Access Delays Related to the Main Memory Hierarchy on SGI Origin2000

S. Seidl

Center for High Performance Computing (ZHR)
Dresden University of Technology
D-01062 Dresden, Germany
E-mail: seidl@zhr.tu-dresden.de

Abstract

Traditionally, performance characteristics of machines have been done by comparisons of the run time behavior of well-known applications. An alternative approach taken here consists in creating artificial kernel applications whose only task it is to determine one particular machine parameter in isolation. Findings based on this method, at first glance, may not appear as practical as those based on the former one, but are actually more fundamental and, therefore, more predictive. In this paper, memory delay times were investigated to determine the performance degradations caused by access to far-off memory sections. The elementary model chosen here assumes that the entire memory is divided into sections with different access levels depending on the respective processor's location. The properties of the different levels can be described sufficiently by two parameters, i.e. first, the size of the corresponding memory section, and second, the time that elapses before data is loaded from that section. This model makes it possible to deal with expected values for access times in dependence of the size of the accessed area. A special program associated with this problem determines a finite set of expected values, integrating hardware-produced probability density functions via Monte-Carlo integration. This set of values can also be derived theoretically from the same abscissae using hardware topology information, but allowing for free level-specific delay times. To obtain the free parameters, the theoretically gained set is best fitted to the experimental one. Theory and measurement supply impressive results. Agreement and reproducibility are excellent, at least for the Origin2000.

1 Introduction

The base of the following study was the Silicon Graphics Origin2000 system installed at Dresden University of Technology (TU Dresden), which has been equipped with 54 R10000 processors with a 4 MB second level cache (SL cache) each, 18 GB main memory in total, 150 GB hard disks, and InfiniteReality2 graphics. The well known overall concept underlying this machine [2] is promising in many ways; however, one of its prerequisites would be fast accessibility of the entire main memory, which although locally distributed, can be globally addressed. The main point of the present paper is the temporal quantification of accesses to data that may be located on different levels of the memory hierarchy.

The reference to memory delays in the title draws attention to the difference between actual *delays* and *bandwidths*. This problem is disposed of by relating all the times to the respective hardware-based elementary operations; in concrete terms, this means that the duration of the operations that the main memory is involved in are based on the SL cache line size, thus 128 Bytes. Times for accesses to the SL cache itself, however, refer to 32 Bytes, i.e. to the size of an on-chip cache line.

The basic objective underlying the program developed for the required measurements is the ascertainment of precise expected values of read/write accesses to certain amounts of data with the best possible predictability of probability of location. Thus, the exactness of data required for the ensuing approximation task presupposes an absolutely relaxed machine. The approximation per se resembles a typical deconvolution problem with a spectrometer function naturally not all too well known in detail. Notwithstanding these premises, seven technical parameters have been established for the Origin2000 from the software point of view, thus calling for a discussion.

2 On the Measuring Program

The program used for measuring was written in C and was normally compiled by means of the GNU C Compiler 2.7.2.1 (GCC). The GCC, which had already been ported onto the Origin2000 by the author much earlier, generates 64 bit MIPS IV instructions for the R8000 processor so that the entire main memory can be accessed. This compiler was chosen because it outputs a very simple code for the important inner loop. In order to study the impact of new, typically generated R10000 prefetch instructions on the measurements results, a comparison is planned with the MIPS IV compiler, which is part of the operating system.

The program is designed to run on different, yet well fixed node boards from which so-called pathological memory accesses are carried out. Reading and writing is done in such a way that, e.g. when pulling in a cache line, the whole spectrum of possible activities gets run. Accesses to remote caches do not happen so that no numerical values were ascertained for them.

The code freed of most inessentials is shown in (1). In the first place, the program reads quantity w of a designated working region called *window* in the following way. The values used for w here are 1024, 2048, and 4096 Bytes. Subsequently, an outer loop is started, which initially defines the size a of an accessible region that is called *area* here. a can then take on values from 16 KB through 8 GB, which generally cannot be stored in the home memory of a node board. It is well known that *malloc*() returns an address in the case of lazy memory management, but that essentially nothing happens yet. This is also true of the Origin2000. The pages are not associated with the running process until they are accessed, which happens during the initialization phase while units are consecutively being written. In this process the operating system distributes physical pages according to hierarchy, i.e. the first pages are always located in the home memory, the following ones on the node board of the same corner, etc. This is important since the page migration facility does not work yet on the operating system release IRIX64 6.4.1.

```
#include "header.h"
void main (void)
{
  unsigned long w, i, a, j, jj, off;
  double *p, t0, t1, t;
  (void) my_input (&w);
  for (i = 4; i < 67; i++) {
    a = (unsigned long) (4096.0 * pow (2.0, (double) i / 2.0) + 1.5);
    a = a - a % w;
    p = malloc (a);
    for (j = 0; j < a / sizeof (double); j++)
      *(p + j) = 1.0;
    t0 = my_get_time_user_plus_sys ();
    for (j = 0; j < 10240000 / w; j++) {
      for (jj = 0; jj < 1000; jj++) {
        off = my_irandom () % (a / w) * w / sizeof (double);
        (void) my_mem_acc (w / sizeof (double), p + off);
        }
      }
    (void) free (p);
    t1 = my_get_time_user_plus_sys ();
    t = (t1 - t0) * (double) w / 1024.0e+7;
    (void) printf ("w = %lu, a = %lu, t = %e\n", w, a, t);
    }
  (void) exit (0);
  }
```

(1)

After initialization a defined number of randomly selected *windows* is read and promptly written by way of sign change. These *windows* are aligned and therefore do not overlap. Practically every one of them is dragged into the on-chip cache as soon as possible and remains there until it gets pushed out at one point. As driving instructions sign changes are sufficiently fast so that the instruction queue is soon full, containing outstanding loads. For relaxation, *my_mem_acc()* executes some further operations which, however, only require communication with the on-chip cache.

On the Origin2000 of the TU Dresden, results were obtained as essentially depicted in figure 1.

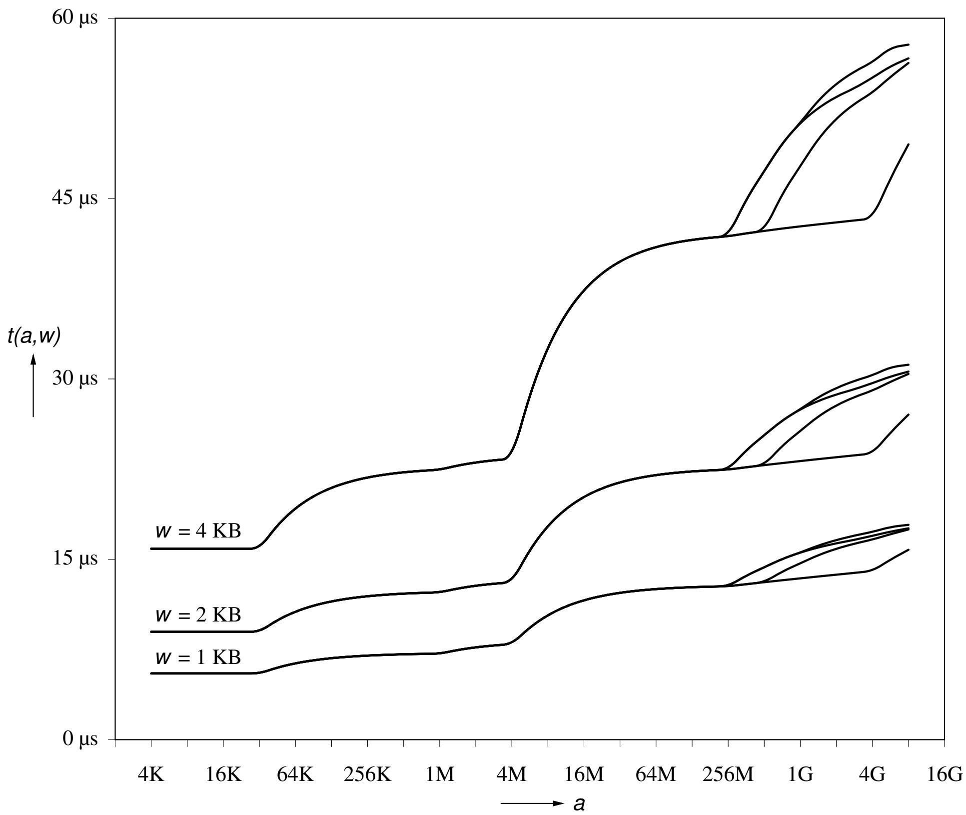

Figure 1: Execution time per *window* $t(a, w)$ depending on the size of the accessed *area* and and the *window* size. The position of the active processor may be interpreted as an additional parameter.

Up to 256 MB, the curves correspond to those of typical workstations. On-chip cache, SL cache and home memory can be clearly identified. The new fact is that the curves branch off in a variety of ways for very large *areas*, typically, at abscissae where the memory size of an hierarchy level is reached. As soon as the home memory is exhausted, additional hardware is made use of, which in turn makes for extra costs. The hub is occupied with something else, the data are run through a standard router, and on the other side, there is another hub. In summary, one can locate five hierarchy levels, two more than on normal machines. Now the question is how to infer the performance of the respective hardware components from the present results.

3 Modelling

The modelling of the main memory accesses of the measuring program takes account of three different factors. The most important factor here is the topology, which determines what hardware components the data have to be pulled through. A secondary effect which has to be quantified can be explained by the fact that somewhat poorly chosen standard page size default shows an effect starting from 1 MB, i.e. still within the SL cache. Finally, there is a vague supposition that starting from about 256 MB some limit is reached which might be linked to page table size. We will begin with considerations concerning topology.

3.1 Topology

Modelling the topology is done in analogy to experiment, i.e. based on a system with 32 regularly set up processors, of which at any one time only one is active and accesses most parts of the memory. Thus, real blocking effects only play a minor role. Furthermore, accesses to remote caches are improbable. Besides, figure 1 expresses that the measuring program cannot allow for any predictions about access times to the on-chip cache. Losses through loads from the on-chip cache are added on to program overhead. Based on these considerations, a 7 level hierarchy is established so that for the time that the measuring program needs to process an assumed, strictly localizable *window* of the size w, the following can be set up.

$$t_{top}(p, w, s_1 ... s_7) = \begin{cases} t_1(w) & \text{for} \quad 0 < p \leq \sum_{i=1}^{1} s_i \\ t_2(w) & \text{for} \quad \sum_{i=1}^{1} s_i < p \leq \sum_{i=1}^{2} s_i \\ t_3(w) & \text{for} \quad \sum_{i=1}^{2} s_i < p \leq \sum_{i=1}^{3} s_i \\ t_4(w) & \text{for} \quad \sum_{i=1}^{3} s_i < p \leq \sum_{i=1}^{4} s_i \\ t_5(w) & \text{for} \quad \sum_{i=1}^{4} s_i < p \leq \sum_{i=1}^{5} s_i \\ t_6(w) & \text{for} \quad \sum_{i=1}^{5} s_i < p \leq \sum_{i=1}^{6} s_i \\ t_7(w) & \text{for} \quad \sum_{i=1}^{6} s_i < p \leq \sum_{i=1}^{7} s_i \end{cases} \tag{2}$$

Here $t_{top}(p, w, s_1 ... s_7)$ represents a fictitious, growing step function, which for the time being presumes that the data are located in such an ordered way in the memory hierarchy that the lowest addresses p correspond to the highest, i.e. fastest levels, and vice versa. In a certain sense, p can thus be conceived of as a base address to w. The s_i are the memories physically present in each individual level. Here s_1 must always be equal to 32 KB, i.e. s_1 describes the size of the on-chip data cache. Evidently, the hierarchy level *in-register* does not exist here. s_2 is equally fixed, amounts to 4 MB and is a characteristic of the SL cache. All other s_i depend on how much memory is mounted on each node board, and therefore vary across measurements. The functions $t_i(w)$ contained in (2) look like this:

$$
\begin{aligned}
t_1(w) &= t_o(w) \\
t_2(w) &= t_o(w) + t_s(w) \\
t_3(w) &= t_o(w) + t_s(w) + t_{h_h}(w) + t_m(w) \\
t_4(w) &= t_o(w) + t_s(w) + t_{h_r}(w) + t_r(w) + t_m(w) \\
t_5(w) &= t_o(w) + t_s(w) + t_{h_r}(w) + 2\,t_r(w) + t_m(w) \\
t_6(w) &= t_o(w) + t_s(w) + t_{h_r}(w) + 3\,t_r(w) + t_m(w) \\
t_7(w) &= t_o(w) + t_s(w) + t_{h_r}(w) + 4\,t_r(w) + t_m(w)
\end{aligned} \tag{3}
$$

Equations (3) will elucidate the above-made assertions. In particular, there is no line possible that $t_o(w)$ would occur in, i.e. the time duration of the program for the processing of w without accessing the on-chip data cache, which is the reason why the loads and stores from and into the on-chip cache, respectively, cannot be taken into account separately using the method chosen. $t_s(w)$ are the times which have to pass before the exchange of data between the on-chip data cache and SL cache has been completed. Whenever the data are located in the home memory, the hub has to be involved, which is estimated with $t_{h_h}(w)$. $t_m(w)$ describes the accesses to the memory chips themselves. In our machine, memory size may vary quite drastically between node boards. In pilot investigations, it was possible to show that the measuring program does not react at all to the different installation states, with the spectrum ranging from 2...8 banks consisting of 32, 64, or 256 MB DIMMs with 16 and 64 MB memory density. Another negligibility is the assumption that the kernel is completely relaxed at the time of measurement and that its memory usage in turn remains negligible.

Line 4 in (3) describes the access to the memory of the other node board in the same corner. Here again the hub is needed for the transition to the so-called *interconnect fabric*, for which a router has to be passed in order to access remote memory via remote hub. Since we have to presume that the entire hub activities are markedly different from the access to the home memory and that, besides that, two hubs are involved, $t_{h_r}(w)$ is set in this line, with one of the routers at $t_r(w)$. The interpretation of the remaining lines should be evident now. Thus, the last line describes the access to one of the node boards logically located on the opposite corner. The six functions contained in (3) are defined in the following way:

$$
\begin{aligned}
t_o(w) &= t_{o,0} + t_{o,1}\,\frac{w}{8\,\mathrm{Byte}} \\
t_s(w) &= t_{s,0} + t_{s,1}\,\frac{w}{32\,\mathrm{Byte}} \\
t_{h_h}(w) &= t_{h_h,0} + t_{h_h,1}\,\frac{w}{128\,\mathrm{Byte}}
\end{aligned}
$$

$$
\begin{aligned}
t_{h_r}(w) &= t_{h_r,0} + t_{h_r,1}\,\frac{w}{128\,\mathrm{Byte}} \\
t_r(w) &= t_{r,0} + t_{r,1}\,\frac{w}{128\,\mathrm{Byte}} \\
t_m(w) &= t_{m,0} + t_{m,1}\,\frac{w}{128\,\mathrm{Byte}}
\end{aligned}
\qquad (4)
$$

Thus, it should be possible to describe some vital parameters of the Origin2000 memory hierarchy based on ten constants with two constants associated with the application. In addition, the following state of affairs needs to be considered. If prefetching is dispensed with in *my_mem_acc()* (1), five of the twelve constants in total disappear in good approximation, viz $t_{s,0}$, $t_{h_h,0}$, $t_{h_r,0}$, $t_{r,0}$, and $t_{m,0}$. This is theoretically most illuminating, as it suggests that delays in their original sense can be neglected in the Origin2000 memory hierarchy, that is to say whenever not only one data is accessed. Consequently, what we have here are basically pure bandwidths, which, of course, transform into delays in extreme cases.

The consequence of this discussion is the following: in the presence of prefetching, the situation is a quite different one and extremely difficult to assess. In that case, the five above constants do not disappear at all, i.e. real delays seem to occur, whose cause has to be seen in the occasionally jammed data paths, and the fact that more data are moved than necessary. Doubtless, prefetching is a good thing, yet in this context it was extremely disturbing, and the search for the cause was very time-consuming. Of course, the program used cannot supply comparable values to $t_{top}(p, w, s_1 ... s_7)$, since the concrete location of the data can only be manipulated indirectly. Rather, corresponding expected values $\overline{t_{top}}(a, w, s_1 ... s_7)$ are measured, based on

$$
\overline{t_{top}}(a, w, ...) = \frac{\int_0^a t_{top}(p, w, ...)\, dp}{\int_0^a dp} \qquad (5)
$$

In principle, (5) transforms the step functions defined by (2) into functions which can be differentiated, and should be predictable from figure 1. This way, the undesirable dependence on p is replaced by one on a, the size of the entire accessible memory *area*.

3.2 TLB misses

The previous paragraph discussed ways of describing the topology inside the machine. In the following, the point is to track down fudge effects which may be disturbing the result of the approximation described below. One such effect is caused by the fact that the 64 entries of the Translation Lookaside Buffer (TLB) are exhausted so that the information in the page tables has to be accessed. Interestingly, this happens with as low as 1 MB of administrated memory in our system. It would certainly be sensible to make the default page size dependent on the size of the SL caches during the boot step of the machine so that with a data amount of 4 MB, which could still be located in the SL cache, no TLB misses would have to be handled. In figure 1, TLB misses can be identified from relatively minor kinks at 1 MB, particularly for small values of w. This effect is here taken into account with the following correction term, where the required integration of the corresponding step function has been carried out manually.

$$\overline{t_{tlb}}(a, s_p) = \begin{cases} 0 & \text{for} \quad a < 64\, s_p \\ t_{tlb} \left(1 - \frac{64\, s_p}{a}\right) & \text{for} \quad a \geq 64\, s_p \end{cases} \tag{6}$$

t_{tlb} designates the time that the handling of a TLB miss takes, and s_p is the page size, here equated with 16 KB. Through (6), another degree of freedom, t_{tlb}, comes into play.

3.3 One more Correction

The second and last correction in improving the fitting of the theoretical curves to the experimental findings has for the time being not been understood yet. Independently, it has proven useful to add the following term:

$$\overline{t_{pt}}(a) = \begin{cases} 0 & \text{for} \quad a < 256\,\mathrm{MB} \\ t_{pt} \, \frac{\log_2 (a/\mathrm{Byte}) - \log_2 2^{26}}{\log_2 2^{26}} & \text{for} \quad a \geq 256\,\mathrm{MB} \end{cases} \tag{7}$$

The pre-factor t_{pt} in (7) represents a last degree of freedom introduced at this point. For the remainder of this correction, no speculations should be made. Nonetheless, there are certain analogies to formulae for the number of operations necessary for the searches in binary trees.

4 Least Square Fit Results

For the fitting of the free parameters to the more than 1000 measurement points, the following problem was worked on, where i runs across the entirety of all n values.

$$\sum_{i=1}^{n} \frac{(\overline{t_{top}}(a_i, w_i, ...) + \overline{t_{tlb}}(a_i, ...) + \overline{t_{pt}}(a_i, ...) - t_i)^2}{t_i^2} \longrightarrow \text{Minimum} \tag{8}$$

Maple V, R3 served as the tool. In order to efficiently face up to potential stability problems, the linear system of equations was built up using arbitrary precision rational arithmetic, to be eventually solved in standard floating point arithmetic. Table 1 shows the results gleaned by this method, and the results gained at least appear plausible. The relative error of the individual value is estimated at 20 percent. A comparison of the values with those resulting from direct analysis of the hardware are still outstanding.

Table 1: Important hardware parameters of the memory hierarchy of the Origin2000 system seen from a measuring program point of view

Mnemonic	Symbol	Value	Related to
on-chip cache $\longleftrightarrow$ SL cache	$t_{s,1}$	53 ns	32 Bytes
local hub for home	$t_{h_h,1}$	110 ns	128 Bytes
local hub for remote + remote hub	$t_{h_r,1}$	358 ns	128 Bytes
standard router	$t_{r,1}$	68 ns	128 Bytes
memory + memory controller	$t_{m,1}$	468 ns	128 Bytes
TLB miss handling	t_{tlb}	1005 ns	1 TLB miss
2-nd type correction	t_{pt}	8356 ns	

From the angle of the results of table 1, the curves contained in figure 1 can be interpreted in the following way. Delay times in the accesses to the home memory are primarily a result of the memory chips. If the home memory has to be left, data flow via the routers. The latter may be fast; nevertheless, the concomitant additional hub activities on both sides of these routers cause noticeable disturbances, i.e. the realization of the protocol. Accesses to the memories of the node boards located further away do not impair times significantly any more. This is more clearly indicated by the depiction of the values contained in table 1 in the form of MFLOPS. Here, the existence of an infinitely fast processor is presupposed, which executes sign changes on consecutive floating point data without paying attention to the possibility of accelerating this task by means of suitable interventions, i.e. it does nothing but sit and wait for the arrival of the respective incoming data.

5 Conclusion

With the help of a carefully considered measuring program and a transparent and manageable theory, seven parameters of the Origin2000 system have been identified, five of which concern the hardware only. Even if the accuracy with which these parameters have been worked out may not be maximally convincing, it will, however, allow for certain simulations that essentially

Table 2: Performance limits for consecutive sign changes caused by the Origin2000 hierarchy level the data reside on

Mnemonic	Level	MFLOPS
SL cache	2	75.5
home memory	3	20.3
same corner	4	14.5
neighbour corner	5	13.6
next but one corner	6	12.9
spatial diagonal	7	12.2

make for a more intimate, better understanding of the machine than would have been possible on the basis of the number of wires and the clock rates. The main reason for this lies in the fact that the measuring program does not only capture wire bandwidths, but also delays caused by the logic, though in a rather indirect way.

Other scenarios concerning the measurement of the same quantities may also be conceivable on the basis of POSIX threads, which would allow for separating allocation/initialization and access activities from each other locally.

Acknowledgements

I want to thank Dr. W. Oed from Silicon Graphics GmbH, Germany, for helpful discussions in understanding many of the details of the Origin2000. Thanks are also due to Dr. S. Maletti and J. Weller from the Universitätsrechenzentrum (TU Dresden) for making an unloaded machine exclusively available for hours on end.

References

[1] Dowd K.: *High Performance Computing*. O'Reilly & Associates, Inc., Sebastopol, June 1993.

[2] Silicon Graphics Inc.: *Origin Servers Technical Report*. April 1997.

Originally published under http://www.cea.fr/~workshop/Program.html, 1997.
Also available as internal report ZHR-IR-9701.