Cache Misses Challenge to Modern Processor Architectures

Milco Prisaganec[1] and Pece Mitrevski[2]

Abstract – **The increase of cores by replication, the increase of threads in the core, the usage of several levels of cache memory and shared memory in the processor are the base for increased processing power, but also the reason for cache misses. These, in turn, increase the average memory access time and decrease processor performance. We performed simulations of different configurations using SMPCache (Simulator for Cache Memory Systems on Symmetric Multiprocessors) and conclude that commercial architectures need to implement techniques for increasing memory level parallelism efficiency, in order to reduce the number of cache misses.**

Keywords – **Memory latency, MLP, memory level parallelism, multi core processor, cache miss.**

I. INTRODUCTION

Memory latency is the bottleneck at out-of-order instruction execution in modern high-performance processors. With the increase of the difference between processor and memory speeds, stalls become more frequent because of cache misses, and penalties can even be a few hundred cycles. The modern architecture of multi-core processors that share same memory is enhancing the complexity of this problem.

Memory architecture is the basic technique for depleting memory latency. But at the same time it is the reason for the appearance of cache misses that damage processor performance. In this paper we used a processor architecture simulator and using different benchmarks we counted the cache misses and established what they depend on.

In the second part we present the basics of Memory Level Parallelism (MLP) and its significance for processor performance. Then, in the third part we present some techniques for increasing the memory level parallelism efficiency. The fourth part gives the results we gathered using a higher processor architecture simulator and the next one, part five, analyzes the results and gives conclusions that might show useful for researchers and designers for future processor development.

II. THE PARADIGM OF MEMORY LEVEL PARALLELISM (MLP)

The basic goal of processor designers is getting as higher performance as possible, for a lower price. At the beginning, the increase of performance was caused by the increase of frequency and number of transistors in the processor [1]. Reaching the upper frequency limit of 4 GHz and the increase of energy consumption are the basic physical limitations for further performance boost. Designers find the solution in architecture with out-of-order instruction execution [3], i.e. Instruction Level Parallelism (ILP) [2,5,10]. That would enable a wider instruction pipeline through the processor and a larger number of executed instructions per cycle. Intel's analysis shows that ILP affected the performance increase until year 2000 and then it lost its power. Some authors go far enough to argue that ILP even needs to be dismissed, because the enlarged instruction window is the reason for cache misses. Memory latency decreases with the implementation of memory hierarchy, but with consequent cache misses. The processor for each cache miss penalty has to pay hundreds of cycles [3].

MLP is a concept that needs to enable parallel execution of more memory operations in order to avoid cache misses and memory latency. The concept of parallel execution of more memory instructions (MLP) is used as an idea for hiding high memory latency. Its goal is to decrease the number of cache misses, i.e. to bring the needed data to L1 before the processor issues a request for access to them.

III. TECHNIQUES TO INCREASE MLP

The penalties that are paid by the processor because of cache misses significantly decrease its performance. The goal of these techniques is to unblock the processor and to provide a continuous pipeline with parallel execution of more memory operations.

One of the ideas for unblocking the processor is enlarging the instruction window. The implementation of hardware leads to increased energy consumption. In [4] the technique of Runahead execution which succeeds at virtually enlarging the instruction window is given. When the processor is blocked with a long latency instruction the status of the architectural registers is preserved. After that the processor enters "runahead mode". In that moment a fake result for the stalled instruction is delivered which eliminates it from the instruction window. That allows the processor to unblock and keep on with the continuous execution of the next instructions. After that, the instructions that follow after the stalled one are also falsely executed from the instruction window. But in this mode they don't regenerate the status of the registers. When the stalled instruction will be executed, the processor returns to "normal mode". It retrieves the saved status of the registers. Then it starts re-executing the instructions, starting with the stalled one.

[1]Milco Prisaganec is with the Faculty of Administration and Information Systems Management, University "St. Kliment Ohridski", Bitola, Macedonia, E-Mail: milco.prisaganec@gmail.com

[2]Pece Mitrevski is with the Department of Computer Science and Engineering, Faculty of Technical Sciences, University "St. Kliment Ohridski", Bitola, Macedonia, E-Mail: pece.mitrevski@uklo.edu.mk

When a cache miss occurs, the processor resources are blocked by the instructions dependent on the instruction that caused the miss. The Continual Flow Pipelines (CFP) technique, analyzed in [9], allows uninterrupted execution of the independent instructions by the instruction that misses. It enables liberating of the blocked resources and allocating them to the independent instructions, which keeps the pipeline undamaged. The idea of CFP is to eliminate the instruction that missed and the instructions that depend on it from the pipeline, thus liberating the resources. By the time the instruction with a miss is released, the resources are liberated and independent instructions from the cache miss can be executed.

The enlargement of the instruction window causes implementing a larger ROB in the processor. The Out of order commit technique [7] includes using control points, which really give a picture of the state of the processor in a given moment, enables efficient out-of-order execution of the instructions without a ROB structure.

Most of the processor performance increasing techniques that use MLP is trying to parallelize the cache misses. That way, after a certain time interval several cache misses would be resolved. But that is the case when a few sequential cache misses appear. The authors [8] point out that if the replacements in cache memory are done consciously, most of the isolated misses can be avoided.

Modern processors implement more threads in every core. That allows parallel execution of more processes. When the thread gets blocked in the processor, it holds the resources causing a bottleneck which blocks the remaining threads. To avoid the previous limitations in [6] an Aware Runahead Threads policy has been proposed. The idea of this policy is that runahead threads should be used only if there are conditions for memory level parallelism in the near future, if not the thread is blocked and the speculative instruction execution is not done.

IV. Testing processor architectures with a Simulator

On more occasions in this paper we underlined that for every cache miss the processor pays penalties - as many cycles as needed for the data to be brought into cache memory. The decrease in processor performance depends on the number of cache misses, i.e. the time spent resolving cache misses. That's why the goal of this paper is to present what is really happening at the time of communication between the processor and the memory system.

For that purpose we used the Simulator for Cache Memory Systems on Symmetric Multiprocessors (SMPCache), a tool from the Department of Computer and Communication Technologies at the University Extremadura Escuela Politecnica in Spain. The simulator enables configuration of different processor architectures with changes in these parameters: the number of processors, cache coherence protocol of the bus, schemes for bus arbitration, word length, number of words in a block, the size of basic memory, the number of cache memory levels, cache memory mapping, cache memory blocks, replacement policies and writing in cache memory.

After the definition of a certain architecture of the cache memory system, the same is tested using benchmarks from the simulator itself. There are two groups of benchmarks: nine for single processor architecture and four for core architecture.

After the testing, the simulator presents the following measured characteristics of the cache memory system: number of bus transactions, number of blocks transported across the bus, number of memory accesses (taking an instruction, reading data, writing data), number of hits and misses in the cache memory.

Using the simulator, our goal was to show the cache misses, to count them and show what they depend on in single-core and multi-core processors.

A. How Cache Memory Size Influences Cache Misses

The purpose of this testing was to show how the enlargement of cache memory influences the number of cache misses. We have tested both a single-core and a multi-core architecture. In both cases there is the same conclusion. The configuration of the tested multi-core architecture was the following: 4 core processor, MESI cache coherence protocol, word length - 8 bits, block size - 256 words. The basic memory size is 1GB. The cache memory was split (instruction and data), two-way set associative, with LRU replacement policy.

We used Simple64 as a benchmark, coded for use on a multi-core processor. During the testing we only changed the size of data and instruction cache memory: 2x4, 2x8, 2x16, 2x32, 2x64, 2x128, 2x256, and the results are shown on Fig.1.

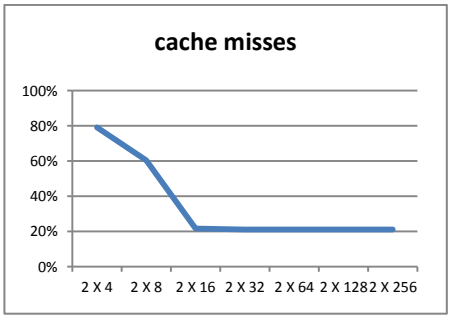

Fig. 1. Cache misses dependency on cache memory size

Fig. 1 shows the dependency between the percentage of cache misses and cache memory size. The graph shows how with the increase of cache memory the number of cache misses almost exponentially decreases. But after the increase surpasses 16Kbytes, its effect on the number of cache misses becomes almost nonexistent. We use the formula Eq(1) to see how cache misses affect processor performance:

$$T_{av} = T_{at} + P_{cm}\ X\ T_p \qquad (1)$$

where:

- Tav – Average memory access time
- Tat – Cache memory access time

- Pcm – Percentage of cache misses
- Tp – Time of cache miss penalty

We roughly determined the average access time at cache memory size of 256Kbytes.

$$T_{av} = 7.5 + 21\% \; X \; 25.5$$
$$T_{av} = 7.5 + 5.1$$

While calculating, we assume that the data that is not found in the cache is taken over from L2 of cache memory instead from main memory, trying to be as closer as possible to commercial processors. The last result shows that there is a big part of average memory access time that is a consequence from cache misses. This means that the penalty paid for cache misses almost doubles the average memory access time, thus decreasing processor performance.

While analyzing the measured results, we came to a significant result shown on Fig. 2.

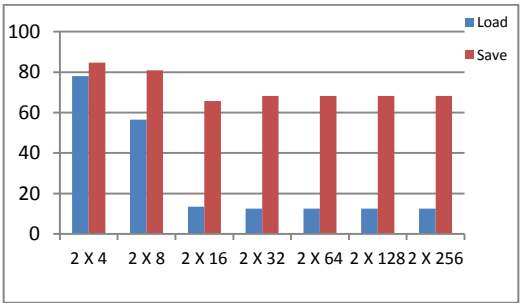

Fig. 2. Load and store cache misses

The graph shows us that for a cache memory with a small size (2x4Kbytes) the percentage of reading and writing cache misses is almost the same. But, with the increase of cache memory to value of 2x256Kbytes, the percentage of reading cache misses decreased for 85%, which is not the case with writing cache misses that have an insignificant decrease of only 20%.

B. How Memory Hierarchy Influences Cache Misses

The purpose of benchmarks in this part is to analyze the changes concerning cache misses that appear in a memory system when cache hierarchy is introduced. The simulated architecture has the following characteristics: A single-core processor with L1 cache memory (instruction and data) – 2x64Kbytes, L2 cache memory with 256Kbytes size and L3 cache memory with 512Kbytes size. The remaining parameters are configured like in the benchmarks before.

At L1 there are 88% hits and 12% misses. Out of all the L1 misses, 8.7% are misses at L2, the next level of cache memory, and 5.3% of them also miss at L3, the last level of cache memory. Table 1 presents the calculated values of average access time.

The access time for the first level of cache memory is 1.5ns, but because of the cache misses, according to the calculations and experimental results we gathered, in the case of only one level of cache memory the average access time is 13.02ns. With the implementation of a second level of cache memory, the average access time decreased for 24%, and the implementation of another level of cache memory decreased the average access time for another 17%. In another words, the three-level cache memory decreased the average access time for 37%, which of course has a significant impact on processor performance.

TABLE I
CALCULATED AVERAGE ACCESS TIME

Cache memory	Average memory access time (ns)
L1	13.02
L1+L2	9.89
L1+L2+L3	8.15

In future modern processor architectures, the memory hierarchy will be an important technique for hiding of the basic memory latency. The next expected step is to increase the number of cache memory levels implemented in the core itself. Of course, we also need to mention the consequences that appear as a result from adding an extra cache memory level, especially at multi-core processors that mostly have shared cache memory.

C. How the Number of Cores Affects Cache Misses

In this part of the benchmarks the simulator was configured as architecture with a symmetrical multi-core processor. The architecture has only one level of cache memory (L1) with a size of 512Kbytes split in two parts: an instruction part with a size of 256Kbytes and a data part with a size of 256Kbytes. The whole cache memory is two-way set associative, and the block replacement is done on the least used block. The bus supports MESI protocol for maintaining memory coherence. The size of the main memory is 1Gbytes.

During the testing the number of cores in the processor changes: 2, 4, 8, 16, 32, 64; and MDLJD is used as a benchmark. The results are shown on Fig. 3:

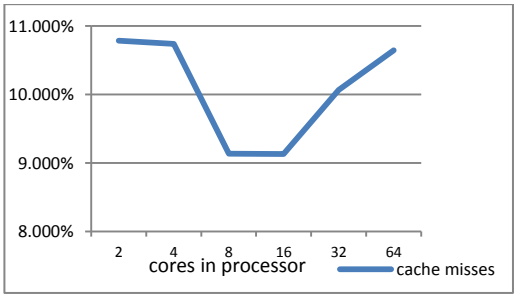

Fig. 3. How the number of processor cores affects cache misses

Fig. 3 presents the percentage of overall cache misses, in which it can be noted that in the beginning, together with the increase of the number of processor cores, the number of cache misses decreases. But when the number of cores surpasses 16, the number of cache misses starts increasing again. That shows that the replication of cores does not lead to a multiple enhancement in the performance, if proper techniques for efficient usage of MLP are not implemented.

V. DIRECTIONS FOR IMPROVEMENT OF MODERN PROCESSOR ARCHITECTURE PERFORMANCE

Besides the many papers that discuss techniques for improving the MLP efficiency, until now they have had low participation in commercial architectures. The reasons for that are the limitations each of them has, either concerning their implementation in the processor or because they would not give the expected results with a real load. For example, the runahead technique in [3] enables a virtual enlargement of the instruction window and pre-fetching of data in the cache memory that the processor will need in the future. But in the case of an isolated cache miss, after which there are no other misses, this technique damages the processor performance, i.e. the time the processor needs to enter runahead mode is wasted.

Memory hierarchy will be the basic technique for decreasing memory latency in future processor architectures as well. The last Intel i7 multi-core processor from the third generation shows that in the future, designers will pledge for implementation of more levels of cache memory in the processor core itself. But as our results have shown, that also causes cache misses which increase the average memory access time, therefore damaging processor performance. This means that MLP will face researchers and designers with the challenge to invent a technique that will decrease the number of cache misses by enabling parallel memory access. This is even more important if we also consider the fact that in future architectures the number of replicated cores and implemented threads in them will increase. Therefore, a bigger number of threads will have parallel execution in the processor, but the number of parallel memory accesses will also increase.

Besides the fact that many authors have experimentally determined that store operations are less present than load operations, we should not ignore the results that have shown us that the percentage of cache misses at store is very high (can add up to 80%). Especially when we know that there are programs in which store instructions (e.g. data acquisition) predominate.

The analysis in this paper show us that the number of cache misses depends on processor architecture, levels of cache memory, size of cache memory, purpose of the processor, but also the original code of the executed programs. The other authors' techniques that we presented here have their advantages and weak points. In order to efficiently use MLP in future architectures, aware hybrid techniques need to be designed – they would be able to parallelize fetching and bringing data to the first level of cache memory right before the processor core needs them, under different circumstances.

In the future, memory latency will keep on increasing because of the gap between processor and memory speed. The instruction window enlarges with the number of replicated cores, the number of parallel memory accesses increases, the cache memory levels grow bigger. All of that is a reason for increased number of cache misses that ultimately has an effect on processor performance. The search for solution should target MLP and its exploitation.

VI. CONCLUSION

Memory Level Parallelism is a modestly researched area, although it has a great impact on processor performance. Most of the works mentioned in this paper still have not found application in commercial processors. More significantly, the future processor development will move in a direction of parallel execution of as many processes as possible. The number of cores and threads will continue growing, and cache memory has a tendency of increasing the number of levels. These changes will enhance the flow of data and memory accesses. The designers of new processor architectures need to offer techniques that will succeed at prefetching data to the cache memory, as close to the processor as possible, to be used in near future.

The increase in frequency has reached its maximum, the increase of cores and threads has limits, the ILP loses its power and in some newer architectures it is decreased or avoided. MLP efficiency represents a challenge for the enhancement of processor performance and needs to get more attention.

REFERENCES

[1] J. L. Hennessy, D. A. Patterson, *Computer Architecture: A quantitiative approach*, 4th edition, 2007.

[2] M. Prisaganec, "Performance Evaluation of the Schemes for Dynamic Branch Prediction", Master Thesis, Faculty of Technical Sciences, Bitola, 2011.

[3] E. Sprangle, D. Carmean, "Increasing processor performance by implementing deeper pipelines", Proceedings of 29th Annual International Symposium on Computer Architecture, pp. 25-34, 2002.

[4] O. Mutlu, J. Stark, C. Wilkerson, Y. N. Patt, "Runahead execution: An alternative to very large instruction windows for out-of-order processors", HPCA '03 Proceedings of the 9th International Symposium on High-Performance Computer Architecture, pp. 129-140, 2003.

[5] M. Gušev, P. Mitrevski**,** "Modeling and Performance Evaluation of Branch and Value Prediction in ILP Processors", International Journal of Computer Mathematics, Vol. 80, No. 1, pp. 19-46, 2003.

[6] K. Van Craeynest, S. Eyerman, L. Eeckhout, "MLP-aware Runahead Threads in a Simultaneous Multithreading Processor", Proc. of the 4th HiPEAC Int. Conf, Paphos, Cyprus, pp. 110-124, 2009.

[7] Cristal, D. Ortega, J. Llosa, M. Valero, "Out Of Order Commit Processors", Proceedings of the 10th International Symposium on High Performance Computer Architecture, pp. 48-59, 2004.

[8] M. K. Qureshi, D. N. Lynch, O. Mutlu, Y. N. Patt, "A Case for MLP-Aware Cache Replacement", Proceedings of the 33rd Annual International Symposium on Computer Architecture, pp. 167-178, 2006.

[9] S. T. Srinivasen, R. Rajwar, H. Akkary, A. Gandhi, M. Upton, "Continual Flow Pipeline", Proc. of the 11th international conference on Architectural support for programming languages and operating systems, pp. 107-119, 2004.

[10] P. Mitrevski, M. Gušev, "On the Performance Potential of Speculative Execution Based on Branch and Value Prediction", International Scientific Journal Facta Universitatis, Series: Electronics and Energetics, Vol. 16, No. 1, pp. 83-91, 2003.