The Effect of Page Size and TLB Entries on Application Performance

Neil Stringfellow
CSCS – Swiss National Supercomputing Centre

June 5, 2006

Abstract

The AMD Opteron processor allows for two page sizes to be selected for memory allocation, with a small page size of 4 Kilobytes being used for most standard Linux systems, and a larger 2 Megabyte page size which was selected for the Catamount lightweight kernel. Although the larger page size appears more attractive for HPC applications running under a lightweight kernel, the increase in page size comes at a cost in that there are very few entries in the TLB available to store references to these pages. This paper reports on work which highlighted problems with the use of the large page size and small number of TLB entries, and shows some of the performance improvements which became possible with the introduction of a small page option to the `yod` job launcher.

Keywords: Cray XT3, AMD Opteron, Page Size, TLB, Translation Lookaside Buffer, Catamount, `yod`

1 Introduction

In 2004 CSCS and the Paul Scherrer Institute put out a tender for a high performance computer system, and in 2005 this resulted in the purchase of a 12 cabinet Cray XT3 system with 1100 compute processors. The procurement used 4 groups of benchmarks to evaluate the bids, and the bidders were asked to run the benchmarks and extrapolate to the full system. Cray ran the benchmarks on an Opteron cluster using standard Linux and extrapolated to an XT3 running catamount.

One major difference between Linux and Catamount is the memory page size used by default, and the Opteron's design means that there are fewer TLB entries for the larger pages which Catamount used. Shortly after the contract for the machine was signed, the local Cray benchmarking expert discovered that the extrapolations used in committing benchmark results were not valid for the large page size and that the committed values for the benchmarks would be difficult or impossible to meet with the default setup.

Cray eventually delivered a small page option to the yod job launcher, and this not only allowed Cray to pass the acceptance on the failing codes, but allowed one of the failing HPC Challenge results to be passed which saved Cray some money, and subsequently proved to be of huge benefit to a large number of CSCS applications.

2 A Brief Review of Virtual Memory

On a typical modern operating system, multiple programs have the same type of memory mapping and they would want to share the same addresses in memory, and in order to avoid this conflict programs operate in a virtual memory space. Processes require sections of memory in order to execute, and

these are often named to represent the type of usage which can be expected of the memory such as a text segment for executable instructions and heap and stack sections which can be dynamically assigned to the process. For example a process might have the addresses given to the various segments at compile time and run time for a theoretical 16-bit addressable processor as in table 1.

Segment	Address (Hex)
Text	0x0000 - 0x1FFF
Text	0x2000 - 0x2FFF
Text	0x3000 - 0xEFFF
Text	0xF000 - 0xFFFF

Table 1: An example of segments and addresses for a 16-bit processor

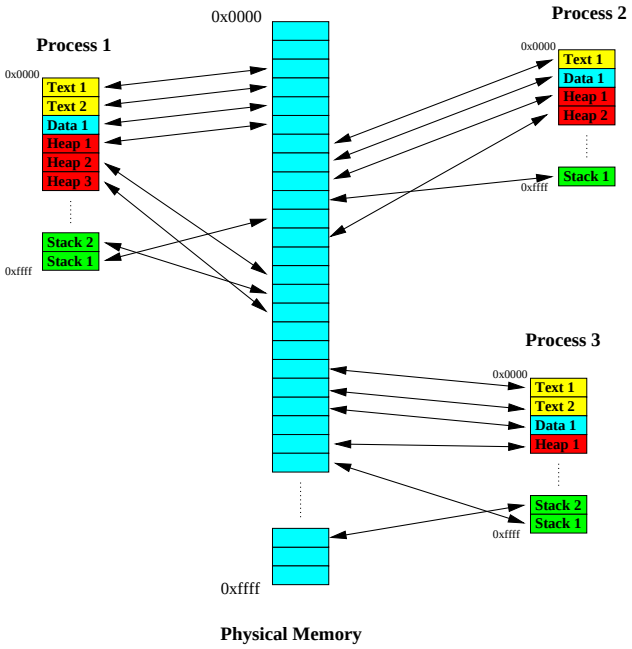

Figure 1: Virtual to physical memory mapping

All processes on a system will have similar mappings for these sections, and would want to use the same addresses which would lead to overlapping memory between processes and only one process would be able to run on such a system.

An operating system provides memory allocation and assignment to a process and when carrying out this allocation the operating system gives out physical memory blocks to a process but it also provides a virtual to physical memory mapping so that the process "thinks" that it is addressing different memory locations to those which it really addresses. With the virtual to physical memory mapping, the process can execute based on memory addresses which may be hard coded or given to it by the operating system and then the CPU maps these to the correct physical addresses so that the program can operate successfully.

An example of a physical to virtual memory mapping is shown in figure 1 where three processes have their various memory segments mapped from their virtual addresses to locations in physical memory. The same concept as in this example occurs on modern operating systems and processors where virtual memory is used, and therefore modern processors have built-in hardware support for virtual to physical mapping.

The CPU accesses the correct physical location by carrying out an address conversion by looking into a translation lookaside buffer (TLB) on the processor and the TLB itself will consist of pairs of addresses which are the base address of a virtual page and the corresponding base address of a physical page. The TLB entries, together with the TLB search and translation algorithms and other associated logic, take up real estate on the processor and therefore manufacturers try to keep the number of TLB entries to the minimum which would be required for good operation.

The memory is split into pages so that not every memory access needs a physical to virtual mapping, as it is sufficient to give the mapping for a contiguous block of whatever page size is used. The size of this memory page varies between different processors and many modern processors have hardware support for different page sizes to accommodate different workloads. One important feature of the distribution by memory pages is that each section of a program needs at least one page and therefore an operating system which has many concurrent processors benefits from using a relatively small page size since this minimises memory allocation to those processes which would otherwise have a small mem-

ory footprint.

3 Some Background Architecture of the AMD Opteron

The AMD Opteron is in the x86 family of processors which can trace its history back for over 20 years, and while the architecture of the chip is radically different to the early Intel 8086, the instruction set has undergone only gradual changes allowing compatibility from one generation to the next.

The Cache hierarchy on the current single core Opteron is 64 kilobytes of primary instruction cache, 64 kilobytes of primary data cache and 1 Megabyte of unified secondary cache.

The main use for these processors would now be expected to be a Windows/Linux base where there are a large number of processes which all require their own memory, be it multiple different user applications, kernel processes and daemons. Each of these processes is given its own area of memory, and therefore its own memory pages, and therefore these operating systems with large numbers of processes prefer smaller pages to minimise memory consumption.

The AMD Opteron has a 4 Kilobyte page size which can be used to allow large numbers of processes to coexist within the memory constraints of a modern computer system, and for this page size the TLB has 64 entries in the primary cache and 1024 entries in the secondary cache.

The AMD Opteron also has a 2 Megabyte page size, which may be less suitable for standard operating systems with large numbers of processes, since even the smallest process would require a large memory footprint, but might find a use on HPC systems. For this 2 Megabyte page size, the TLB has 8 entries in the primary cache , but *no* entries in the secondary cache so that there are only 8 TLB entries overall for this page size.

One other point to note about the difference in page size, is that for the small 4 kilobyte pages the total memory footprint which can be covered from all TLB entries is about 4 Megabytes, whereas the memory footprint covered by the larger 2 Megabyte pages is 16 Megabytes, and for some low memory consumption applications this could also prove to be significant.

4 Design Choice for Red Storm

In order to limit the interference from the operating system on a machine with large numbers of processors, the Red Storm system developed at Sandia uses the catamount kernel on the compute nodes. Catamount is very lightweight with very few "processes" operating on a processor and with this model there are few processes requiring memory and consequently not much memory is consumed by using 2 Megabyte pages as the default, therefore for Red Storm the choice was made to use the 2 Megabyte pages.

Whilst the intention of this choice may well have been to minimise TLB misses by using large pages, with only 8 TLB entries for 2 Megabyte pages, many codes could be expected to *increase* the number of TLB misses within an application and this was certainly the experience at CSCS.

5 Poor Code Performance with Default Choice

The benchmarks for the CSCS procurement which resulted in the purchase of the Cray XT3 system were IOZone, HPC Challenge and two user codes, one being a molecular dynamics code and one a code based on the Trilinos library. For the procurement, Cray used an Opteron cluster and extrapolated to the Cray XT3 based on the cluster results and the expected performance of the interconnect combined with a low OS jitter.

Shortly after delivery, most of the IOZone benchmarks (using an unstable Lustre) and HPC Challenge benchmarks (apart from the latency tests) were meeting expectations, however at this time the user benchmarks were all failing to meet expectations. Analysis by the Cray benchmarker showed that the poor performance was associated by a very high TLB miss rate, and a brief examination of one

of the codes showed that this was due to large memory strides in a solver, whereas for a molecular dynamics code the need to compare atoms which were distributed in a pseudo-random fashion in memory also produced high numbers of TLB misses.

Analysis was carried out by CSCS' computational chemistry support specialist of a set of molecular dynamics codes which were not in the procurement benchmarks and these showed similar levels of TLB misses to those which had been seen already, and at this stage it was still unclear what proportion of CSCS' entire application portfolio would be affected by this problem.

6 A New Option to the `yod` Job Launcher

Initially, Cray tried to get around the problem with the benchmarks by rewriting some codes to avoid the large number of TLB misses, and while this was probably a good exercise it was invalid for the purposes of passing the procurement as the request for proposal document had said that code changes would not be allowed - a special dispensation was made to rewrite IOZone as an MPI code.

It was then clear that the only way to move forward was to provide an option to the `yod` job launcher to allow codes to be run with the smaller 4 kilobyte pages as this would then be a much closer match to the type of machine for which the extrapolations had been made. Furthermore, at this time a significant number of CSCS applications were shown to be having very high values for TLB misses, particularly in the field of molecular dynamics simulations.

After significant pressure, and the realisation that the acceptance of the Cray XT3 at CSCS would not be possible without this option, Cray delivered a version of `yod` job launcher with the small page option in October 2005 and immediately improvements were seen in several of the acceptance codes as well as some of the scientific applications which would be used in production.

7 Improvements Seen at CSCS' Machine Acceptance

With the introduction of the small page option to the `yod` job launcher, Cray revisited the performance benchmark figures which had been committed during the procurement of the XT3, and they were able to pass all but 6 of the benchmarks where previously they had missed more than the 8 targets which had been allowed. Each missed target had a penalty attached, but the machine was now able to begin the acceptance procedure.

The benchmarks for the acceptance were split into the HPC Challenge suite of benchmarks which were all able to be passed apart from the two latency tests, the IO-Zone benchmark to test the parallel file system capabilities and a set of user benchmarks which were the ones which had failed in several places before the small page option. After the small page option had been introduced, the number of user benchmarks which failed had been reduced to 4.

To see the effect of the small pages option on a number of other codes, the local Cray benchmarker also ran the HPC Challenge suite with the small page option, and while some tests reduced in performance such as the random access GUPs benchmark, and High Performance Linpack, there was a surprising reduction in the latency such that the XT3 was now able to pass the random ring latency test within tolerance. With this result, the number of missed targets was reduced to 5 and Cray therefore saved money by having a reduced penalty from missed targets.

8 Performance Data

To highlight the wide usability of this small pages option, some examples are taken from a variety of applications which form the day-to-day workload on the Cray XT3 at CSCS or which have been tested by external benchmarkers on the machine.

Firstly, revisiting the HPC Challenge benchmark shows that in general the single node embarrassingly parallel benchmarks performed better with large pages whereas the global benchmarks normally performed better with the small pages option. The

actual figures are shown in table 2 with some large differences on a single node, but as the XT3 is not expected to generally run single processor jobs, the global benchmarks show that even for these relatively simple kernels the small page option can offer benefits.

Benchmark	Small page change
Stream	1-2% worse
Single node Random Access	over 40% worse
DGEMM	1% worse
Single node FFT	10% worse
HPL	1% worse
Ptrans	3% better
Random Access	5% better
Latency	4-5% better
Bandwidth	no change
Global FFT	2% better

Table 2: HPC Challenge improvements using small pages

An important application at CSCS is the numerical weather prediction code LM which is used by MeteoSwiss to produce the twice daily weather forecasts for Switzerland. This application has different characteristics depending upon which part of the code is being executed with some parts carrying out advection and diffusion and others calculating precipitation, and each of these has different characteristic memory access patterns so that an a-priori estimate of the effect of small pages is not an easy thing to produce. In fact the code benefits from the small page option with a reduced run-time of around 10%, and this has shown to be the case from a small 28 processor run using the LM_RAPS public version of the code right up to 400 processor runs of the next generation operational model.

A turbulence code from the United Kingdom Turbulence Consortium was run on the Cray XT3 at CSCS and initial results showed that although the code was able to scale to hundreds of processors with the given input configuration, the timings were worse than on other machines. By running with the small page option, a significant improvement in timings was achieved and the timings for this PCHAN code are shown in table 3 where a percentage improvement figure is calculated as $100.0(T_{\text{large}} - T_{\text{small}})/T_{\text{small}}$ as this determines the percentage increase in the amount of work which could be achieved with the same amount of computational resource.

Processors	Large	Small	% Improvement
24	3117	2287	36
32	2395	1765	36
48	1551	1211	28
64	1205	905	33
96	831	613	36
128	630	482	31
192	450	345	30
256	370	297	25
384	240	208	15

Table 3: Timings for PCHAN code for large and small pages

A set of experimental ScaLAPACK routines were also run on the smaller system at CSCS with both small and large pages, and in this case a performance degradation of between 1 and 6 per cent was seen using the small page option so that the use of small pages is not appropriate in all cases.

A significant amount of the computational resource at CSCS is used for classical molecular dynamics codes, with typically 15 to 20 per cent of the monthly usage on the Cray XT3 being made up from the codes NAMD, Amber, Orac, DL_POLY and MoldyPSI. For these codes significant performance gains can be delivered by using the small pages option, with standard tests for Orac showing up to 20 per cent improvement, and similar numbers having been produced for Amber and DL_POLY. For NAMD, the improvements can be even higher, as is easily demonstrated by running the standard Apoa1 benchmark with results shown in table 4.

Processors	Large	Small	% Improvement
4	10.35	7.78	33
8	5.16	3.94	31
16	2.63	2.07	27
32	1.35	1.05	29
64	0.69	0.55	25
128	0.35	0.29	21
256	0.20	0.17	17

Table 4: Days per Nanosecond for Apoa1 benchmark of NAMD

Most molecular dynamics runs at CSCS are now run with the small page option, but there are still around 30% of runs which are carried out without specifying this option on the `yod` line, and therefore there is still some wastage of resource occurring on the system.

9 Conclusions

The original committed figures for the CSCS performance benchmark suite could not be achieved with the default page size, although the HPC Challenge suite would not have shown up any issue which would have later been seen by user applications.

Not only the acceptance benchmarks, but also key CSCS user codes, particularly in the field of classical molecular dynamics, were performing poorly with the default large page option. Particularly in the case of NAMD the poor performance was also related to the Catamount default `malloc` library, and the impact of this on a number of applications which carry out frequent memory allocations is something to be further investigated.

With the changes to Catamount and the `-small_pages` option to the `yod` job launcher, the acceptance tests could be passed and also as an added bonus there was an unexpected reduction in latency on the HPC Challenge suite.

The small page option has been of benefit to many applications on the Cray XT3 and new codes are being ported which may take advantage of this option. It is sometimes difficult to get users to include the `-small_pages` option in their submission line and a site-wide default would be preferable.

The Cray XT3 at CSCS is due to be expanded to 18 cabinets during the summer of 2006, and if the usage of the machine continues as it is at present with around 20 per cent usage of molecular dynamics applications, and if the average improvement on these codes of using the small page option is around 20 to 30 per cent, then the effect on the research which can be carried out on the system will have been equivalent to having been given an extra cabinet just from the use of this simple optimisation flag.

10 Recommendations

It follows from this analysis that there should be some recommendations made to Cray to allow a site or user configurable default page size to be set, and consequently there would need to be a `-large_pages` option to `yod` to allow the current behaviour where a site sets small pages as the default.

There may need to be some work done by Cray's Scientific Libraries group in order to test whether different numerical libraries need to be delivered based on which page size an application intends to use, and this might particularly apply to the GOTO BLAS library.

All of this discussion would become unnecessary if AMD provided more TLB entries for large pages on the Opteron, and in particular if there were some entries provide in the L2 cache as this is the main division between large and small pages at the moment. It should be remembered that this is not an issue of small versus large pages, but an issue of the number of TLB entries available when using the large page support provided on the AMD Opteron processor.

About the Author

Neil D. Stringfellow studied Mathematics at undergraduate and master's level before attending Cranfield University where he obtaining a PhD also in Mathematics. After completing his post-doctoral studies he joined the staff of the University of Manchester where he provided application and optimisation support to scientific researchers who used CSAR, one of the United Kingdom's national supercomputing services. Neil joined the staff of CSCS in 2004 and since then he has been providing support to specific groups from CSCS' user community and he has also been involved in the introduction and development of the Cray XT3 supercomputer.

Acknowledgements

The original work to discover the high numbers of TLB misses in the CSCS' user benchmarks and the

realisation that a small page option to `yod` was required was carried out by Roberto Ansaloni from Cray. The subsequent effort to have this option included for both the acceptance requirements and for the improvements it would provide to CSCS' applications was driven by Mario Mattia and Roberto Ansaloni from Cray.

Claudio Redaelli carried out early work on CSCS' chemistry applications to determine the high number of TLB misses particularly for classical molecular dynamics codes and stressed the importance of this new `yod` option to CSCS' user base.

The author wishes to express his appreciation to Mike Ashworth of CCLRC Daresbury, Kevin Roy and Craig Lucas of the university of Manchester and Maria Grazia Giuffreda for providing benchmark results on the use of small pages.