International Journal of Hybrid Information Technology
Vol.8, No.8 (2015), pp.57-68
http://dx.doi.org/10.14257/ijhit.2015.8.8.05

Virtual Memory Systems Should Use Larger Pages rather than the Traditional 4KB Pages

Pinchas Weisberg and Yair Wiseman

Computer Science Department, Bar-Ilan University, Ramat-Gan 52900, Israel
wiseman@cs.biu.ac.il

Abstract

Choosing the best page size for Virtual Memory requires considering several factors. A smaller page size reduces the amount of internal fragmentation. On the other hand, a larger page needs smaller page tables. However, this paper argues that the main reason to prefer a larger page is to increase the virtual to physical translation speed i.e. because the size of a TLB is limited, to facilitate increasing of TLB coverage we have to use larger pages

Keywords: *Memory Management, Virtual memory, Memory Pages*

1. Introduction

Over the history of memory management units (MMUs) of processors, their hardware has employed page size of 0.5KB to 8KB. E.g. VAX developed by Digital Equipment Corporation (DEC) in the mid-1970s used 512 bytes page size [1]. In the mid-1990s Digital Equipment Corporation decided to upgrade its processors and replace the old VAX by the DEC Alpha. The page size of DEC Alpha was considerably increased to 8KB [2]. Sun 1 manufactured by Sun Microsystems and launched in 1982 has a 2K bytes page size [3]. IA-32/x86 has been making use of 4KB page size since the mid-1980s [4] and even when they moved to x86-64 at 2003 the 4KB page size was remained [5].

Essentially, a 4KB page size has been employed for Virtual Memory since the 1960s. What's more, nowadays, the most widespread page size is still 4KB. Selecting a page size is actually compromising between a number of concerns and issues taken into account.

From one point of view, a larger page size will cause more fragmentation; therefore selecting a smaller page size will save some memory space. Indeed, when the physical memory is quite small, such a consideration is very significant because too much load on a small memory can cause a Thrashing [6, 7]. However, nowadays computer hardware usually has abundance of memory which is usually much more than a conventional user needs, so the trashing issue is usually less important and frequently inconsiderable.

In addition, when several processes share memory, the sharing is always of full pages [8]. Therefore, if the page size is larger, the resolution will be poorer.

On the other side, selecting a larger page will enlarge the TLB coverage and as a result will reduce the TLB misses and the necessity to read page tables in the main memory.

Over the years when many processors have used the traditional 4KB; however, the memory size has been upgraded from some hundreds of Kilobytes to several Gigabytes, therefore we can forfeit some memory space so as to obtain higher performance [9].

The most important motivation for have a preference of larger page is making the translation time of virtual addresses to physical addresses better. In a virtual memory scheme, the memory management unit (MMU) hardware translates each virtual address generated by the CPU into a physical address. The page tables of all the processes containing all the physical addresses for the possible virtual addresses are stored in the main memory. This means that each access to a memory address will be doubled, because an extra access for the translation will be needed. To facilitate a shorter virtual to physical address translation time, the most recently used addresses are stored in a Translation Lookaside Buffer (TLB). TLB is a high-speed access cache implemented by a dedicated registers.

Some processors have just one level of TLB cache; whereas some processors like Itanium 2 have two levels of TLB cache [10]. In the first level of the TLB memory, Itanium 2 has 32 entries Instruction TLB cache and 32 entries Data TLB cache. In addition, Itanium 2 has 128 entries Instruction TLB cache and 128 entries Data TLB cache in the second level of the TLB memory.

If a TLB miss occurs at the first level, the second level is accessed. The penalty for such a miss is just 2 clock cycles for a miss in the Instruction TLB and 4 clock cycles in the Data TLB; however, if s miss occurs at the second level, an access to the main memory is required because the physical address must be looked for in the original page table in the main memory and then the physical address have to be loaded into the appropriate TLB. Because an access to each of the TLBs must take only very short time, the TLBs are built with only a small number of entries.

The term "TLB coverage" refers to the total amount of memory mapped by all of the TLBs. Let us assume TLBs with 256 entries and a standard page size of 4KB. In such a case the TLB coverage is only one megabyte of memory. Because the size of the TLBs is constrained, to facilitate enlargement of TLB coverage, larger pages are required. A different way for better performance can be an improvement of the Memory Management Unit cache system [11].

However, the most common option for increasing the TLB coverage is by making use of super-pages [12, 13]. Many contemporary CPU architectures provide a support for super-pages. Such an architecture allows the Operating System to make use of a number of page sizes. Super-paging mechanism enables selecting a suitable page size for any allocation.

A small page size will be selected for a small spatial locality with the purpose of saving memory and a large page size will be selected for a large spatial locality with the purpose of enlargement of the TLB coverage. The spatial locality of an information segment can be analyzed as we suggested at [14].

E. g., very large pages are suitable for allocations of non-paged memory, such as for mapping frame buffers or for the fixed pieces of the operating system kernel; whereas small page size can be suitable for the kernel stack [15]. When super-pages are used for paging the code segment and data segment of a user process, an intermediate sized page should be selected. The average unused memory space produced by internal fragmentation with too large pages might be substantial. Writing even just one byte to such a page can be costly, because there is only one dirty bit and there is a need to update the entire page.

Most of the modern Operating systems do not employ Super-pages, even though most of the CPU architectures support Super-Pages [16]. So in point of fact only the 4KB page size has been employed for Virtual Memory systems in most of the architectures since the 1960s. Actually, nowadays, the most frequent page size is still 4KB. It should be noted however that Linux running on SGI Altix systems uses a fixed page size of 16 KB for all processes, using the 16KB super-page of SGI Altix [17]; rather than using the usual 4KB pages that supported as base pages by SGI Altix.

Over the years, the factors that have an effect on the page size have been changed. The common memory size of standard computers has been increased from some hundreds of Kilobytes to several Gigabytes. Consequently, standard TLBs cover only a small portion of a common memory of contemporary computers. In addition, access times of standard contemporary disks have not kept up with throughput augments. In last years, usual throughput has been enhanced by a factor of 100; though, access time has enhanced by just a factor of 3 [18]. Therefore, transferring of larger pages from and to a disk has become more reasonable.

2. Selecting Page Size Range Results

With the intention of assessing the best page size for virtual memory systems, several benchmarks from the SPEC suite [19] have been run. The TLB misses have been counted so as to observe the differences between the memory usages of a variety of benchmarks employing various page sizes. The most acceptable preference for a page size is when there will be a substantial reduction in TLB misses but with almost no enlargement in memory space usage.

When making use of super-paging, many operating systems (*e. g.*, HP, IRIX and Solaris) employ the Allocation method *i.e.*, the operating system kernel allocates a large enough page when the first page fault occurs. These operating systems use this method because it is simpler to allocate and map the entire superpage when the operating system accesses the page at the first time and because not enough research has been done about the possible advantages of the other complex methods [20]. In this paper this scheme for employing super-paging is implemented.

Many processors supporting super-pages have a range of page sizes. In spite of this, as mentioned above, there is no established strategy for the operating system kernel to select the most appropriate size.

There are three major different methods to make use of Super-pages

- **Allocation**

When a page fault occurs, a large page will be allocated and all its base-pages will be loaded into the memory. *I. e.*, the page is considered as a super-page in the page table without more ado. There are commercial OSes like Solaris MPSS [21] that make use of this method for super-paging.

IRIX [22] and HP-UX [23] implement multiple page sizes in such a method. At a page fault time, they allocate large enough pages. In IRIX the preferred page size is specified by the user/compiler prior to the application execution by a system call compiled into the application code. In HP-UX the page size is specified either by the user like IRIX or transparently set by the operating system according to the size of the required memory allocation.

Allocating and populating one large page at the same time is advantageous because any access to the large page from the allocation time will not incur a TLB miss and will also use only one TLB entry. In addition, transferring many base pages from the secondary storage is less efficient than transferring one large page entirely.

- **Reservation**

Unlike the allocation scheme, in reservation-based allocation, when the first access to an address in a base page occurs, the entire super-page that contains this base-page will be allocated, but only the base page will be populated. After that, only the base pages in this super-page that induce page faults will be loaded from the secondary storage to memory. When the number of the populated frames in the super-page reaches a threshold, the missing pages will be fetched from secondary storage to memory and the base pages will be promoted into one superpage.

In [24] Talluri *et al.*, suggest a reservation based scheme with a TLB that can map two fixed page sizes – 4KB and 64KB. Unlike Talluri *et al.*, at [25] Navarro suggests a reservation based scheme with multiple page sizes wherein promotion can be done incrementally.

The main benefit of reservation based scheme as compared to full allocation based scheme is that it defers loading of base pages until it will be more convinced that more base pages included in the super-page will be used. It also makes the launch time of a process shorter.

- **Relocation**

In a relocation based scheme, base pages are brought to the main memory. The memory usage is frequently monitored so as to make a decision when it is beneficial to promote several base pages to a super-page. When the system decides on promotion, a contiguous area of memory is sought and the base pages are moved into it. If not all of the base pages are resident in the main memory, the missing base pages will be brought from secondary storage.

In another paper of Talluri *et al.*, [26] they sugegst a relocation scheme wherein there are two page sizes – 4KB and 32KB. The threshold they have set for relocation is whether at least half of the base pages have been brought to the main memory. Romer *et al.*, [27] suggest a scheme for tracking potential promotions to super-pages and deciding dynamically when it is beneficial to promote the base pages to one super-page. According to their approach super-pages will be created only when it is exceedingly needed therefore internal fragmentation is minimized.

In this paper a basic super-paging scheme is assumed in which just one of two page sizes are selected, either the base page size or a super-page size. According to this scheme, the strategy of the operating system kernel is allocating a fixed large page only if the memory object is large enough and also there is enough memory space for this memory object. When running the SPEC benchmark suite we observed that dTLB misses decrease just with larger pages. This can be of help for us on the way of deciding about the large page size appropriate for substantial data segments.

3. Experiments Results

With the aim of counting the TLB misses and analyzing the memory usage of a variety of page sizes, 12 applications from the SPEC suite have been simulated. Such simulations of SPEC benchmarks in reality take a long time to come to an end, therefore we traced each benchmark for only its first 48 hours. The hardware used for these experiments was a 3.66GHz Intel(R) Xeon(TM) CPU. This hardware was dedicated to these benchmark simulations. We used "valgrind" [28] which is a suite of simulations based debugging and profiling tools for the Linux operating system. One of "valgrind"'s tools called "Lackey" was adjusted to produce a trace of memory references. The output of the adjusted tool incorporates a trace of page references for all the page size that are power of 2 multiples from 4KB up to 256KB.

The traced output includes enormous data. Therefore, if this output was saved into a file, it would swiftly enlarge to several gigabytes. To facilitate a solution for this constraint, we have used an on-the-fly simulation. The traced output of "valgrind" was redirected to another process via a pipe. The other process analyzed the data and generated the results.

This analyzing process actually simulated an LRU based TLB and counted the TLB misses for every page sizes. The simulated TLB of this process was a fully associative TLB [29]. It has been assumed that the TLB have 64 entries for instructions and 64 entries for data.

It has been also assumed that the TLB is dedicated to only the benchmark *i.e.*, it has been assumed that the operating system kernel or other processes that usually can take entries in the TLB do not take these entries. As was mentioned above, the operating system kernel can use large pages and therefore usually take a small number of entries in the TLB. For that reason, there is typically just an insignificant effect of a running operating system kernel on the TLB performance.

In the analyzing process, the pages allocated to every process have been counted. For every page size, the total sum of memory space allocations of the process during its execution has been calculated. A large volume of physical memory with no need of swapping has been presumed.

Figures 1a and 1b show the numbers of the TLB misses and the memory consumption of the crafty benchmark; whereas Figures 2a and 2b show the numbers of the TLB misses and the memory consumption of the parser benchmark.

The benchmarks crafty and parser have produced similar results – The iTLB misses dramatically decrease when using 16KB pages and are almost eliminated. Actually, these two benchmarks characterize 10 out of the 12 benchmarks that have been run in our experiments. The relative iTLB miss percentage is diminutive for almost every benchmark we ran when employing 16KB pages.

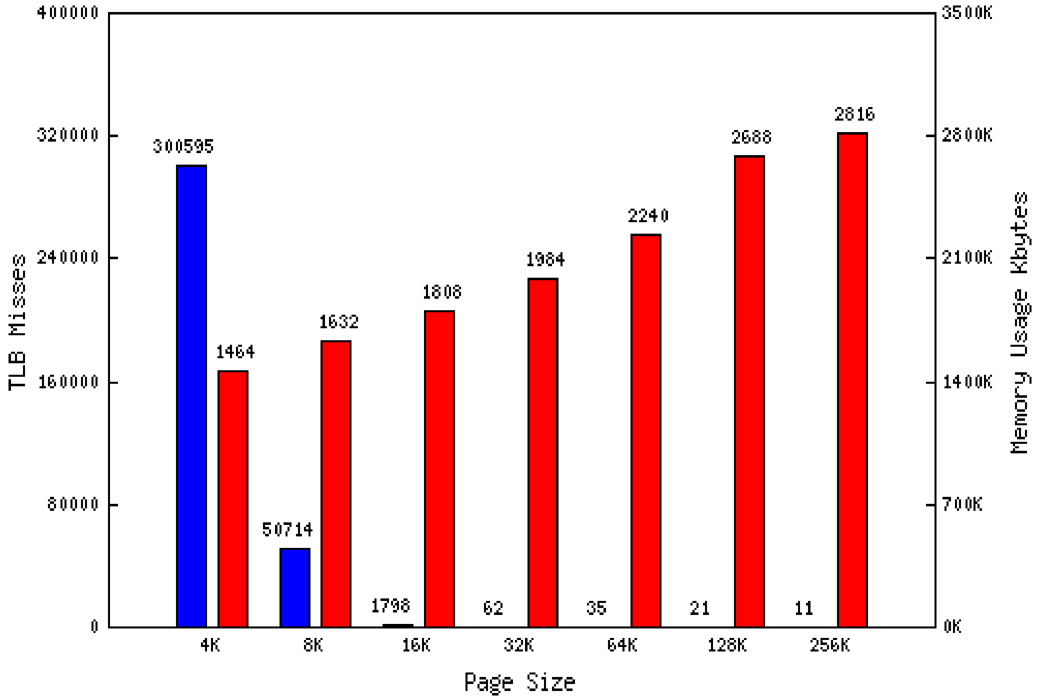

Figure 1a. Instruction TLB misses (in blue) and Memory Consumption (in red) of the Crafty Benchmark

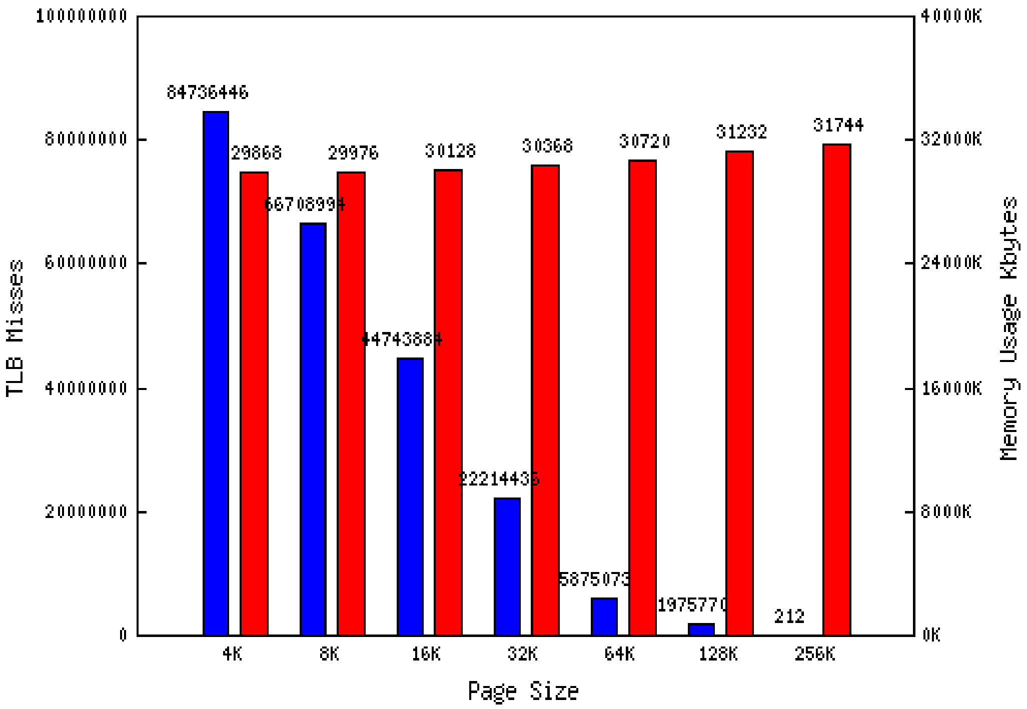

Figure 1b. Data TLB Misses (in blue) and Memory Consumption (in red) of the Crafty Benchmark

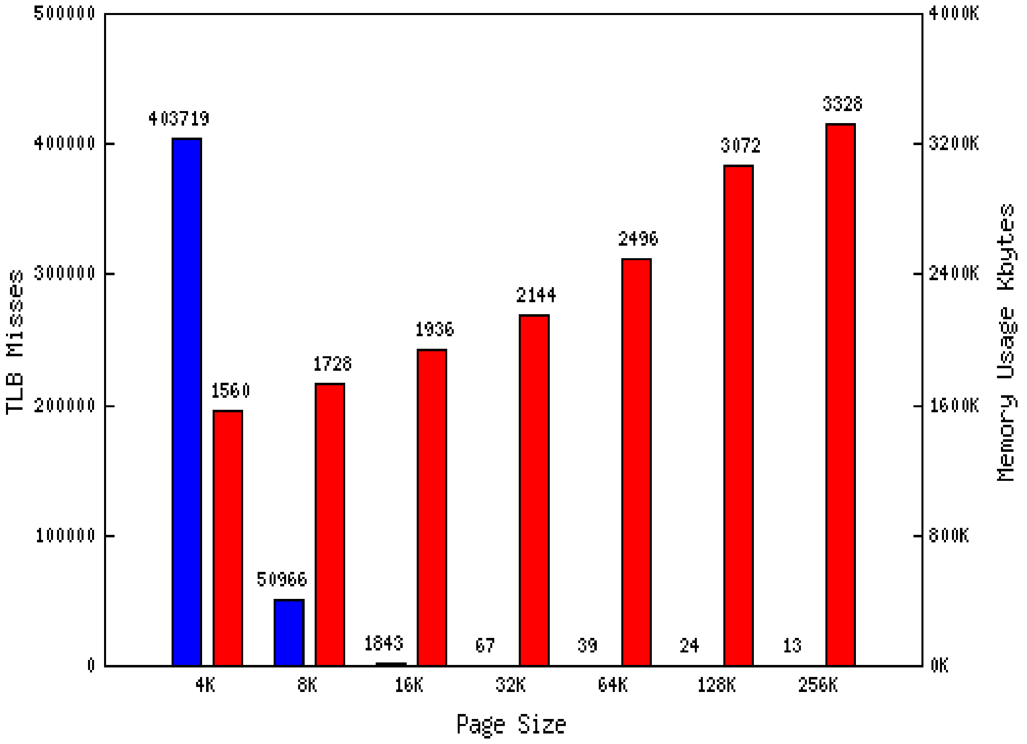

Figure 2a. Instruction TLB Misses (in blue) and Memory Consumption (in red) of the Parser Benchmark

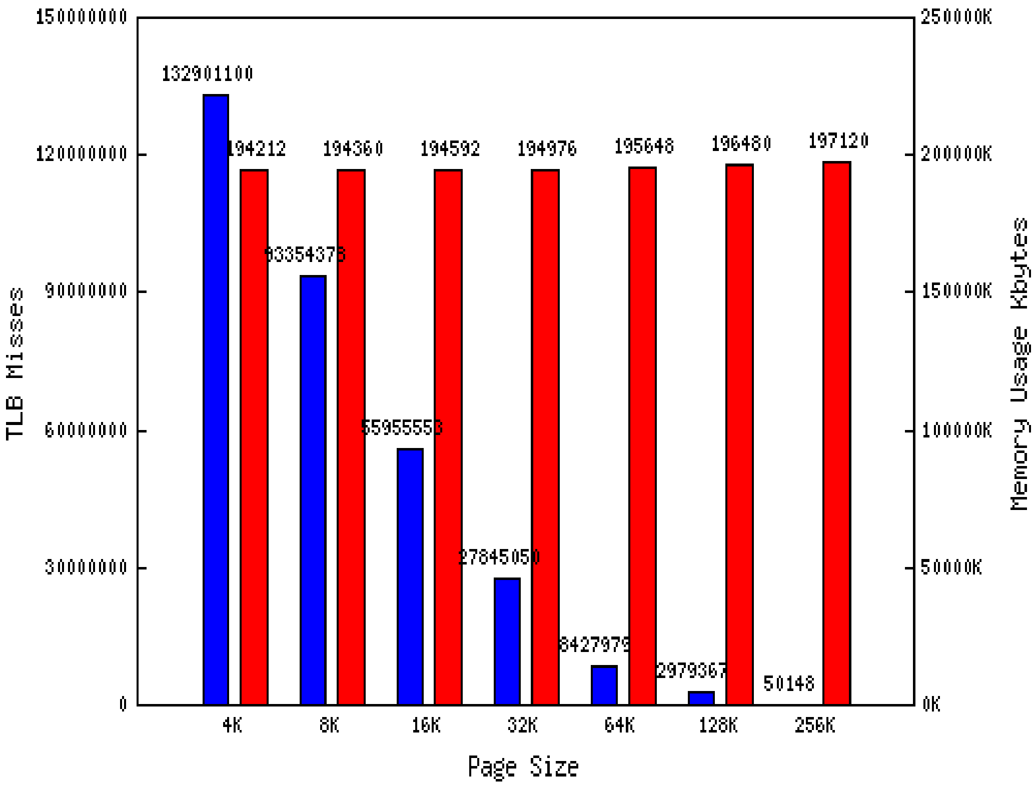

Figure 2b. Data TLB Misses (in blue) and Memory Consumption (in red) of the Parser Benchmark

It can also be noticed that the increase in memory space usage for crafty is 10.3% and for parser is only 0.69%. The additional memory utilization is without doubt worth the performance boost. Nowadays computer hardware has abundance of memory which is usually much more than a conventional user needs, so utilizing more memory for performance improvement looks like a quite good deal [30].

Concerning the data portion of crafty and parser, the dTLB misses are almost eliminated just when using 256KB pages. These are the results of almost all applications; however it should be noted that the dTLB misses of vpr (described herein below) and gcc (that we did not put a graph for it) have been almost eliminated at 32KB pages and on the contrary the dTLB misses of apsi have not been eliminated even at 256KB. Regarding memory space consumption for data, the space will be almost constant when increasing the page size.

Figures 3a and 3b show the numbers of the TLB misses and the memory consumption of the apsi benchmark; whereas Figures 4a and 4b show the numbers of the TLB misses and the memory consumption of the vpr benchmark.

The apsi and vpr benchmarks have somewhat dissimilar functioning. Actually, these benchmarks are with the poorest memory space usage results; however even when looking at the apsi and vpr results, the iTLB decreases to approximately 1/3 and the memory space usage increases by approximately 2/3 at 16KB page size. Still such results appear to be acceptable considering today’s computer hardware which usually has large quantity of memory, especially since the introducing of the 64-bit microprocessors more than two decades ago [31].

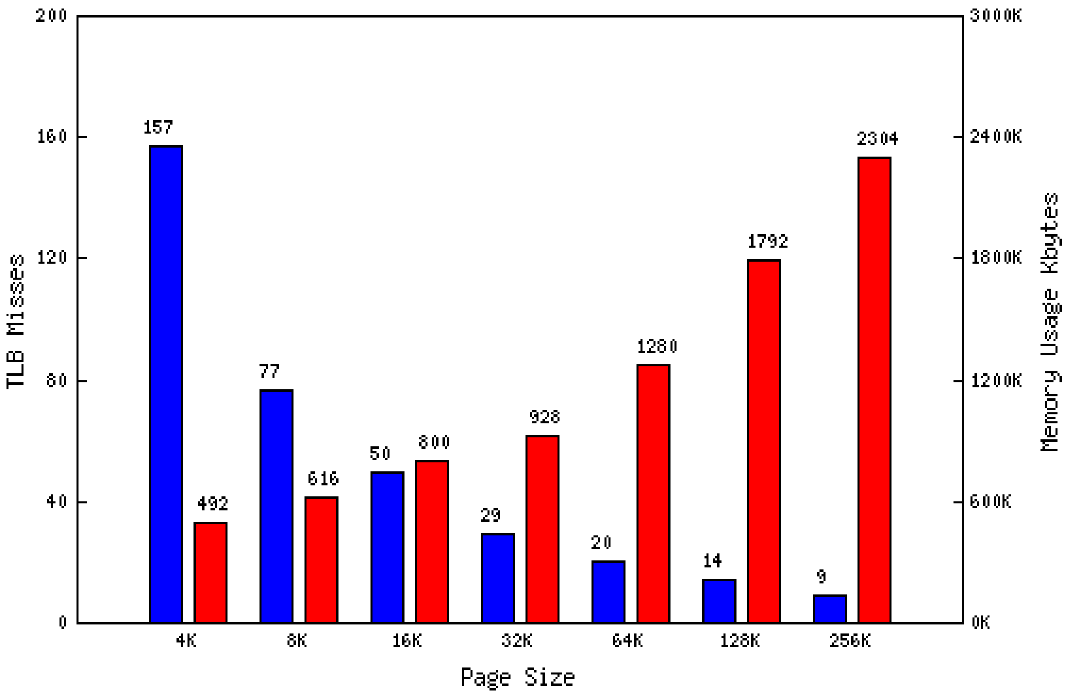

Figure 3a. Instruction TLB Misses (in blue) and Memory Consumption (in red) of the apsi Benchmark

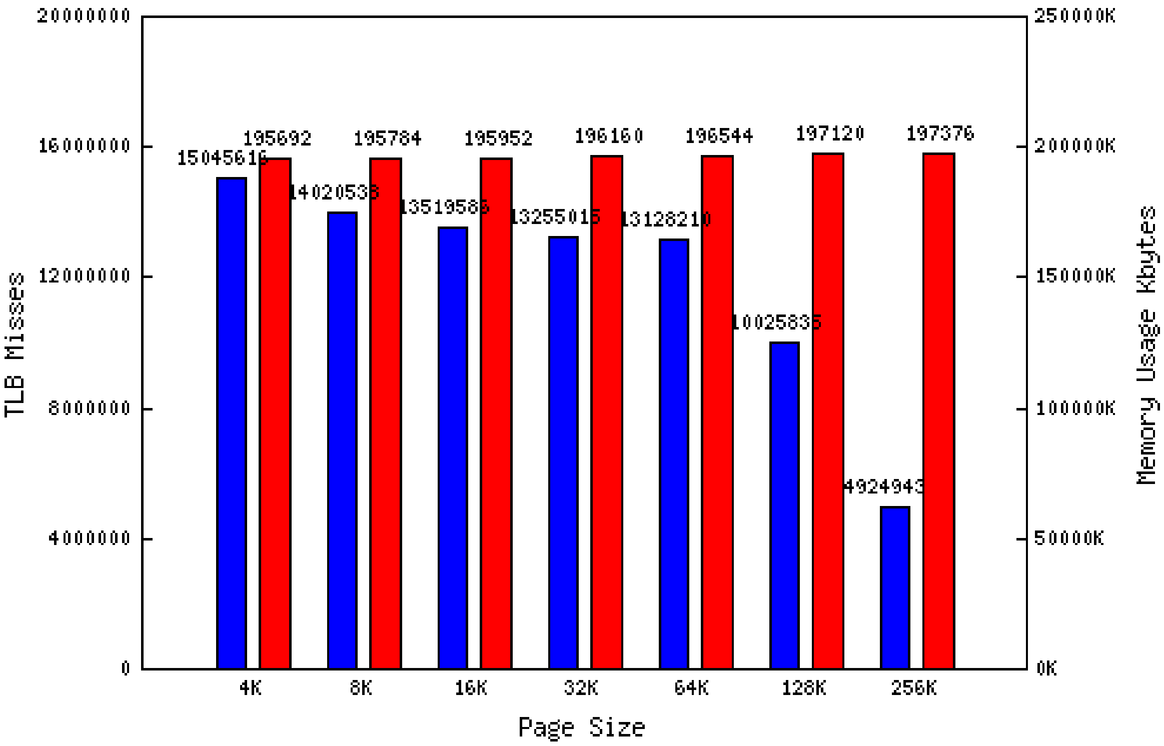

Figure 3b. Data TLB Misses (in blue) and Memory Consumption (in red) of the apsi Benchmark

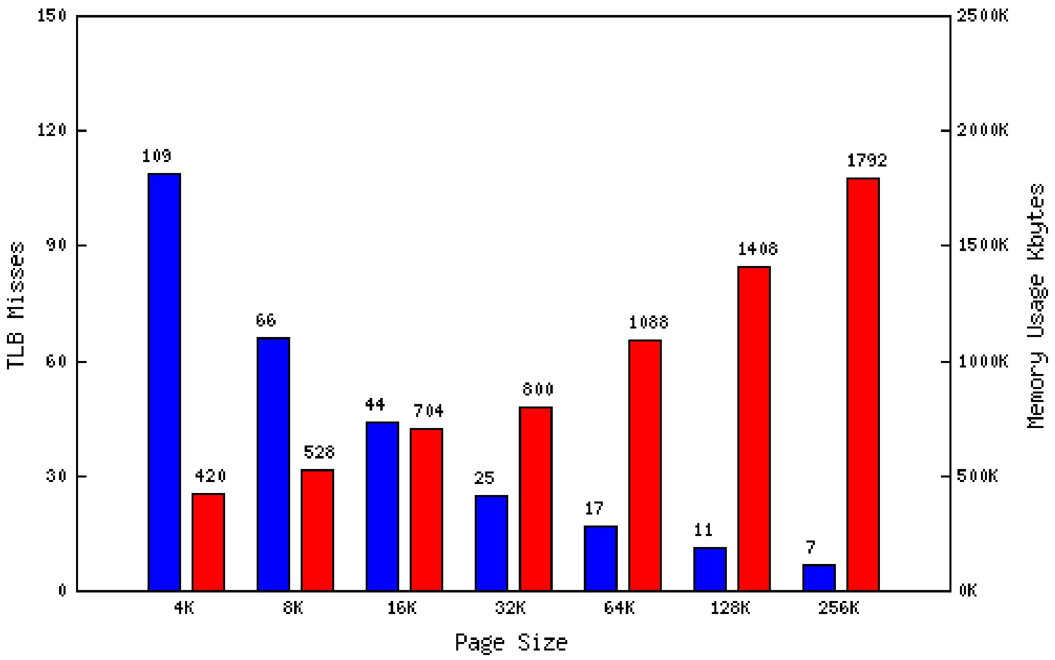

Figure 4a. Instruction TLB Misses (in blue) and Memory Consumption (in red) of the vpr Benchmark

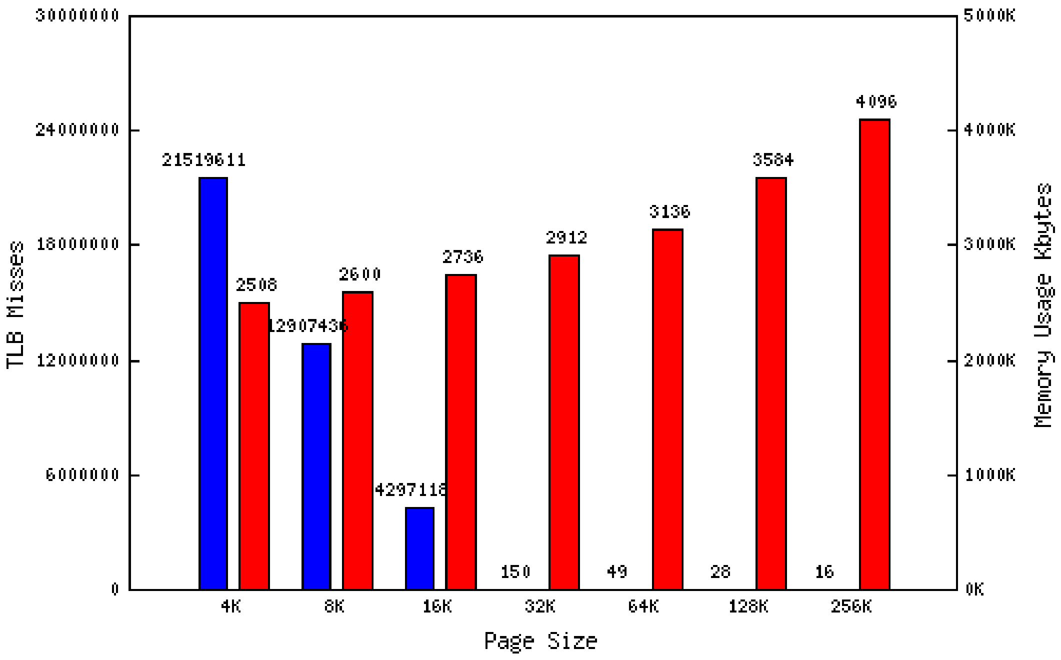

Figure 4b. Data TLB Misses (in blue) and Memory Consumption (in red) of the vpr Benchmark

4. Conclusions

It can be concluded that the traditional 4KB page size is not suitable for modern computer hardware. 16KB base page is much more suitable size for the allocation of code segments. 16KB base page almost eliminates iTLB misses for a large amount of applications without bring upon the computer hardware a high memory price tag. Allocating larger pages however, is not advisable because for nearly all applications it barely decreases iTLB misses but for several applications, it can significantly increase the required memory space.

The ending conclusion of this paper is that for large data objects, the operating system is supposed to allocate 256KB pages. On the other hand, for small data objects and for code segments, the operating system is supposed to allocate smaller pages in size of 16KB. For that reason, it is advised that new base page size of the new architectures will be 16KB.

References

[1] D. W. Clark and J. S. Emer, "Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement", ACM Transactions on Computer Systems (TOCS), Vol. 3, No. 1, pp. 31-62, New-York, **(1985)**.

[2] J. H. Edmondson, P. I. Rubinfeld, P. J. Bannon, B. J. Benschneider, D. Bernstein, R. W. Castelino, E. M. Cooper, D. E. Dever, D. R. Donchin, T. C. Fischer, A. K. Jain, S. Mehta, J. E. Meyer, R. P. Preston, V. Rajagopalan, C. Somanathan, S. A. Taylor, G. M. Wolrich, "Internal Organization of The Alpha 21164, A 300-MHz 64-Bit Quad-Issue CMOS RISC Microprocessor", Digital Technical Journal Vol. 7, No. 1, **(1995)**.

[3] G. Larry, R. Lyon, L. Delzompo and B. Callaghan, "The Open Network Computing Environment", In The Sun Technology Papers, pp. 3-12, Springer, New York, **(1990)**.

[4] E. Grochowski and K. Shoemaker, "Issues in the Implementation of the i486 Cache and Bus", In IEEE International Conference on Computer Design: VLSI in Computers and Processors, ICCD'89, pp. 193-198, **(1989)**.

[5] T. W. Barr, A. L. Cox and S. Rixner, "Translation Caching: Skip, Don't Walk (The Page Table)", In ACM SIGARCH Computer Architecture News, Vol. 38, No. 3, pp. 48-59, New York, NY, USA, **(2010)**.

[6] M. Reuven and Y. Wiseman, "Medium-Term Scheduler as a Solution for the Thrashing Effect", The Computer Journal, Oxford University Press, Swindon, United Kingdom, Vol. 49, No. 3, pp. 297-309, **(2006)**.

[7] M. Reuven and Y. Wiseman, "Reducing the Thrashing Effect Using Bin Packing" In Proceedings of Modeling, Simulation, and Optimization Conference, MSO-2005, Oranjestad, Aruba, pp. 5-10, **(2005)**.

[8] M. Geva and Y. Wiseman, "Distributed Shared Memory Integration", In Proceedings of IEEE Conference on Information Reuse and Integration (IEEE IRI-2007), Las Vegas, Nevada, pp. 146-151, **(2007)**.

[9] P. Weisberg and Y. Wiseman, "Using 4KB Page Size for Virtual Memory is Obsolete", In Proceedings of IEEE Conference on Information Reuse and Integration (IEEE IRI-2009), Las Vegas, Nevada, pp. 262-265, **(2009)**.

[10] Intel, Intel Itanium 2 Processor Reference Manual – For Software Development and Optimization, Document-No.: 251110-003, **(2004)**.

[11] B. Abhishek, "Large-Reach Memory Management Unit Caches", In Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 383-394, Davis, California, USA, **(2013)**.

[12] M. Itshak and Y. Wiseman, "AMSQM: Adaptive Multiple SuperPage Queue Management", Special issue of the International Journal of Information and Decision Sciences (IJIDS) on the best papers of IEEE Conference on Information Reuse and Integration (IEEE IRI-2008), Vol. 1(3), pp. 323-341, **(2009)**.

[13] Y. Wiseman, "ARC Based SuperPaging", Operating Systems Review, Vol. 39(2), pp. 74-78, **(2005)**.

[14] P. Weisberg and Y. Wiseman, "Efficient Memory Control for Avionics and Embedded Systems", International Journal of Embedded Systems, Inderscience Publishers, Vol. 5, No. 4, pp. 225-238, **(2013)**.

[15] Y. Wiseman, J. Isaacson and E. Lubovsky, "Eliminating the Threat of Kernel Stack Overflows", Proc. IEEE Conference on Information Reuse and Integration (IEEE IRI-2008), Las Vegas, Nevada, pp. 116-121, **(2008)**.

[16] S. Yuki, B. Gerofi, and Y. Ishikawa, "Revisiting Virtual Memory for High Performance Computing on Manycore Architectures: A Hybrid Segmentation Kernel Approach" In Proceedings of the 4th ACM International Workshop on Runtime and Operating Systems for Supercomputers, Article no. 3, **(2014)**.

[17] J. Guido, M. S. Müller, W. E. Nagel, and S. Pflüger, "Accessing Data on SGI ALTIX: An Experience With Reality" In Proceedins of 4th Workshop on Memory Performance Issues, (WMPI-2006), Austin, Texas, **(2006)**.

[18] P. Schmid, "15 Years Of Hard Drive History: Capacities Outran Performance", Tom's Hardware, **(2006)**.

[19] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions" ACM SIGARCH Computer Architecture News, Vol. 34, no. 4, pp. 1-17, **(2006)**.

[20] D. Yu, M. Zhou, B. R. Childers, D. Mossé and R. Melhem, "Supporting Superpages in Non-Contiguous Physical Memory" In proceedings of IEEE 21st International Symposium on High Performance Computer Architecture (HPCA-2015), pp. 223-234, San Francisco Bay, California, USA, **(2015)**.

[21] R. McDougall, "Supporting Multiple Page Sizes in the Solaris Operating System", Sun BluePrints On-Line, March **(2004)**.

[22] N. Ganapathy and C. Schimmel, "General Purpose Operating System Support for Multiple Page Sizes", In Proceedings of the USENIX 1998 Annual Technical Conference, USENIX Association, No. 98, pp. 91-104, New Orleans, **(1998)**.

[23] I. Subramanian, C. Mather, K. Peterson, and B. Raghunath, "Implementation of Multiple Pagesize Support in HP-UX", In Proceedings of the USENIX 1998 Annual Technical Conference, USENIX Association, New Orleans, **(1998)**.

[24] M. Talluri and M. D. Hill, "Surpassing the TLB performance of super-pages with less operating system support", In Proceddings of 6th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), San Jose, CA, USA, pages 171-182, **(1994)**.

[25] J. Navarro, S. Iyer, P. Druschel, and A. Cox, "Practical, transparent operating system support for superpages", In Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI), **(2002)**.

[26] M.Talluri, S. Kong., M. Hill and D. Patterson, "Tradeoffs in Supporting Two Page Sizes", In Proceedings of the 19th Annual International Symposium on Computer Architecture, pp. 415-424, **(1992)**.

[27] T. Romer, W. Ohlrich, A. Karlin, and B. Bershad, "Reducing TLB and memory overhead using online superpage promotion", In Proceedings of the 22nd Annual International Symposium on Computer Architecture, pages 176-187, **(1995)**.

[28] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation", ACM Sigplan notices, vol. 42, no. 6, pp. 89-100, **(2007)**.

[29] S. Masaya, "A 800 MHz single cycle access 32 entry fully associative TLB with a 240 ps access match circuit", In proceedings of 2001 IEEE Symposium on VLSI Circuits, Digest of Technical Papers, pp. 231-232, **(2001)**.

[30] P. Weisberg and Y. Wiseman, "Virtual Memory Systems Should use Larger Pages", Advanced Science and Technology Letters, **(2015)**.

[31] K. Les and N. Margulis, "Introducing the Intel i860 64-bit microprocessor", IEEE Micro, Vol. 4, pp. 15-30, **(1989)**.

Authors

Pinchas Weisberg, got an M.Sc. from Bar-Ilan University and he is now about to finish his PhD thesis in the Computer Science department of Bar-Ilan University. Mr. Weisberg's research interests include Efficient Memory Handling, Embedded Systems and Operating System Kernel Manipulation. Mr Weisberg is now with the Computer Science and the Math departments of Bar-Ilan University in Ramat-Gan, Israel.

Yair Wiseman, got a Summa Cum Laude M.Sc. and a PhD from Bar-Ilan University and completed two Post-Docs - one at the Hebrew University of Jerusalem and one in Georgia Institute of Technology. Dr. Wiseman's research interests include Process Scheduling, Hardware-Software Codesign, Memory Management, Computer Clusters, Data Compression, JPEG, Embedded Systems, Real-Time Systems, Operating Systems, Computational Transportation Science and Intelligent Transportation Systems. Dr. Wiseman is on the editorial board of several journals, a member of dozens of conference committees and a reviewer of many scholarly journals. Dr. Wiseman authored two books as well. In addition, Dr. Wiseman has been teaching in many institutes including Bar-Ilan University, The Hebrew University of Jerusalem, Israel Aircraft

Industry, Holon Institute of Technology and Jerusalem College of Technology. Dr. Wiseman has been supervising many graduate students and an interesting point is that Albert Einstein is Dr. Wiseman's academic great-great-grandfather (i.e. the advisor of the advisor of the advisor of Dr. Wiseman's advisor). Dr. Wiseman has collaborated with other partners and received research grants to run an active laboratory from inter alia Sun Microsystems, Intel, Polak Foundation and the Open University. Dr. Wiseman is an international expert who has reviewed and evaluated several large projects of the European Union, Israel Science Foundation, MB Logic and more. Dr. Wiseman's papers have been published in many venues around the world.