IBM System z and @server zSeries Processor Performance: Processor Design Considerations

Customers migrating to zSeries processors (z800, z900, z990, and z890) have reported some programs are using larger amounts of CPU time than expected when compared to LSPR projections. It is not uncommon to have individual programs under-perform or over-perform, as judged by LSPR. After review of some of these programs it has been determined some of the programs are under-performing because they are using unexpected programming practices. These coding practices are being impacted by elements of the microprocessor design as implemented in the 9672, eServer zSeries and IBM System z9 processors.

Overview:

For the IBM eServer zSeries and IBM System z9 families of processors some major new processor design changes have been implemented. Some of these changes are:

- a split Level-1 Instruction / Data processor cache
- a larger level-2 processor cache
- improved branch prediction
- increased prefetch and pipeline exploitation

All modern microprocessor architecture is designed with the understanding there is a codependence between a processor's design and the code it expects to execute. This codependency has caused certain coding practices to become standardized within the past 15 years. Some of these conventions involve code usability issues, (modularity of code, structured programming practices, no self-modifying code), and some involve performance issues, (separating data and instructions, localizing storage references, no self-modifying code). The vast majority of applications and user code were created with the understanding of the codependency in mind and so will react favorably to the design choices made in the 9672, the eServer zSeries and the IBM System z9 families.

Most current compilers have been written with microprocessor architecture in mind. This means compilers will automatically generate code needed to keep pipelined processors running well. This is also true of most code written to be reentrant such as the operating system, major IBM subsystems, and many vendor packages. Experience has shown most customer applications being migrated to the IBM System z9 or the eServer zSeries processors are seeing the expected relative processor performance for the migration as predicted by IBM's LSPR data.

Workload Analysis:

Reviewing some of the under-performing programs or applications which have been brought to IBM's attention, the performance shortfalls can be traced to assembler code which doesn't follow the coding conventions discussed above. Let's review the performance impacts of not following the coding practices recommended above by describing one particular interlock situation seen in some of the under-performing programs.

Instruction Fetch Interlock

In a typical program an instruction fetch interlock condition will occur when a store-type instruction modifies one of the immediate successor instructions. The successor instruction cannot start through the pipeline until the alteration completes. The interlock condition may also occur if instructions and modified data variables are too close, (in the same cache line). This type of interlock is sometimes referred to as a program store compare. Self-modifying code is another way this situation might occur. A typical use of self-modifying code would be to "create" an instruction to be executed as part of the program, and store the instruction into the stream. Another example would be to modify an EXECUTE instruction, or a branch instruction to execute differently based on program conditions.

On IBM System z9, IBM eServer zSeries, and 9672 G5/G6 processors the cache line size is 256 bytes, while on 9021 H2/H5 and 9672 G3 through G4 processors the cache line size is 128 bytes. The IBM System z9 and the eServer zSeries processors have a cache size of 256KB, and there are two caches, one for instructions and one for data, for a total of 512KB of cache. The 9672 G5/G6 also had a 256KB cache size, though instructions and data were unified in the single cache. The advantages of the System z9 and eServer zSeries split Instructions / Data cache is there is now greater bandwidth due to the use of address and data busses, and there is more cache memory available but still accessible with good access times. Each of these processor design features provides greater performance and throughput for the IBM System z9 and the eServer zSeries.

The program store compare operation forces the processor to experience a data buffer miss, subsequently the instruction buffer is purged, and this causes instructions to be re-fetched from the next common store, which in the case of the IBM System z9 and the eServer zSeries is the Level 2 buffer. With any processor design as you need to fetch data from lower levels of the storage hierarchy, memory accesses will take longer. This can have a performance impact with a Level 1 split Instruction/Data (I/D) cache design like on the IBM System z9 or the eServer zSeries, but it may have a negative effect on any processor which uses lookahead. The G5/G6 also had negative impacts from program stores but since the Level 1 cache is unified, it could resolve the program store condition within the Level 1 cache.

Benchmark Analysis:
As an example of the potential performance impact for program store compares, a simple assembler benchmark loop was run to do some stores into various data fields. In one case, the data fields are close enough to the instructions so some portion of the stores would occur to areas within the same cache line as the executing instruction, ("Program Store"). In the other case, 256 bytes of empty storage was added to separate the data areas from the instructions, ("No Program Store").

Here are the results for various machines. Times are in seconds of execution time.

	"No Program Store"	**"Program Store"**	
9672-RY4	**98.4**	**98.2**	**(essentially no impact)**
9672-Y86	**43.7**	**73.7**	
9672-Z47	**34.3**	**57.6**	
9672-ZY7	**34.4**	**58.1**	
2064-109	**29.7**	**102.2**	

The 2064-109 shows the most elongation, because of the split Level-1 cache design, but the G5, and G6 processors also see a penalty. One thing to remember about this simple benchmark job is nearly 100% of the code execution was comprised of the loop which executed with a "program store" behavior. **A SYNTHETIC BENCHMARK JOB OF THE TYPE USED ABOVE WILL, IN ALMOST ALL CASES, NOT REFLECT A PRODUCTION OPERATING ENVIRONMENT.** The benchmark results can be viewed as a worst case scenario. In any program which has interlock conditions, the affected areas will most likely not be across the entire execution of the source program and so the expected impact would potentially be much less.

Correction of the interlock conditions has the potential to affect the CPU demands of applications, significantly reducing the application's CPU costs across many different generations of processors.

This also explains why migration to new processors may cause some programs to use more CPU than expected based on LSPR measurements. In the case of the migration from the 9672-ZY7 to the 2064-109, the expected ITRR was 1.17 and the program saw an actual ITRR of 1.16 when using programming techniques without program store sequences. As stated before, code which is compiled, reentrant, or follows standard programming processes, will not be impacted.

Detection and Correction:

After migration to the IBM System z9 or the eServer zSeries processor if there are programs which are significantly below performance expectations as set by LSPR the following steps can be used to try to improve the performance of these programs. Migrations from an eServer zSeries processor to an IBM System z9 processor would be highly unlikely to see this issue since both processors in the migration path are already using the same type and size of cache.

1. Identify the programs in question via an analysis of the SMF 30 records for the affected work. Look for programs which are significantly outside the expectations set for the work by LSPR, and which are large users of CPU seconds. Though many jobs may have interlock conditions it may be necessary to apply a cost benefit standard to determine which programs should be analyzed. Large CPU consumers offer a greater potential for significant savings. Another choice would be to review subroutines or programs used by a large number of users.

2. If possible use a hot spot analysis tool to identify which sections of an application are using the most time. Certain tools allow the analysis to be done at the instruction level, and it may be useful to look for excessive CPU at a very granular level. The best situation for this type of analysis is to use a hot spot analysis tool against the program when it runs on different processors. By comparing the before and after hot spots it may be easier to detect code segments which are different.

The only way to ultimately identify and correct the situation is to examine the source code. The presence of an interlock condition by itself may not be reason enough to correct the code. The condition should be in a section of the code which is heavily used, or frequently executed. A circumvention such as taking an interpretive execution and compiling the source may provide needed relief.

Coding Examples:

Some examples seen in application and user code are:

1. Storing into the object of an EX instruction before executing it.
2. Storing into what starts out as a no-op branch, but which has a condition placed into it as a result of a halfword store
3. Storing entire instructions into a storage area before they are executed.
4. Modifying instruction operand references before they are executed.
5. Storing within multiple inline storage areas always close to currently executing instructions.

Additional Information

Sources of information on programming practices can be found by reviewing the numerous publications available. Some programming sources which may be of interest are:

1. Advanced Assembler Language and MVS Interfaces: For IBM Systems and Application Programmers by Carmine Cannatello
2. Computer Architecture, A Quantitative Approach by John Hennessy and David Patterson
3. The Practice of Programming by Brian W. Kernighan and Rob Pike
4. Software Optimization for High Performance Computing by Kevin Wadleigh and Isom Crawford
5. Computer Engineering Handbook by C.H. Chen

For information on the microprocessor design employed by the IBM Servers interested parties can read:

1. The Microprocessor Architecture of the IBM eServer z900 Processor by E.M Schwarz, M.A. Check, C.L. Shum, T. Koehler, S.B. Swaney, J. D. MacDougall, and C. A. Krygowski,IBM Journal of Research and Design, Vol 46, No. 4/5.
2. IBM S/390 Parallel Enterprise Servers G3 and G4 by G. S. Rao, T. A. Gregg, C. A. Price, C. L. Rao, and S. J. Repka, IBM Journal of Research and Design, Vol 41, No. 4/5.
3. S/390 Parallel Enterprise Server Generation 3: A balanced system and cache structure by G. Doettling, K. J. Getzlaff, B. Leppla, W. Lipponer, T. Pflueger, T. Schlipf, D. Schmunkamp, and U. Wille, IBM Journal of Research and Design, Vol 41, No. 4/5.
4. Shared-cache clusters in a system with a fully shared memory by P. Mak, M. A. Blake, C. C. Jones, G. E. Strait, and P. R. Turgeon, IBM Journal of Research and Design, Vol 41, No. 4/5.
5. The S/390 G5/G6 Binodal cache by P. R. Turgeon, P. Mak, M. A. Blake, M. F. Fee, C. B. Ford III, P. J. Meaney, R. Seigler, and W. W. Shen, p. 661, IBM Journal of Research and Development, Vol 43, No. 5/6.
6. IBM S/390 Storage Hierarchy - G5 and G6 performance considerations by K. M. Jackson and K. N. Langston, IBM Journal of Research and Development, Vol 43, No. 5/6.
7. S/390 Microprocessor Design by C.F. Webb, p.899, IBM Journal of Research and Development, Vol 44, No. 6.

The IBM Journal of Research and Development can be found at: http://www.research.ibm.com/journal

Special Notices