International Journal of Computer Applications (0975 – 888)
Volume 47– No.3, June 2012

A Dynamic Hybrid Cache Coherency Protocol for Shared-Memory MPSoC Architectures

H. Chtioui
CES laboratory
University of Sfax, Tunisia

S. Niar Lamih,
University of Valenciennes, France

R. Ben-Atitallah
University Lille Nord de France

M.Zahran
ECE Department, Polytechnic Institute, New York University, USA

JL. Dekeyser
INRIA LILLE-Europe, France

M. Abid
CES laboratory
University of Sfax, Tunisia

ABSTRACT

Nowadays, Multi-Processor System-on-Chip (MPSoC) have become an essential solution for embedded applications. In this paper we focus on MPSoCs using shared-memory programming model, which facilitates the programmer task. Moreover, one of the main factors affecting the performance of such systems is the management of cache coherency problem. In this context, we propose a new cache-coherency protocol. The proposed protocol is able to dynamically adapt its functioning mode according to variations in application memory access patterns. Experimental results show that with four cores, the new protocol reduces the number of cache misses by 77%, which results in 20% reduction in execution time and 34% decrease in the total energy consumption.

General Terms

Multi-Processor System-on-Chip (MPSoC), Shared-memory, cache coherence.

Keywords

Shared-memory, MPSoC, cache coherence, performance evaluation, energy consumption.

1. INTRODUCTION

Embedded and mobile applications are becoming more and more complex. These applications require powerful and flexible architectures with short time-to-market. The use of Multi-Processor System-on-Chip (MPSoC) architectures is the best solution for this compromise. This approach allows reaching high performances while limiting power and energy consumption. Nevertheless to be attractive, MPSoC utilization must be achieved through simple and user-friendly parallel programming models. For this reason, in this paper, we have chosen the shared memory programming model. This model simplifies porting an application from single-core to multicore platform.

The main goal of this paper is to propose a high-performance NoC-based-MPSoC that is scalable in terms of performance and power (NoC for Network-on-Chip). NoC utilization in a cache-based shared memory MPSoC architecture is affected by the cache-coherence protocol. When a processor modifies a cached data element that is also located in different caches, an action must be taken to prevent using non-up-dated data copies. In the literature, a significant amount of work has been proposed. Most of these protocols are variants of either the invalidation protocol or the update protocol. In the invalidation protocol, when a processor needs to modify a cache block, copies of that block at other caches must be invalidated. In the update protocol, the update is sent to all replicate blocks in other caches. These two protocols are mostly dedicated to high performance general purpose multi-processor architectures and thus do not take specific resources and energy constraints that exist in embedded systems into account. Adding that they assume that the concurrent tasks include identical and static access patterns to the shared data and identical communications patterns. As embedded systems are becoming more and more pervasive and widely used, different applications can be loaded and executed. These applications can exhibit different memory access and inter-task communication patterns. It is therefore more efficient to use protocols that are able to adapt to the application behaviors and their evolutions at run-time. The main objective of this paper is to improve the performance and the energy consumption of shared memory NoC-based MPSoC. This improvement is obtained through the utilization of an efficient dynamic hybrid update/invalidate cache coherence protocol. The proposed protocol is able to capture changes of the data access patterns at run-time and to adapt to enhance performance and reduce power consumption.

The rest of the paper is organized as follows: In the second section, we summarize the main existing works in the design of dynamic hybrid cache coherence protocol. In the third section, we present a detailed description of the proposed hybrid protocol. Finally, section four presents experimental results.

2. BACKGROUND AND PREVIOUS WORK

Existing hybrid protocols for coherency management in multi-processor architectures [1] [2], can be classified into two families:

2.1 On-line hybrid protocol

In on-line hybrid protocols the selection between invalidation and update protocols is done dynamically. The write-once protocol [3] is an example of this group. In this protocol, the first write to the cache block results in an update to the main memory and invalidations of the block in the other caches. The next write by the same processor to the same block results in a modification of the block only in the local cache and the memory is no longer updated. The Archibald scheme [4] extends the write-once protocol by allowing a greater number of updates. The competitive scheme [5] is another example of such protocols. With this scheme, a counter is associated with each cache block is preset to a value called the Competitive threshold. On an update transaction, the counter in the writer's cache is decremented. Once, the counter reaches zero the protocol switch to the invalidation mode. The competitive scheme is still suboptimal for shared migratory data. Shared migratory data are characterized by exclusive read and write access by each processor in turn, for this type of shared data the invalidation protocol gives better results.

In a recent work, [1] propose an adaptive directory based protocol that detects access pattern to shared migratory data. Although there is a variety of adaptive hybrid protocols, these protocols have some limitations. These solutions require in general gathering an important quantity of information from all the processors. Consequently, an important traffic overhead in the interconnection network is generated.

2.2 Off-line hybrid protocol

In these protocols an off-line profiling of the application is realized prior to application execution. The decision to use update or invalidation mode is realized by taking into account not only previous accesses to the block, but also information from the source code during compilation. This option offers the possibility to predict future data access patterns. In [2] a combined hardware-software strategy is presented. It uses the predictive capability of the compiler to select updating or invalidating for each write reference to a block. The adaptive sequential prefetching scheme proposed in [6] reduces the number of read misses by fetching speculatively several consecutive blocks into the cache in anticipation of future misses. The number of prefetched blocks is adapted according to the spatial locality level of the referenced block.

In [7] the authors aims at reducing the penalties associated with memory accesses by extending a hardware prefetching scheme. The idea here is to implement a memory system that combines prefetching and competitive coherency protocol while taking into account migratory sharing data.

The hybrid protocol that we present in this paper was basically proposed in [8] where some preliminary simulation results are presented. However, this paper presents a detailed description of the proposed protocol and an extension of the architecture from a centralized one-bank memory to a more realistic multi-banked shared memory. A new experimental results for different applications are presented. These applications have been selected because they depict different memory reference patterns.

3. DYNAMIC HYBRID CACHE COHERENCE PROTOCOL

3.1 System Architecture

As the number of cores sharing the memory increases, the pressure on that shared memory also increases. Increasing the number of memory banks is a clever way to decrease the access time. This is because each bank has less ports than the alternative scheme of having few but multi-ported banks. For this reason, in this paper we present a cache-coherency protocol for multi-banked MPSoC architecture.

Our protocol is based on a full bit-vector directory [9]. We also use the write-through technique where all write operations in the cache are also applied simultaneously to the corresponding block in the shared memory banks (figure 1). This method is chosen because in the context of MPSoC the write-back technique requires a relatively complex mechanism to maintain coherency.

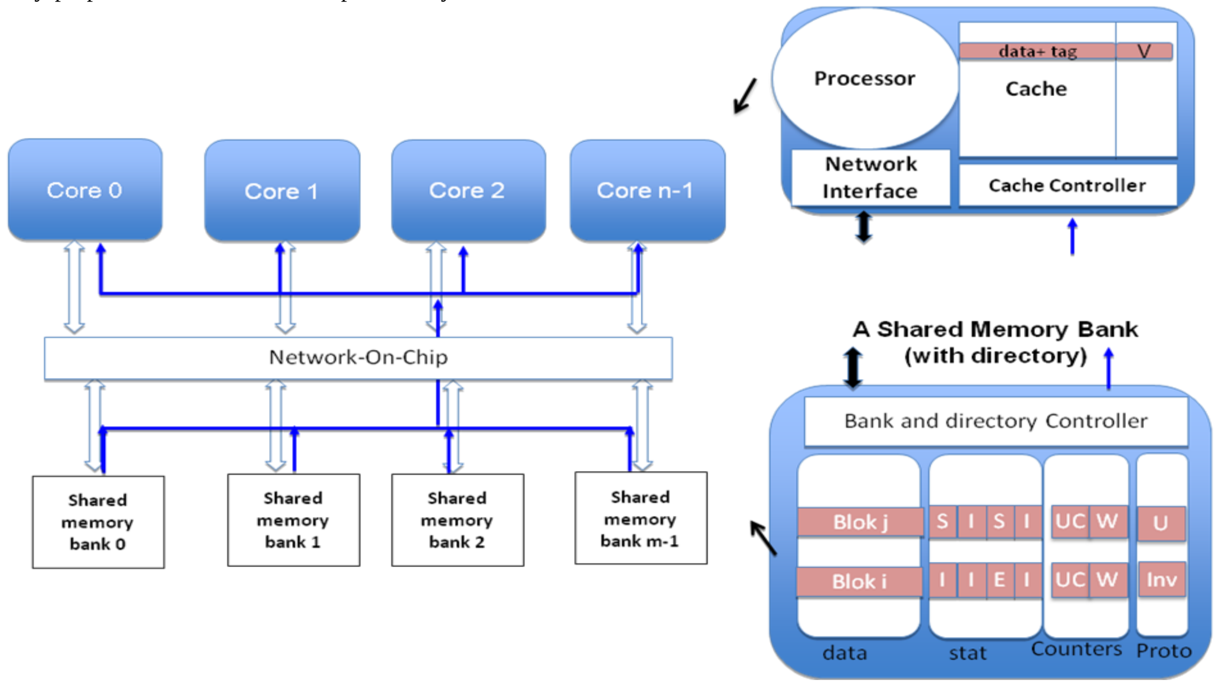


Fig 1: Architecture of our multi-banked shared memory MPSoC

We use write-no-allocate technique when miss with a write operation occurs. Since the shared memory is updated after each write cache operation, we implement a directory within the shared memory unit. A directory update operation is triggered after each write operation. Consequently, this mechanism simplifies coherency, since it does not require dedicated communications to update the directory.

To update or invalidate shared blocks in different caches, most of the existing architectures uses the shared memory controller to send an update or invalidate packet to the different caches through the NoC. However this solution tremendously increases NoC traffic. Our solution is to use a dedicated low-cost network for coherency handling. This network has the form of a unidirectional bus, that transfers update/invalidate packets from shared memory to the different caches (figure 1). This architecture is simple and reduces the overhead on the NoC.

Each entry of the directory represents the state of the corresponding block in the different caches and can have four different values (or states):

- E (Exclusive): The current value of the block is valid only in this cache and in the shared memory.
- S (Shared) : The current value of the block is valid in this cache, in other caches and in the shared memory.
- I (Invalid): The block is not valid in the corresponding cache. Either because it has not been yet loaded or it has been loaded but it was replaced by another block.
- O (invalidated by others): The block is not valid in this cache because it has been invalidated by another processor.

The states E, S, and I have the same meaning to states E, S and I in the MSI and MESI protocols [11]. In this work, we add the state called invalidated by others and noted "O" which is a special case of the Invalid state. This state plays an important role in the proposed hybrid protocol. It distinguishes blocks that have not been yet loaded and blocks that have been ejected in one side from blocks that have been loaded but invalidated by another processor in the other side. As explained in the following section, this state helps in choosing the appropriate protocol for a given block at run-time.

3.2 Hybrid protocol algorithm

The selection of the protocol is performed when a write operation is triggered by a processor and captured by the directory controller within the memory bank. To each memory block in this bank a bit, called "P", is associated to represent the protocol in use. If "P" bit is set to "INV" (0 in the implementation) then the protocol is the invalidation otherwise it is the update "U" (1 in the implementation) (figure 2). Initially, the "P" is set to "INV". When a processor executes a write operation of a memory block, if this memory block is in "E" or "S" states for the other processors, then an invalidation message will be sent to those processors. The corresponding entries of the directory become in "O" state to indicate that this block has been invalidated by another processor. So, the protocol must be switched from invalidation to update. The "P" bit goes from "INV" to "U". To return to the invalidation protocol our algorithm must detect unnecessary update operations for the considered block. The new solution that we propose, estimates at run-time and

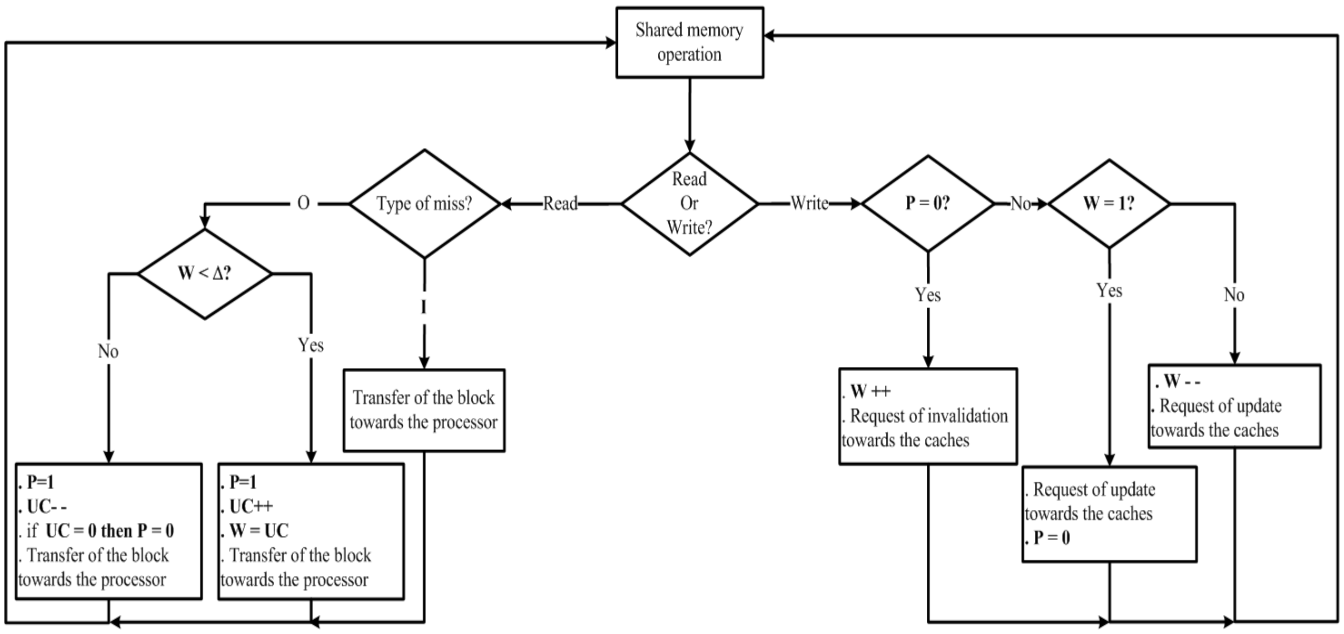

Shared memory operation
Read Or Write?
Read
Write
Type of miss?
O
I
W < Δ?
No
Yes
. P=1
. UC- -
. if UC = 0 then P = 0
. Transfer of the block towards the processor
. P=1
. UC++
. W = UC
. Transfer of the block towards the processor
Transfer of the block towards the processor
P = 0?
No
Yes
W = 1?
Yes
No
. W ++
. Request of invalidation towards the caches
. Request of update towards the caches
. P = 0
. W - -
. Request of update towards the caches

Fig 2: The algorithm of the proposed hybrid protocol

for each memory block the suitable threshold value. Indeed, this threshold increases and decreases according to data access patterns. To determine dynamically the threshold value we associate with each memory block two counters. The first one, noted "UC" for Update Counter, represents the update threshold value. "UC" is initially set to zero. The second counter, noted "W" has 2 functions :

1. In the invalidation mode, "W" counts the number of successive write operations. When a miss occurs for a read operation of the corresponding block with "O" state (i.e. the block is invalidate), "W" is tested and two situations are possible (figure 2 left part):

- "W" is relatively small (less than Δ): In this case, cache-read misses caused by the invalidation protocol are very close to each other. To avoid memory latencies, the memory block must to be updated and the protocol must switch from invalidation to update ("P" is set to "U"). The counter "UC" is incremented to indicate that during the corresponding execution phase the update protocol is the protocol to use. The counter "W" is decremented after each update operation until it reaches 0. At this moment, the protocol switches from update to invalidation ("P" is set to "INV").
- "W" is relatively important (exceeds Δ): Cache-read misses caused by invalidation operation are distant from each other. Consequently, during this period of execution, the invalidation protocol will give better performances than the update protocol. The "P" bit associated with the corresponding block is set to 1 and the "UC" counter is decremented. By this way, the period in which the update protocol will be used is reduced and consequently after a read miss the system switch quickly to invalidation. When "UC" reaches 0 then "P" is set to "INV".

2. In the update mode, the "W" register is used to count the number of allowed write operation before switching from update to invalidation. As mentioned before, after a cache-read miss, the "P" and "W" (resp.) are set to "U" and "W" (resp.). "W" is decremented after each write operation (figure 2 right part). When "W" reaches 0, the protocol switch from "U" to "INV".

Consequently, with the ESIO, the "UC" counter, that represents the update threshold, varies according to the number and distance in cycle read operations. This allows to choose dynamically which protocol (either update or invalidate) to use. In the rest of the paper, we set experimentally (Δ) at 1000. Figure 2 summarizes the proposed hybrid coherency protocol.

4. EXPERIMENTAL RESULTS

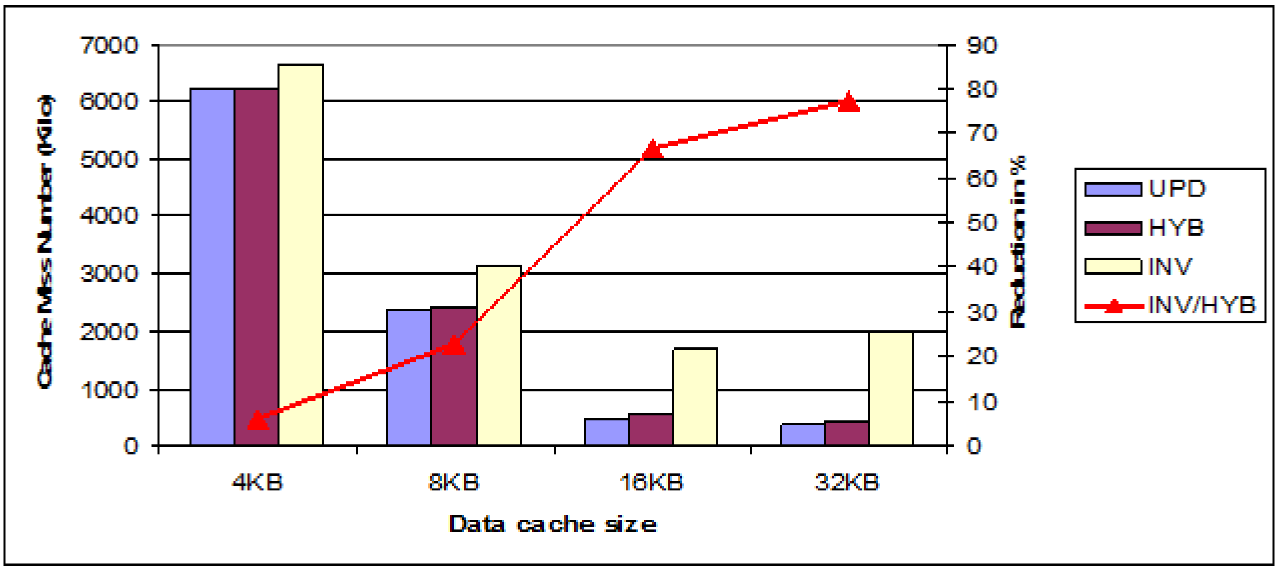
In this section we evaluate the efficiency of our proposed protocol. We performed the experiments using SoCLib platform [10]. As mentioned in the previous sections, coherency messages are taken in charge by an additional bus, called here coherency protocol bus. The experimental results that we present here have been obtained with the matrix multiplication application (MM), the Fast Fourier Transform (FFT) application and the JPEG. These results have been obtained with 4 processors MPSoC architecture and 4 memory banks. The experimental results for the matrix multiplication are shown in Figure 3. Figure 3(a) shows that the proposed hybrid protocol reduces significantly the total number of cache misses. Compared to the invalidation protocol a reduction factor of 77 % for 32 KB D-cache size is obtained. This cache-miss reduction gives a reduction in the execution time by a factor of 20% (figure 3(b)). When compared to the update protocol, we notice that the hybrid protocol does not give any reduction in cache-misses or execution time. Indeed, when using update protocol there is no cache coherency misses. By consequence, the NoC traffic is less significant. Adding to this, the unnecessary update operations are done by the additional bus for the coherency.

Figure 3(c) presents the energy consumption for the 5 main architectural components of our MPSoC : processors, caches, NoC, additional bus for coherency and the shared multi-banked memory. Concerning the additional bus, the energy is greatly reduced with the hybrid protocol compared to the update protocol. Since this protocol eliminates unnecessary updates (the reduction is up to 70%). With the invalidation protocol, the bus consumes less energy than with the two other protocols since unnecessary updates operations are avoided with this protocol. The caches energy consumption is

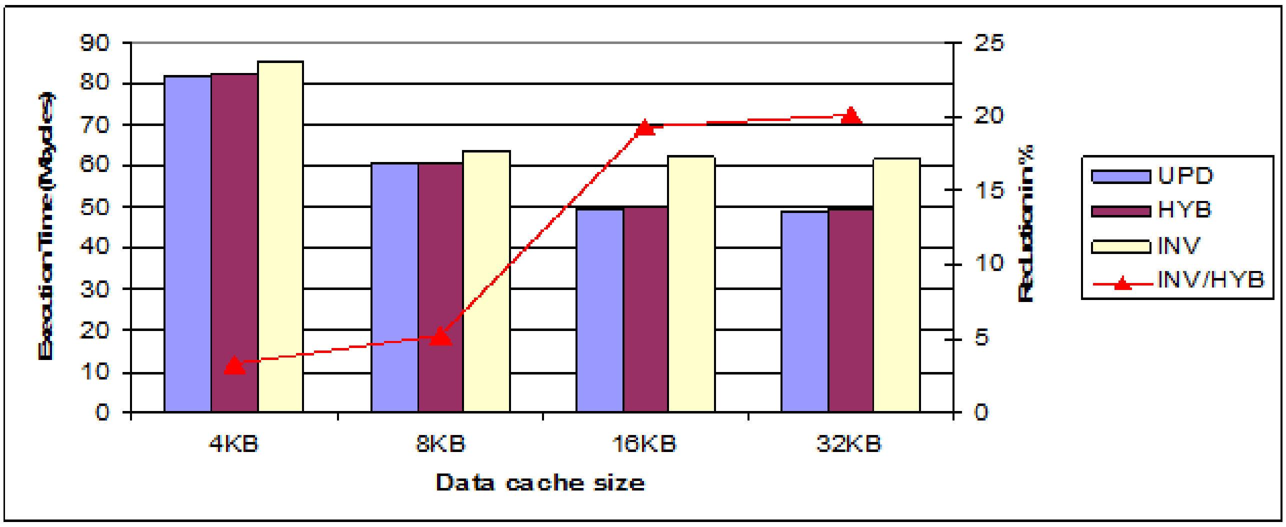
also reduced with the hybrid protocol compared to the update protocol by a factor of 4% for 32 KB. This is due to elimination of write operations of the unnecessary updates in caches. This value may appear not being significant and the reason is the presence of misses on store instructions when the hybrid protocol is used. Compared to the invalidation protocol, the reduction of energy consumption in the caches is quite significant up to 39% for large d-caches. When the cache size is important, data blocks stay for a longer time in the cache so the risk to be invalidated is more important.

The NoC energy consumption is significant with the invalidation protocol due to the traffic of cache coherency misses. This value is reduced with the hybrid protocol. The energy reduction is up to 60% for 32 KB D-cache. For the NoC power consumption, the update protocol is the less consuming since there cache coherency misses are eliminated. For energy consumption at the memory-bank level, the energy consumption is also reduced by the hybrid protocol compared to the invalidation. This reduction varies from 4,5% for 4 KB d-cache to 40% for 32 KB d-cache. Finally, the energy consumption at the processor level is also reduced by the proposed hybrid protocol compared to the invalidation up to 17% for 32KB d-cache. In summary, thanks to the hybrid protocol the total energy consumption has been reduced by a factor up to 34% for a 32 KB d-cache when compared to the invalidation protocol. Relative to the update protocol, the gain of our protocol is less significant up to 5.5% for a 32 KB d-cache. This difference is due to the fact that the unnecessary updates operation performed in the update protocol are handled by the shared bus. This bus consumes less energy than the NoC.

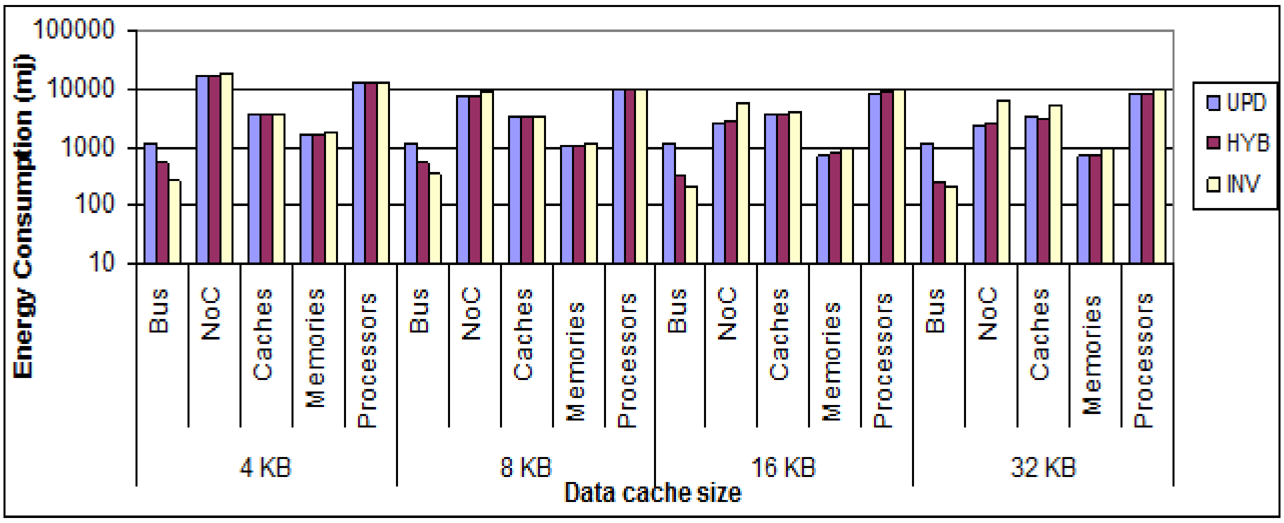
To simplify the presentation of obtained results, we used a table which includes these different results (table 1) for a 32 KB d-cache.



(a) Total number of cache misses function of data cache size



(b) Execution time function of data cache size



(c) Energy consumption for the 5 main architectural components function of cache size

Fig 3: Performance comparison for the MM application on a 4 processor-MPSoC

Table 1: Gain of hybrid protocol compared to Invalidation and Update protocols

Application	MM		FFT		JPEG	
Protocol	INV	UPD	INV	UPD	INV	UPD
Cache misses reduction	77 %	_	36%	_	47%	_
Execution time	20 %	_	10%	_	2,5%	_
Total energy consumption	34%	5,5%	8%	1%	13%	2%

As shown in table (1) for the FFT and JPEG application the hybrid protocol reduced the total miss number compared to the invalidation protocol by a factor of 47% for the JPEG application and 36 % for the FFT application. As consequence of this reduction, the execution time is also reduced up to 2.5% for the JPEG application. This value is not very significant due to the reduced total number of cache misses compared to data traffic in the NoC. Concerning to the FFT application the execution time is reduced up to 10%. Finally, the total energy consumption reduction is significant with our Proposed protocol compared to the invalidation protocol reaches up to 8% for the FFT application and up to 13 % for the JPEG application. Compared to the update protocol and thanks to the additional bus for coherency messages, only a maximum gain of 2% is obtained.

5. CONCLUSION

In this paper, we propose a new cache coherency protocol for MPSoC architectures equipped with a non-shared-bus network-on-chip (NoC). In order to avoid performance limitations when the same fixed a-priory protocol is used to maintain data coherency for this kind of MPSoC. Experimental results show that our hybrid protocol can significantly reduce both cache misses, compared to invalidation protocol, and unnecessary updates, compared to update protocol. The proposed protocol may reduce the energy consumption by factors of 34% for the MM, 13% for the JPEG and 8% for FFT. More importantly, our protocol adapts dynamically with the data access patterns of the application and does not generate unnecessary memory update operations.

6. REFERENCES

[1] D. Jhalani, D. Palsetia, “Adaptive cache coherence protocol using migratory shared data”, 2007.

[2] D. J. L. Farnaz Mounes-Toussi, “The potential of compile-time analysis to adapt the cache coherence enforcement strategy to the data sharing characteristics”, in: IEEE Transactions on Parallel and Distributed Systems, May 1995, p. 6(5) :470.

[3] J. R. Goodman, “Using cache memory to reduce processor-memory traffic”, in: Proceedings of the 10th Annual International Symposium on Computer Architecture, June 1983, pp. 124{131.

[4] J. K. Archibald, “A cache coherence approach for large multiprocessor systems”, In Proceedings of the 2nd International Conference on Supercomputing, France, July 1988, pp. pages 337{345.

[5] H. Grahn, P. Stenstrom, M. Dubois, “Implementation and evaluation of update-based cache protocols under relaxed memory consistency models”, in: Future Generation Computer Systems, June 1995, pp. 11(3) :247{ 271.

[6] F. Dahlgren, M. Dubois, P. Stenstrm, “Sequential hardware prefetching in shared-memory multiprocessors”, in :IEEE Trans. Parallel and Distributed Systems, 733-746, July 1995, pp. vol. 6, no. 7.

[7] F. Dahlgren, “ Performance evaluation and cost analysis of cache protocol extensions for shared-memory multiprocessors”, in : IEEE Transactions on Computers, october 1998, pp. I, vol. 47, no. 10.

[8] H. Chtioui, R. Ben Atitallah, S. Niar, J. L. Dekeyser, M. Abid, “A dynamic hybrid cache coherency protocol for shared-memory mpsoc”, in 12th Euromicro Conference On Digital System Design Architectures, Methods and Tools (DSD'09), Patras, Greece, 27-29 August, 2009.

[9] L. M. Censier, P. Feautrier, “A new solution to coherence problems in multicache systems”, in: IEEE Transactions on Computers, December 1978.

[10] SoCLib, “An integrated system-on-chip modeling and simulation platform”, 2003.technical report, cnrs,.

[11] R. Sendag, A. Yilmazer, J. J. Yi, A. K. Uht, “Quantifying and reducing the effects of wrong-path memory references in cache-coherent multi-processor systems”, in : Parallel and Distributed Processing Symposium, April 2006