MPL-54

To: Multics Performance Log

From: A. Sekino

Date: June 16, 1971

Subject: Measurement of Memory Cycle Interference

It has been observed that the per event overhead time of a two processor system is much longer than that of a one processor system (See MPL-53). This increased overhead is due to (1) the memory cycle interference, i.e., the interference caused by occasional conflicts of two processors for memory cycles of a particular core memory box and (2) the data base lockout. As far as the system overhead is concerned, the second is the major effect especially in the current Multics implementation (this will be reported in a separate MPL in the near future). However, the first effect increases not only the overhead but also the effective CPU demand of each user's computation simply because processors run more slowly.

This memo describes the result of experiments evaluating the first effect, i.e., the memory cycle interference. For this purpose, several special programs were coded in assembly language. Then, these programs were run and measured using a real-time hardware clock. It was found that programs representative of typical Multics users run at the approximately 20% reduced speed when the two programs compete for memory cycles of the same core memory box at all times and that they run at the approximately 7% ($\simeq$ 20 x 1/3) reduced speed if the programs are scattered over three core memory boxes. One of the programs representative of memory bound computation, which were run on the two CPU system, competing in a single core memory box showed a show-down effect amounting to 52%. The consistency of these results imply that the typical Multics users are spending 7% increased CPU time because of memory cycle interference on the normal two CPU configuration system.

1. Introduction

The degree of memory cycle interference depends on many factors such as the system configuration (the number of processors and core memory boxes, the relative location of programs in the core memory system), the characteristics of the program being run (the proportion of time during the length of which the memory must be tied up, the length of each memory cycle, the relative timing of consecutive memory cycles), and the characteristics of the program which executes on another CPU. Therefore, if the execution time of a particular

program is to be measured on the two CPU system for the purpose of comparing the resulting execution time with the corresponding execution time measured on the single CPU system, the system configuration and the characteristics of two mutually competing programs must be accurately specified.

Because we are most interested in the performance of typical Multics programs running on the normal two CPU configuration system, the memory cycle interference of a typical program running against another typical program must be evaluated in the case of three core memory boxes. The measurement of interference of memory bound programs is also interesting because such measurements give a sort of upper bound for the typical degree of interference. On the other hand, the interference to be encountered in the case of only one core memory box (complete interference) should be measured, because the result of such basic measurements can be used in predicting the degree of interference expected in the case of the arbitrary number of core memory boxes. However, it will turn out that this measurement is fairly complex and hard, as will be described later.

2. The Programs to be Measured

To begin with, two short instruction sequences intended to represent a portion of typical Multics user programs and two memory bound instruction sequences intended to represent a sort of the worst case were created to be incorporated into the programs to be run in the experiments. In designing the typical instruction sequences, it was carefully considered that the sequences should possess at least the average characteristics, of typical Multics programs, concerning both the instruction execution rate and the number of memory references per instruction*. Therefore, these sequences are assured to possess at least the typical characteristics concerning the proportion of time during the length of which the memory is to be tied up. On the other hand, the memory bound sequences were more arbitrarily designed. Actually, a particular instruction chosen somewhat arbitrarily is simply repeated many times using a "repeat" instruction in each of these sequences. The usage of repeat instruction makes it possible to avoid instruction fetching otherwise needed each time the same instruction is repeated and therefore only the memory reference needed for operand manipulation is periodically repeated.

Each of the particular instruction sequences used in the experiments is given below. The sequences named "mipt_ty" and "mip_ty*" are intended to represent a portion of typical programs and the ones

---

* M. Schroeder reported in MPL-51, 52 that 341,945 instructions making a total of 419,425 memory references were executed per second, on average, by the heavily loaded single processor system.

named "mipt_lda" and "mipt_orsq" are memory bound sequences*. The execution of each sequence takes roughly 200 ~ 300 microseconds on the single processor system. Each program was designed to contain one of these sequences and the chosen sequence was repeatly executed 100 times using an outer loop in the program. In order to detect the distortions caused by occasional interrupts and drum transfers, the real-time system clock was read at both the beginning and the end of instruction sequence in each of those 100 repeated cycles. If the execution time of a cycle

```
(mipt_ty)                (mipt_ty*)               (mipt_lda)

          lda    y                 lda    y2       rpt 200 times
          aos    w                 aos    w2       lda x,7
          ada    x                 ada    x2
          sbaq   z                 sbaq   z2
10 times  ada    y       10 times  ada    y2
          eapbp  bp|0,*            staq   u        (mipt_orsq)
          ldaq   bp|0              eapbp  bp|0,*
          eax0   -5                ldaq   bp|0      rpt 50 times
          eax1   -1,1              eax0   -5        orsq x,7
          tnz    -9,ic             eax1   -1,1
                                   tnz    -10,ic
```

was found to be longer than the predetermined threshold time which is properly longer than the expected execution time, then the measured execution time of that cycle was judged to have been distorted and simply discarded. Furthermore, each program was coded into a one-page impure segment in order to assure that both instruction and operand references are directed to the same memory controller.

3. The Measurement Method and the Associated Problems

It was initially decided that all the experiments would be carried out from the user consoles, during the normal user session, using only facilities which are available to normal console users. Under this condition, no particular difficulty was encountered on the single processor system, in measuring the execution time of the programs mentioned above, except the careful setting of the threshold time to discard the distorted data. The measured execution times were found to be fairly stable and almost independent of the system conditions, possessing a distribution with a small variance. On the other hand, there exist some difficulties in measuring the program execution times on the two CPU system, because many factors concerning the programs and the system affect the result significantly, as mentioned in the introduction.

---

* The sequence "mipt_ty" was designed by J. H. Saltzer.

Two kinds of experiments were carried out for each of these four programs on the two CPU system. The first kind of experiments aim to evaluate the degree of memory cycle interference to be experienced by users competing with normal (typical) Multics users running on the Multics full configuration system (2 CPUs and 3 core memory boxes). For this purpose, each program was run when the full configuration system was heavily loaded, so that it is relatively likely that one of the normal Multics users is using another processor at almost any time during which each of those programs is being run. Because the characteristics of the user running on another processor and its usage of core memory connot be exactly controlled, except in a statistical sense, the result shows a distribution with a large variance. Therefore, the sample size of this experiment was chosen to be much larger than that of the single processor measurements.

The second kind of experiments aim to evaluate the degree of memory cycle interference to be experienced by each user running against the same sort of user as himself running on another processor using the same core memory box at all times. In this experiment a special arrangement must be made to ensure that (1) two programs of the same sort use the same core memory box at all times during the measurement period and (2) these programs are running at all times during the same period on both processors. The first requirement was satisfied by combining those two programs to be run simultaneously into a page of shared segment in such a way that the segment has an entry for each impure program, as shown in Figure 1. The second requirement was satisfied by using a two-step synchronization of the operations of two programs. The first step synchronization, intended to be a rough synchronization, was carried out using Multics inter-process communication facilities. Now let those two processes expected to execute the combined segment be Process A and Process B. As shown in Figure 1, Process A activated from a console blocks itself first and then Process B activated from another console wakes up Process A and calls the combined segment via one of those two entries, continuing to use, say, CPU 2. The awakened Process A is then scheduled, obtains CPU 1, and calls the same combined segment via another entry. Therefore, the delay associated with the scheduling of Process A corresponds to the error of the first step synchronization. The second step synchronization, intended to be a refined synchronization, was carried out using an interlocking technique. The use of this technique allows each process to start executing the imbedded instruction sequence chosen for this combined segment only when both processes are ready to do so, as is clear in Figure 1. Because the eligibility time (quantum) of the Multics multiprogrammed scheduling scheme is fairly short (two seconds), the first step rough synchronization was needed. Actually, it was observed in one of the measurements made on the moderately loaded system (46 simultaneous users) that the error of the first step synchronization ranged from 0.4 seconds to 2 seconds and the error of the (successful) second step synchronization from 1 to 10 microseconds.

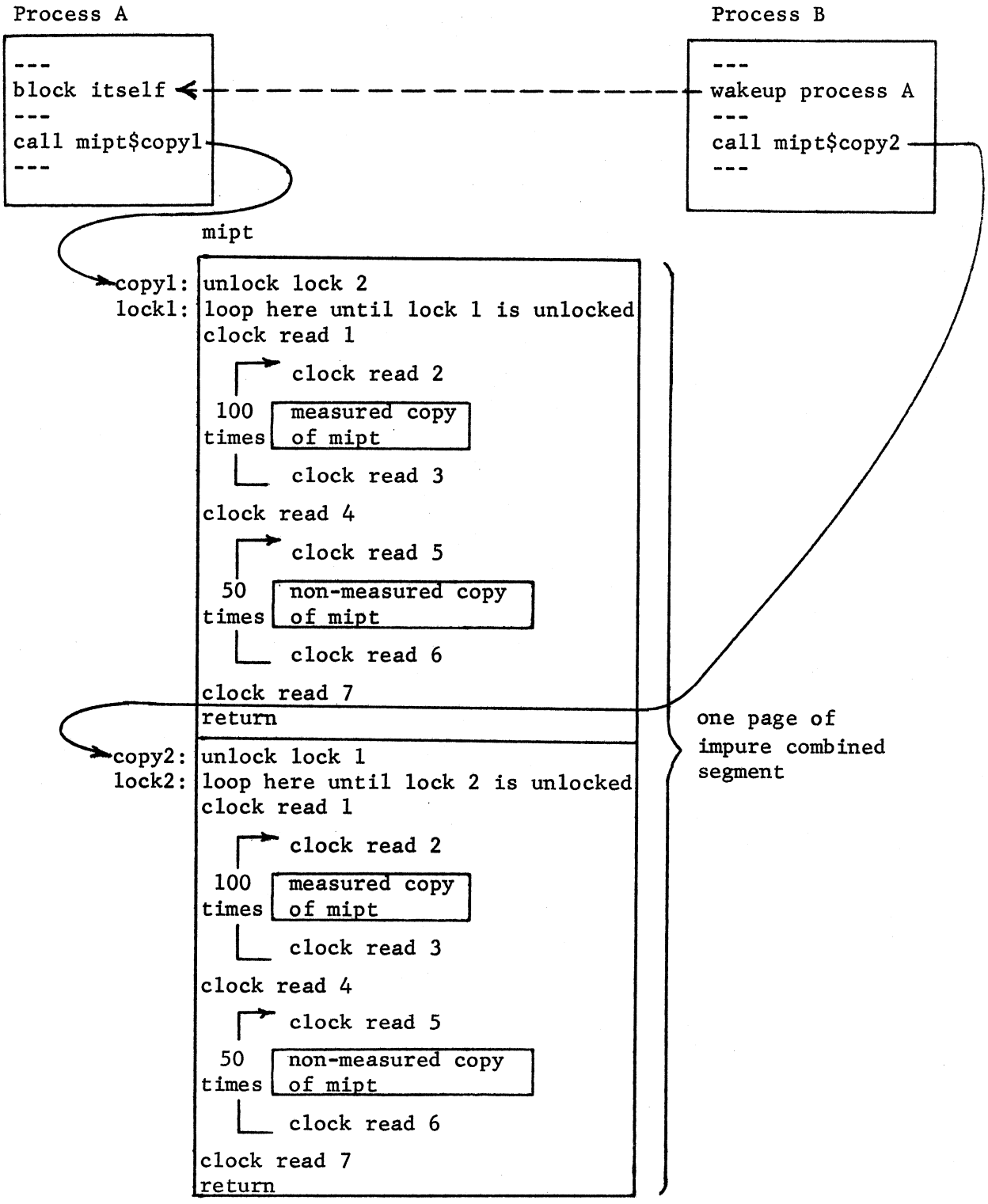

Figure 1: Two-Step Synchronization of Two Processes Executing the Impure Combined Segment

In this manner, each of the memory bound programs (mipt_lda, mipt_orsq) were run against another copy of itself. Similarly, the program "mipt_ty" intended to be a typical program was run against another copy of "mipt_ty", but in this case a very interesting timing problem was observed. When two "mipt_ty" copies were executed in parallel in almost complete synchronization on two processors, the measured execution time of each program was found to always be one of two widely separated values, each of which is about equally likely. These two values apparently correspond to two modes of precise, instruction-by-instruction, synchronization of the identical programs. Therefore, it was decided to get the second copy (mipt_ty*) to have slightly longer length by inserting "staq" instruction in the middle of mipt_ty and then those two conjugate programs (processes) will be hopefully staggered in all ways when they are run simultaneously, avoiding the above timing problem.

Another care exercised concerns the distortion caused by interrupts and drum data transfers. It was sometimes observed that the second step refined synchronization was not successful; while Process B was waiting for Process A at "lock 2" after unlocking "lock 1" for Process A, Process B was preempted by the higher priority process, which had been possibly in a page-wait status when Process B had obtained the processor, and then Process A arrived at "lock 1" using another processor and started executing the imbedded instruction sequence much earlier than Process B. In order to decrease the possibility of preemption interrupts during the refined synchronization period, the second kind of experiments were, after all, carried out on the lightly loaded system. The occasional preemption interrupts were detected by examining the difference of the first clock readings of the competing programs (see Figure 1) and these distorted data were all discarded.* The interrupts and drum data transfers may occur also during the execution of the imbedded instruction sequence. This kind of distortions were detected by examining the difference of a pair of clock readings which sandwich each instruction sequence and the measurement datum was discarded if any interrupt or drum data transfer was detected in this manner during the execution of any part of these two competing instruction sequences because it is not guaranteed that the two processes ran without any distortion. Moreover, the same instruction sequence (non-measured copy; See Figure 1) was executed for some time after the measurement of the instruction sequence was finished in order to guarantee that the slower process also competes with the supposed competitor even towards the end of the execution of the instruction sequence being measured.

---

* The use of "interrupt inhibit" mode would solve this problem completely if the normal users were allowed to use it.

4. The Measurement Results

The execution time to be required by 100 passes of each instruction sequences was measured, controlling the system configuration and avoiding the distortions, in the way described in the previous section. All the results are summarized in Table 1.

When the programs were run on the single processor system it was observed that each of them runs approximately 30% more slowly than the expected time derived being based on the catalogued instruction execution time for Honeywell 645.* According to J. Ammon's speculation, this discrepancy is due to the combination of the slower memory cycles (5 ~ 10% slower than the catalogued values), the slower logic operations (details are not known), and the delay caused by associative memory searches (additive 300 nsec. per successful search).

The results of mipt_ty and mipt_ty* obtained by running against normal (typical) users on the full Multics configuration system suggest that the typical users run at the approximately 7% reduced speed because of only memory cycle interference.** (In deriving the effective speed of typical users on real Multics, other factors such as data lockout must be considered). When a typical program (mipt_ty) was run against another typical program (mipt_ty*) using two processors and only one core memory box, the programs ran at the approximately 13 ~ 20% reduced speed. This result agrees to the prediction which can be derived from the 6.5% degradation measured on the full configuration system and the assumption that normal users use each core memory box with approximately equal probability. That is to say,

$$106.5 \simeq 100 \times \frac{2}{3} + 119.9 \times \frac{1}{3}.$$

---

* For example, the expected execution time of one hundred cycles of "mipt_ty" is 22341 microseconds while the measured execution time was 28905 microseconds (29% slower) on the single processor system.

** The execution time of each sequence against normal users depends which users happen to be simultaneously on the system. Therefore, the experiments for these four sequences were carried out in the same console session.

Table 1 The Measured Program Execution Times

instruction sequence		1 CPU [20 samples in each experiment]	2 CPUs: against normal users (3 core boxes)	[sample size] (percent increase in time)	2 CPUs: against another copy or its conjugate (1 core box)	[sample size] (percent increase in time)
mipt_ty	ave.	28905 μsec	30784 μsec	(+6.5%)	34658 μsec	(+19.9%)
	min.	-290 μsec	-1553 μsec		-1690 μsec	
	max.	+690 μsec	+1996 μsec		+4341 μsec	
	ave. rate	.353 mips	.332 mips	[40]	.295 mips	[36]
mipt_ty*	ave.	31082	33451	(+7.5%)	35071	(+12.8%)
	min.	-359	-1713		-1934	
	max.	+585	+1776		+3929	
	ave. rate	.360	.335	[40]	.319	[36]
mipt_lda	ave.	31696	34578	(+9.1%)	43160	(+36.2%)
	min.	-760	-3347		-604	
	max.	+624	+3852		+1642	
	ave. rate	.640	.587	[40]	.470	[56]
mipt_orsq	ave.	19297	20463	(+5.8%)	29279	(+51.7%)
	min.	-380	-1134		-847	
	max.	+825	+984		+795	
	ave. rate	.275	.259	[40]	.181	[38]

* mips = million Honeywell 645 instructions per second

** The minimum (maximum) execution time observed in each experiment is represented as a deviation from its average.

Each of memory bound programs naturally suffered from severer degradation when they were run on the two CPU system. Especially, when mipt_orsq was run against another copy of mipt_orsq using a single core memory box 52% degradation was observed. This actually means that the proportion of time during the length of which the memory must be tied up for the operand manipulation of a single copy of mipt_orsq is (at most) 76% (= 152% /2) of its execution time.* The reason why the same sequence did not suffer very much when it was run against normal users is that the "orsq" instruction uses a "read-alter-rewrite" cycle, which is much longer than the more frequent memory cycles, and exclusively occupies the memory during this rather long cycle time, giving the normal user the lesser chance to execute its program. On the other hand, the "lda" instruction of the sequence "mipt_lda" uses a normal "read-rewrite" cycle and therefore this sequence suffered more (9.1% degradation) than the sequence "mipt_orsq" in the above situation.

---

* This remark is consistent with the following observation. Each repeated execution of "orsq" takes 3.69 μsec (measured) making a "read-alter-rewrite" cycle which requires 2.69 μsec (estimated). Therefore, the proportion of time during the length of which the memory must be tied up is about 73%.