoy et al. measured the cost of context switch between multiple processes using lmbench [7]. While this work provides results of experiments done in real-world environments, it considers multiprocessing rather than concurrency in general and accounts not only CPU but also OS overhead.

Li et al. has conducted a research similar to our work [5]. He has quantified context switch cost, but like in previous cases, he has focused on Linux multiprocessing, and had not cut off overhead created by contention on memory bus.

Finally, the David et al. [2] has considered concurrency in general instead of multiprocessing. But his measurements were performed on RISC processor (ARM), while we were interested in understanding of concurrency overhead created on advanced and full-featured CISC processors (IA-32).

We are not aware about any good paper with evaluation of virtual memory overhead. It is interesting, but the only paper that sheds some light to this issue is an overview of Singularity OS [4]. But even it considers entire overhead created by virtual memory and does not dig into details.

3. Measurement Methodology

Our measurement methodology was designed to provide the cleanest and accurate results with a fine-grained resolution. We have eliminated interference with interrupt handling and operating system kernel scheduler on macro-level. Furthermore, we have taken into account potential disturbance of the branch prediction, influence of the measurements on the instrumented code execution time and potential issues related to caching on micro-level.

3.1. System Structure

Experimental setup is depicted on Figure 1. We have used an advantage of having our own experimetal operating system to create testbed. In particular, the OS loader was modified by injections of three code modifications.

One of the injections is used to disable multiprocessor boot. By this, influence of the contention on memory bus to the results of measurements was eliminated. Despite the

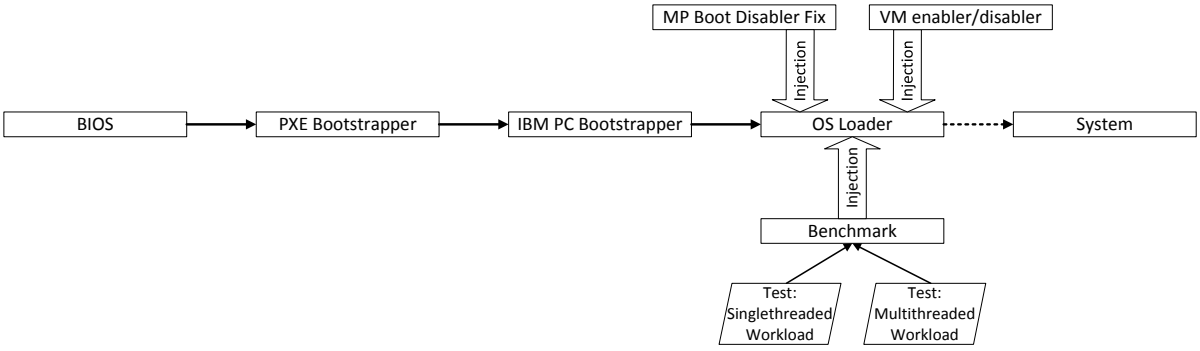

Fig 1. Experiment setup

fact that computer system equipped by two CPUs was used during performance tests, only one CPU was bootstrapped and used. The second one was leaved in the uninitialized state. In addition, the Hyper-Threading technology was disabled through the BIOS for the time of all experiments.

The second code injection has added the switch which controls enabling of the virtual memory. Availability of this switch has allowed to consider influence of virtual memory to the system performance.

Finally, the last code injection contains microbenchmark with two test cases. The first test case reproduces batched singlethreaded processing of workload. In the second scenario, the same workload is processed concurrently by two user-level threads. Benchmark finishes its work by halting the future loading of the operating system, which is unneeded and unsafe.

Benchmark applications use the same preallocated buffer in both test scenarios.

3.2. Benchmark Implementation

Benchmark starts work by disabling interrupts. By this, we eliminate potential interference of measurements with interrupt handling, which can affect accuracy of the measurement results.

Before the start of measurements, user-level multithreading environment is initialized. In our experiments only two threads are in use. Thus thread switch pointer should be set up and two stacks should be allocated and prepared. Each user-mode thread uses its own "top of the stack" pointer, which is also initialized during preparation to measurements, as well as initial stack frame.

Measurements is taken for a number of different workload sizes starting from 0 and ending by 24MB. Due to this benchmark performs a number of iterations of measurements, each time doubling the workload (0, 1b, 2b, 4b, ..., 24MB). On each iterations two experiment scenarios are executed: singlethreaded and multithreaded and, therefore, two measurements are taken. Before each experiment, cache warmup is performed by run of few experiments which execution time is not accounted for in measurement results. Finally, two experiment scenarios are played.

During experiment we emulated work of the application with variable working sets. Two buffers of the same fixed size were used to emulate working sets. The goal of the application was zeroing of buffers. Each scenario was replayed 100000 times.

In the singlethreaded scenario application sequentially refills first buffer by zeroes. In such a way it emulates the first program working with data in its working set. After the first buffer will be refilled 100000 times, application repeats the same procedure for the second buffer, emulating the second sequentially executing program.

In the multithreaded scenario, both emulated applications execute concurrently. After each buffer refill context switch to another application is performed. This process continues until both buffers will be refilled 100000 times.

3.3. User-Level Threads Implementation

The primary goal of our experiments is to understand the impact of concurrency to the performance, but not the impact of some specific implementation of concurrency. Due to this the lightest form of multithreading was chosen.

Implemented multithreading environment supports only two user-level threads and is based on cooperative form of multitasking. As a result, it does not require presence of scheduler. Additionally, run queue has degraded to the switch point, because we always have a system in a state where one thread is running and one thread is in ready state. Hence, context switch can be considered as a simple swapping of the roles/states between these two threads. Due to this environment maintains minimal global state consisting only of one variable, which stores stack pointer of the thread that is in the ready state. State of the user-level threads includes only state of eight general purpose registers of IA-32 processor. During context switch, outgoing thread stores its state on the top of its own stack. Then it swaps content of the global state variable and stack pointer register of processor. Finally, incoming thread loads its state from the top of its own stack.

The function for the user-level thread switch is implemented as follows:

```
; ECX - address of global state
SwitchThread:
    pushad
    mov    EAX , [ECX]
    mov    [ECX] , ESP
    mov    ESP , EAX
    popad
    ret
```

3.4. Measurement Methodology and Points

The primary tool which was used for execution time measurements is a IA-32 time stamp counter. Time stamp counter provides a highest resolution with granularity equal to a tick of system bus. It was safe to use time stamp counter in our experiments, because we had full control over CPU and were assured that thread migration to another CPU is prevented.

To reduce the effects of the measurements on the results, the number of the measurements taken was minimized. Furthermore, all measurements were done using the same framework, where only test scenarios were replaced. As a result the difference in measured time for two experiments will show only difference in execution time of test scenarios. Framework and test scenarios are depicted on Figure 2.

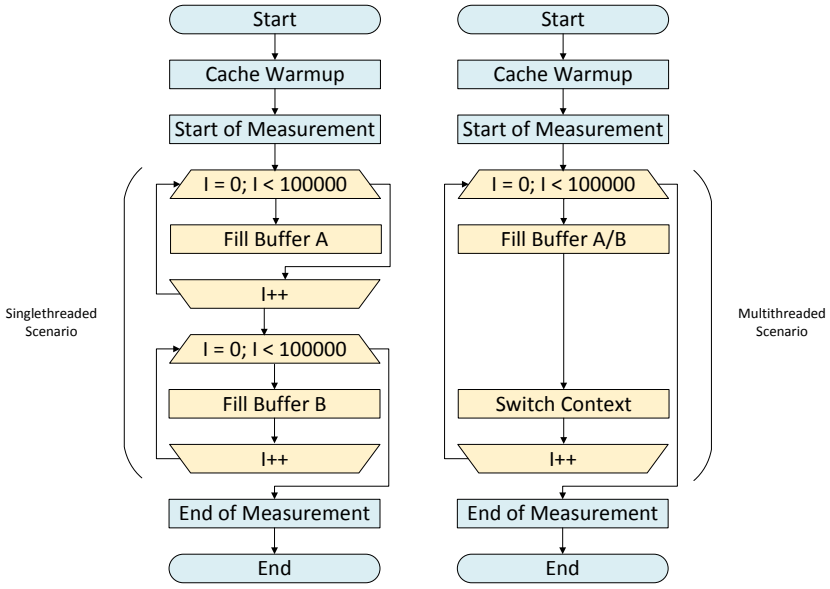

Fig 2. Experiment Scenarios

In our approach we have taken into account all effects of advanced processor features such as superscalarity, branch prediction and out-of-order execution, because we wanted to know the full impact of multithreading and virtual memory on the performance. Due to this the overall time of the entire series of 100000 experiments was measured. Note that in multithreaded experiment scenario two iteration counters were actually used, one per each thread stack. Thus, the entire experiment actually performs 200000 buffer refills, 100000 per each buffer.

3.5. Experimental Platform

All measurements were done on the Dell PowerEdge 2650 server [3] with two Intel Xeon CPUs [6] (Details are provided in Table 1).

Table 1. Experiment platform

CPU	Intel® Xeon® Processor
Microarchitecture	NetBurst
Codename	Prestonia
Frequency	2.4 GHz
TLB	64 + 64 entries
L1 code cache	12 K-Ojop 8-way set associative
L1 data cache	8 KByte 4-way set associative 64 byte line size
L2 cache	512 KByte 8-way set associative 64 byte line size

4. Results

Figures 3 and 4 visualize the measured overhead imposed by concurrency in the form of light-weight user-level multithreading as discussed in section 3.3. Both figures contain two graphs that show overhead measured in the environment with enabled and disabled virtual memory. Figure 3 represents an absolute overhead measured during a series of experiments and Figure 4 shows the same results, but in a form of relative overhead.

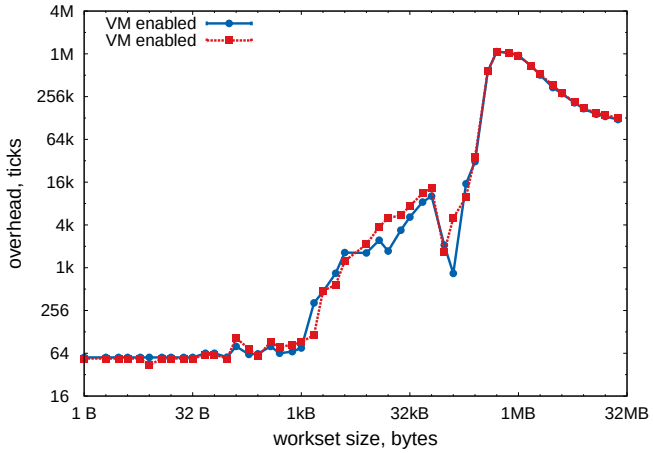

Fig 3. Absolute overhead of multithreading

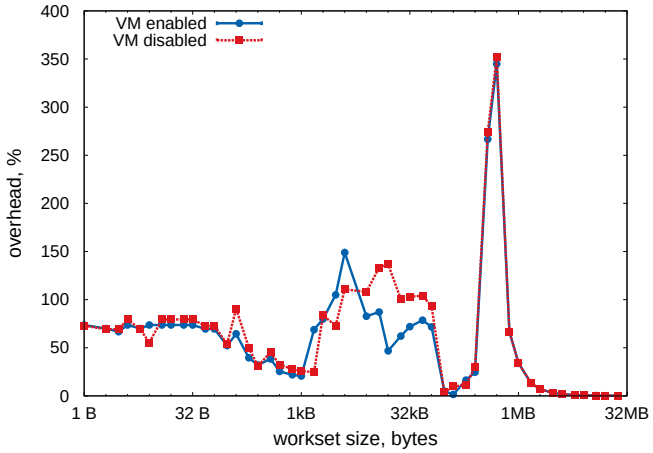

Fig 4. Relative overhead of multithreading

As reader can see, concurrency always creates performance penalty which is in the worst case raises up to almost 1.125 millions of wasted CPU ticks per time quantum used by thread or 352% of overhead. This worst case represents the scenario with working set of size 0.5MB, which is a size of L2 cache in our experimental setup. In the case of absence of concurrency entire working set is loaded into the cache once, where it is processed multiple times. But in the case of concurrency entire cache is considered as completely polluted and thus is constantly reloaded after each context switch. In general, overhead is significant and reaches acceptable levels only for working set sizes of near 128KB and bigger than 2MB.

Figure 5 shows how enabled virtual memory affects the overhead imposed by concurrency. In most cases it reduces performance penalty created by concurrency. In some cases even significantly (up to -90% for the case of working set of 16KB). In other cases, enabled virtual memory can boost concurrency overhead (up to +44% for the case of working set of 1.5KB). It would be interesting to note that influence of enabled virtual memory on concurrency overhead for the scenarios with working sets bigger than 0.5MB is negligible.

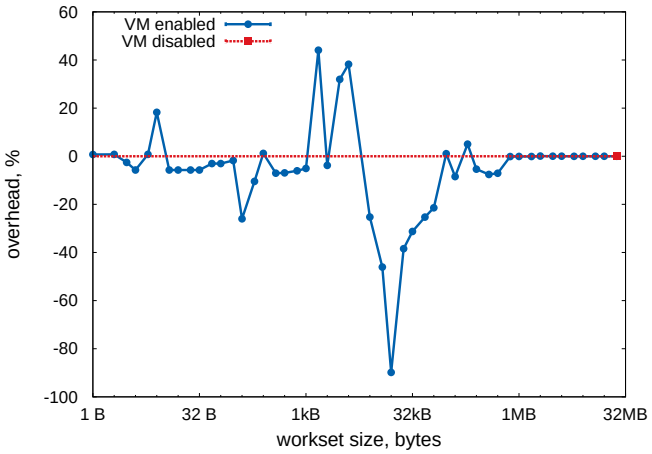

Fig 5. Impact of virtual memory onto the concurrency overhead

Another aspect discussed here is how the enabled virtual memory affects performance of the application and concurrency overhead. Figures 6 and 7 shows the results of measurements of overhead for enabled virtual memory. As in the previous case, Figure 6 represents performance penalty in absolute values, when Figure 7 represents the same results but as relative overhead.

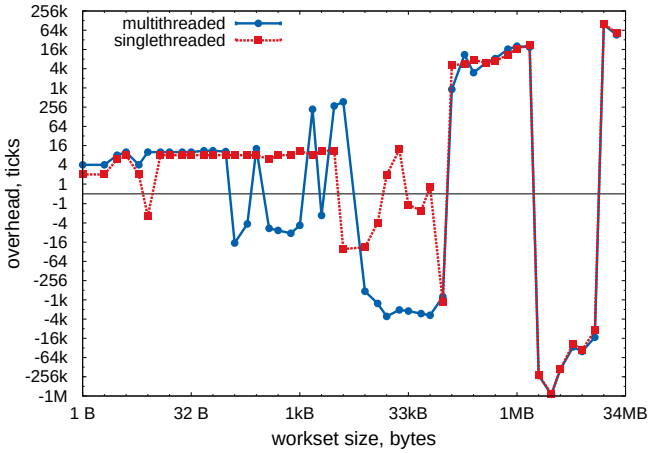

Fig 6. Absolute overhead of virtual memory

It is interesting to note that in contrast to the concurrency, impact of the virtual memory to the application performance is not trivial. In most cases enabled virtual memory degrades performance of the application, or its impact is negligible. But actually,

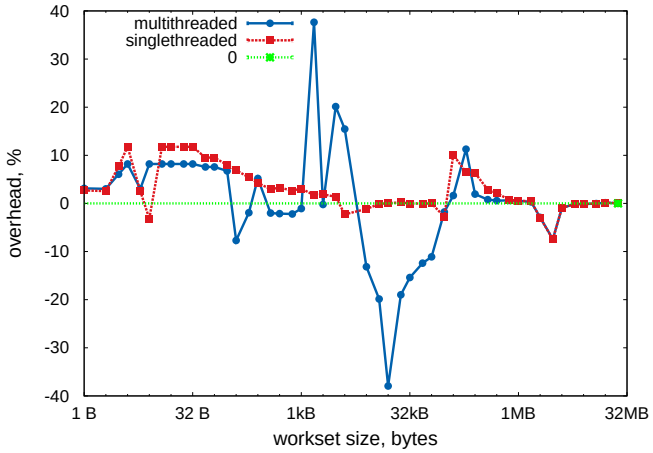

Fig 7. Relative overhead of virtual memory

we can see the cases when virtual memory boosts application performance. What is even more interesting, that boost is significant for the concurrent workload (up to 39% for 16KB working set) when it negligible for the serialized workload (up to 7.5% for 3MB working set). Impact of concurrency onto virtual memory overhead is demonstrated on Figure 8.

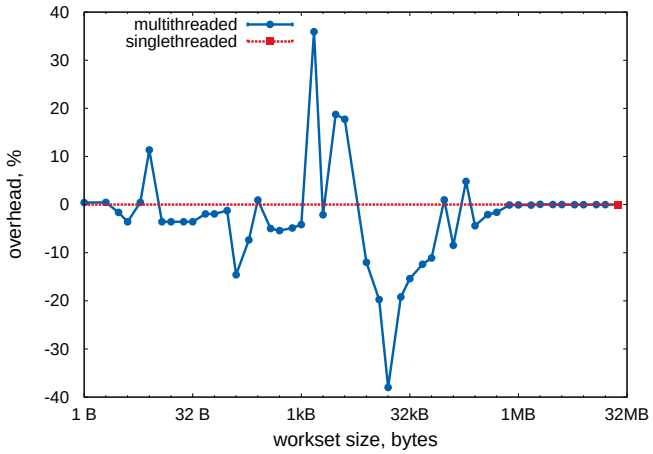

Fig 8. Impact of concurrency on virtual memory overhead

5. Conclusions and Outlook

We presented the results of the measurements of the overhead incurred by concurrency and virtual memory. During measurements special focus was pointed to achievement of the most accurate and fine-grained results. Due to this the “Clean room” methodology was used.

We have demonstrated that concurrency creates significant overhead which can grow up to 350% in a terminal case. Reason for this is an advanced multilevel caching used

by CPUs to hide the cost of access to main memory and by cache pollution introduced by concurrency.

In contrast, impact of virtual memory on the system performance is not trivial. Furthermore, it largely depends on the type of workload. In particular, we have found an interesting phenomena when virtual memory even significantly boosts performance of concurrent workload (up to 39%).

References

[1] A. Agarwal, J. Hennessy, M. Horowitz, "Cache Performance of Operating System and Multiprogramming Workloads", *ACM Trans. Comput. Syst.*, **6**:4 (1988), 393–431.

[2] F. M. David, J. C. Carlyle, R. H. Campbell, "Context Switch Overheads for Linux on ARM Platforms", *Proceedings of the 2007 Workshop on Experimental Computer Science, ExpCS'07*, ACM, New York, NY, USA, 2007.

[3] *INFOBrief: Dell PowerEdge 2650*, Dell Inc., 2004.

[4] G. Hunt, J. Larus, "Singularity: Rethinking the Software Stack", *ACM SIGOPS Operating Systems Review*, **41**:2 (2007), 37–49.

[5] C. Li, C. Ding, K. Shen, "Quantifying the Cost of Context Switch", *Proceedings of the 2007 Workshop on Experimental Computer Science, ExpCS'07*, ACM, New York, NY, USA, 2007.

[6] *IA-32 Intel® Architecture Software Developer's Manual. Volume 3: System Programming Guide*, Intel Corporation, 2002 (245472-007).

[7] L. McVoy, C. Staelin, "Lmbench: Portable Tools for Performance Analysis", *Proceedings of the USENIX Annual Technical Conference*, San Diego, California, USA, January 22–26, 1996, 279–294.

[8] J. C. Mogul, A. Borg, "The Effect of Context Switches on Cache Performance", *SIGPLAN Not.*, **26**:4 (1991), 75–84.

Клименков Е. И., "Измерение накладных расходов на параллелизм и виртуальную память", *Моделирование и анализ информационных систем*, **25**:2 (2018), 165–173.

DOI: 10.18255/1818-1015-2018-2-165-173

Аннотация. В данной статье представляется методология и результаты измерений и оценки накладных расходов, связанных с параллелизмом и виртуальной памятью. Для получения наиболее точных экспериментальных данных использовалась специальная методика измерений. Данная методика сфокусирована на измерениях совокупных потерь производительности, создаваемых параллелизмом, выраженным в форме легковесных потоков пользовательского режима на процессорах с архитектурой IA-32. Были получены и проанализированы данные, произведенные в средах с виртуальной памятью и без нее. Таким образом стало известно, какая потеря производительности вызывается виртуальной памятью, а также то, как она влияет на накладные расходы, связанные с параллелизмом. Эксперименты показали, что накладные расходы на параллелизм гораздо существеннее накладных расходов на виртуальную память. И тем не менее, между ними существует сложная взаимозависимость. Статья публикуется в авторской редакции.

Ключевые слова: виртуальная память, параллелизм, накладные расходы, измерения

Об авторах:
Клименков Евгений Иванович, orcid.org/0000-0001-7449-7986, магистр техн. наук, аспирант,
Белорусский государственный университет информатики и радиоэлектроники,
ул. П. Бровки, 6, г. Минск, 220013 Республика Беларусь, e-mail: klimenkov@bsuir.by