2009 12th Euromicro Conference on Digital System Design / Architectures, Methods and Tools

A Dynamic Hybrid Cache Coherency Protocol for Shared-Memory MPSoC

Hajer Chtioui[1,2,3], Rabie Ben Atitallah[2,3], Smail Niar[2,3], Jean-Luc Dekeyser[2,4], Mohamed Abid[1]
[1] CES Laboratory, University of Sfax, Tunisia
[2] Univ Lille Nord de France, F-59000 Lille, France
[3] UVHC, LAMIH, F-59313 Valenciennes, France
[4]INRIA, F-59650 Villeneuve d'Ascq, France

Abstract—
In Multi-Processor System-on-Chip (MPSoC) architectures equipped with shared-memory, caches have significant impact on performance and energy consumption. Indeed, if the executed application depicts a high degree of reference locality, caches may reduce the amount of shared-memory accesses and data transfers on the interconnection network. Hence, execution time and energy consumption can be greatly optimized. However, caches in MPSoC architectures put forward the data coherency problem. In this context, most of the existing solutions are based either on data invalidation or data update protocols. These protocols do not consider the change in the application behavior. This paper presents a new hybrid cache-coherency protocol that is able to dynamically adapt its functioning mode according to the application needs. An original architecture which facilitates this protocol's implementation in Network-On-Chip based MPSoC architectures is also proposed. Performances, in terms of speed up factor and energy reduction gain of the proposed protocol, have been evaluated using a Cycle Accurate Bit Accurate (CABA) simulation platform. Experimental results in comparison with other existing solutions show that this protocol may give significant reductions in execution time and energy consumption can be achieved.

Index Terms—**Shared-memory, MPSoC, cache coherence, performance evaluation, energy consumption.**

I. INTRODUCTION

Multi-Processor System-on-Chip (MPSoC) architectures are becoming an incontrovertible solution for embedded systems designed for applications that require intensive parallel computations. In this paper, we focus on MPSoC equipped with a centralized shared memory and a non-shared-bus network-on-chip (NoC). These architectures are very attractive as they facilitate both the development of parallel applications, due to their programming model, and the possibility to integrate on-chip a large number of processors. However, the important disparity in term of speed between processor and memory is especially important in shared memory MPSoC. In this situation, caches may represent an interesting mechanism aiming at reducing both memory access latencies and energy consumption.

Nevertheless, the problem of data coherence emerges. In this way, when a processor modifies a cached data element that is also located in another processor's cache, a protocol must be used to maintain data coherency. Generally speaking, two families of protocols are used. The invalidation protocol consists in sending invalidation messages to sharer caches that contain the modified block. At the opposite, in the update protocol the addressed of the modified data and its value is send to all other sharers. Several studies [1] have demonstrated that the use of a unique protocol (invalidation or update), does not take into account the patterns of data accesses realized by processors. Hence, the necessity for dynamic hybrid cache coherence protocol, which takes advantage of the two protocols and that, adapts the way in which the data is used. As the new protocol can be changed during the execution of the application, it can be considered as dynamic. Various dynamic hybrid protocols have been proposed and implemented in the past [2] [3] [4] [5] [6]. Although these protocols require an important quantity of hardware and software resources and are very energy-consuming, they achieve a limited improvement of overall performance compared to a single protocol. This is mainly due to the inadequacy between the applications requirements and the architecture.

The main objective of this paper is to improve the performance and the energy consumption of shared memory NoC-based MPSoC by integrating an efficient dynamic hybrid update/invalidate cache coherence protocol. It is based on a hardware solution that uses an original architecture which facilitates its implementation. Furthermore, the proposed protocol is able to capture changes of the data access patterns at run-time and automatically to switch to a new more efficient protocol when necessary.

This 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 detail the key points that have been taken into account in the design of our cache coherency protocol. The environment of simulation that has been used is presented in the last section. This section presents also the experimental results that demonstrate the benefit of the new protocol and its supporting architecture

II. STATE OF THE ART

The limited performances due to the use of a unique protocol to maintain coherency of data with different access patterns in NoC-based MPSoC, has caused the emergence

978-0-7695-3782-5/09 $25.00 © 2009 IEEE
DOI 10.1109/DSD.2009.220

of several dynamic hybrid protocols. These protocols can be gathered into two families depending on the moment at which the decision to choose the appropriate protocol is make: “prior to application execution” or “during the execution of an application”.

In the first family of protocol, called “on-line hybrid” protocols, the selection (invalidation or update) is based on previous accesses to the cache block. In the *write-once protocol* [7] is an example of this group. The first write to the cache block in the write once protocol, results in an update to the main memory and an invalidation of the block in the other caches. The next write by the same processor to same block results in a modification of the block only in the local cache and the memory is no longer updated. The *Archibald scheme* [2] [3] extends the write-once protocol by allowing a greater number of updates. The *competitive scheme* is another example of protocol of this family. It is widely used with *snoopy protocols* [4] and with *directory protocols* [5]. It associates with each block a counter. It is the update protocol that is chosen when the block is referenced for the first time and the counter in incremented with each write access. Once it reaches a given threshold, the protocol switch to the invalidation mode. It is interesting to notice here that having the same threshold for all the blocks is not relevant. To solve this problem, Anderson and Karlin [8] proposed the using of different thresholds. This threshold is calculated according to the number of consecutive writes realized by the same processor. Despite of the improvements, this approach is still suboptimal for migratory data, i.e., data that is read, modified, and written by many processors during the same period. Grahn and Stenstrom [9] proposed a mechanism to detect dynamically migratory data. As this solution requires gathering information from all the processors; it generates an important traffic overhead in the interconnection network [10].

The second family of protocols realizes an “off-line” profiling of the application. In theses protocols, the decision to use update or the invalidation mode is realized not only on previous accesses to the block, but also by extracting information from the source code, during code compilation. In [6] a combined hardware-software strategy is used. More precisely, the predictive capability of the compiler to select either update or invalidate mode for each memory write access is exploited.

The cache coherency protocol that is proposed in the next section has interesting features compared to the competitive scheme [4] [5]. It requires neither complex traffic communications [9] nor a sophisticated network [11] [12]. Moreover, its implementation uses a reduced resource overhead, as only few flags and counters are needed.

III. DYNAMIC HYBRID CACHE COHERENCE PROTOCOL

The proposed cache coherence protocol is based on a full bit-vector directory [13]. This directory (d) is shared among all the processors and maintains information about the data stored in caches. It is implemented generally as a matrix of m lines and p columns, where m is the number of memory blocks and p the number of processor in the MPSoC. If processor j ($0 \leq j \leq p$) holds a copy of block i ($0 \leq i \leq m$), the element d[i, j] is set. Moreover, when a processor executes a store operation into a shared data, it must first examine the directory and send update or invalidate messages to processors sharing this data. In the first part of this section, the interaction between the directory and the shared memory MPSoC architecture is presented. Then, in the second part, we give a detailed description of the hybrid solution that we propose.

To have efficient cache architecture either in mono or in multiprocessor configuration, several cache parameters must be selected. Among these parameters there is the write policy. Here we used the write-through technique. In this mechanism, all write operations in the cache are also applied simultaneously in the centralized shared memory. In the context of MPSoC, the write-back technique, where data is updated only when the block is ejected, requires a more complex mechanism to maintain coherence between the cache and the shared memory.

To exploit the fact that the shared memory is updated after each write cache operation, we implement a centralized directory within the shared memory unit. Consequently, when a processor sends a write packet through the NoC to this unit, the update operation of the directory is triggered. This mechanism simplifies the coherency maintain, since it does not require dedicated packets to update the directory.

To update or invalidate shared blocks within the different caches, in the context of non-shared-bus based multiprocessor architectures, several solutions have been explored. The naive solution consists in sending an update/invalidate packet by the shared memory controller to the different caches through the NoC. This solution is not interesting because the NoC becomes the bottleneck of the system and this approach may lead to performance reduction.

Our proposition is to add a unidirectional bus, that transfers update/invalidate packets from shared memory to caches. This architecture is simple and reduces the overhead on the NoC. Figure 1 shows the architecture of the overall system.

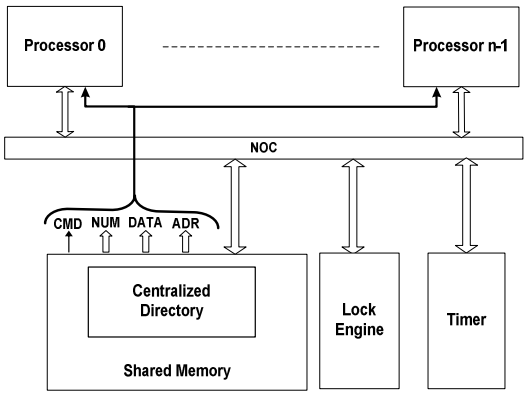


Figure 1: Architecture of the overall system

ADR and DATA correspond respectively to the address of the block to be updated or invalidated and the corresponding value in case of update protocol. Finally, NUM is used to identify the processor number that has caused the write operation. This information is needed to invalidate or update all the caches except that cache with number NUM. Each entry of the directory represents the state of a block in the different caches and may have four different values (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. It has not been yet loaded or has been but replaced by another block.
- O (invalidated by other): 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 [14]. In this work we add a new state called invalidated by the other "O" which is a special case of the Invalid state. This new state plays an important role in the proposed hybrid protocol. It distinguishes blocks that has not been yet loaded or ejected from blocks that have been loaded but invalidated by another processor. As explained in the following sections, this state helps in choosing the appropriate protocol for a given memory block in a given application phase. Our four state (ESIO) protocol uses two bits per block. The approximations that we realized with CACTI [15] demonstrate that only 3% of the area of the shared memory is used by the directory.

Figure 2 shows the finite state machine (FSM) that controls each directory entry. In this figure, we use the following notation:

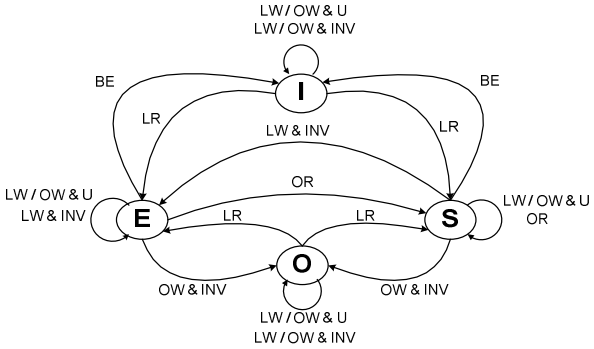
- OR (Other Read): Read by a remote processor.
- LR (Local Read): Read by the local processor.
- OW (Other Write): Write by a remote processor.
- LW (Local Write): Write by the local processor.
- BE: Block Ejection due to replacement.
- INV: Invalidation operation.
- U: Update operation.

In this figure, the notation "Ox/Ly & z" on the arcs corresponds to an operation of type x by a remote processor or an operation of type y by the local processor and the protocol in use is z. Here, x and y may be either read (R) or write (W) and z is either update (U) or Invalidate (Inv).

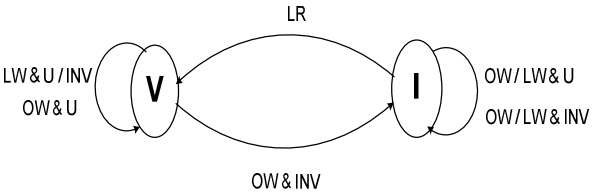
IV. HYBRID PROTOCOL IMPLEMENTATION

In this subsection, we discuss the key points related to the design of the proposed hybrid protocol. We present the criteria that must be taken into account in choosing the appropriate protocol for a given block. These criteria are determined dynamically during the execution.

Figure 3 provides an overview of the hybrid protocol algorithm that is presented in detail in the following sections.



a) FSM of the hybrid protocol in the directory



b) FSM of the hybrid protocol in the cache

Figure 2: FSM of the hybrid protocol

When ESIO is used, the selection of the protocol is realized when a write operation is triggered by a processor and captured by the shared memory controller. For this purpose a bit-vector, called "P", is used to represent the protocol to use. If "P" is 0 then the protocol is the invalidation otherwise it is the update. Initially, the protocol to use is the invalidation. The "O" state for a block indicates that this block has been invalidated by another processor. So, the protocol must be switched from invalidation to update. Detecting unnecessary update operations of a block is required to return to the invalidation protocol.

The detection of unnecessary update operations is done by controlling read operations. Usually, in existing protocols, several messages have to be broadcasted to the different processors, thus increase data traffic in the NoC and consequently affecting the energy consumption. To avoid this situation, several solutions have been proposed to determine a threshold of update operations which must be achieved before returning to the invalidation. Nevertheless almost all existing solutions do not take into account the dynamic behavior of the program, and thus are not able to adopt the protocol to the access pattern changes.

In this work we propose a new method that estimates at run-time and for each memory block the suitable threshold value. Recall this threshold represents the number of update operations to realize before switching to the invalidation mode. This threshold increases and decreases according to variation in the data access patterns.

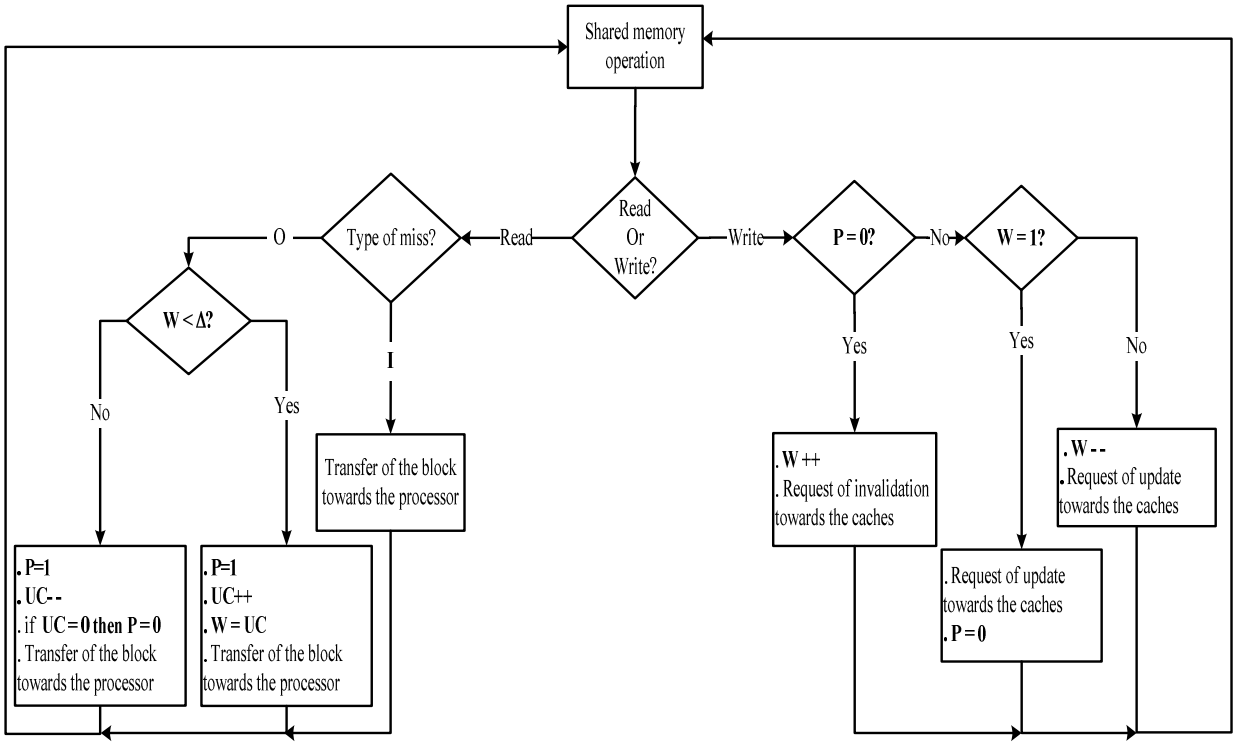


Figure 3: Algorithm of the hybrid protocol

First, we associate with each memory block a counter that is initially set to zero and represents the update threshold value. This counter is noted "UC" for *Update Counter*. To determine dynamically the threshold value, we associate with each memory block a second counter, noted "W". W is incremented after each invalidation operation of the memory block. If there is a read operation of the corresponding block with "O" state, "W" is tested and two situations are possible:

W is relatively small (less than **Δ**): In this case, cache misses caused by the invalidation protocol are very close to each other, so the memory block needs to be updated and the protocol must switch to update (P set to 1) and "UC" is incremented because at this period of run-time update protocol is the best. Then "W" takes 0 because is used also as an intermediate counter which takes initially the value of "UC", it decreases after each update operation until reaches 0, then the protocol must switch to invalidation ("P" takes 0). Consequently "W" allows to not erasing the old value of "UC" because it will be used after.

W is relatively important (exceeds Δ), cache misses caused by invalidation operation are distant from each other. Consequently, during this period of time, the invalidation protocol may give better performance than the update protocol. The P bit associated with the corresponding block is set to 1 and the "UC" counter is decremented. When "UC" reaches 0 then P is set to 0.

Consequently, with the ESIO, the "UC" counter, that represents the update threshold, varies according to the intensity of read operations in a certain period of time. This allows to choose dynamically which protocol (either update or invalidate) to use. In the rest of the paper, we set Δ experimentally at 1000 cycles.

V. Simulation Framework and Benchmarks

In this section we present the framework and the benchmarks used during the experiments. We also detail the results that we obtained. We choose to use the SoCLib platform [16], to simulate and to evaluate the proposed hybrid protocol. SoCLib is a library of reusable hardware components which makes it possible to model and to simulate MPSoC architectures. During the experiments, the NoC that has been used to connect the processors to the shared memory is a crossbar (figure 4). However the proposed protocol can be used for other NoC such as multi-stage interconnection network (MIN), hierarchical bus, etc.

The experimental results that we present here have been obtained with two benchmarks: the Fast Fourier Transform (FFT) application and the matrix multiplication (MM). These two applications are parallelized onto several processors. However, parallel programs exhibit different shared memory access patterns. This feature is one of the key factors affecting the performance of cache coherence protocol; therefore, it is important to study it influence on our protocol effectiveness.

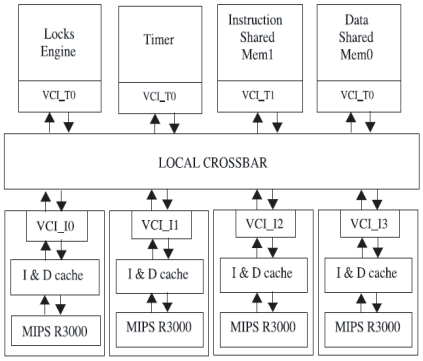


Figure 4: Architecture of SoCLib platform

We use the classification of shared blocks between the processors that has been proposed by Gupta and Weber [17] [18]. Specifically, in our work shared blocks between the processors belong to 2 sets: *Mostly-read* shared blocks and *frequently read-written* shared blocks. Mostly-read shared blocks refer to blocks of data that is read more often than it is written. This type of sharing is for instance present in pipelined or producer/consumer parallel tasks. At the opposite frequently read-written shared blocks correspond to blocks of data elements that are read and written several times. In this case, all the processors cooperate in the calculation of the final result. Domain decomposition and task decomposition are examples of parallelization approach leading to this type of blocks sharing.

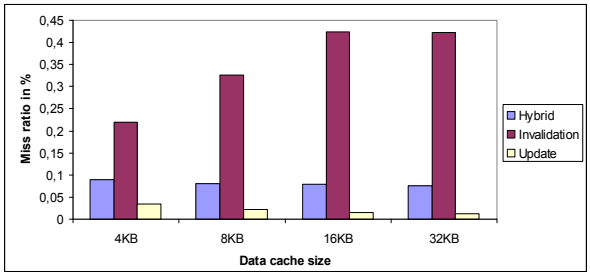
Based on this classification, we write 3 versions of FFT application. In the first version, the pipeline model has been used and consequently shared data are all of Mostly-read type. In the second version, task decomposition has been used and thus all the blocks of date are of type frequently read-written. Finally, in the third version, for some tasks we used the producer/consumer model while for some others, task decomposition has been used. Consequently this version contains both mostly-read and frequently read-written shared data and is called *mixed FFT* version. We present for each FFT version, a performance comparison between three directory-based protocols: hybrid, invalidation and update protocols. This comparison is in terms of miss ratio, execution time and energy consumption. This MPSoC platform incorporates consumption models for each type of components (processor, memory, interconnect network, cache, etc.). These power consumption models are detailed in [19].

VI. Experimental Results

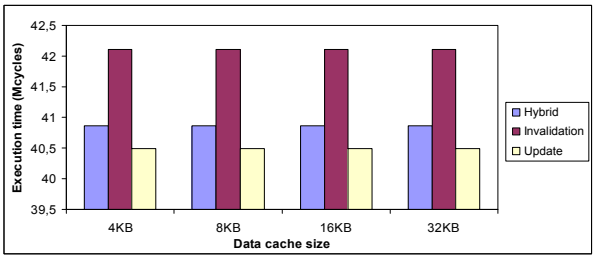
1. *Performance comparison Hybrid vs. Update vs. Invalidation*

1.1. FFT with mixed shared data

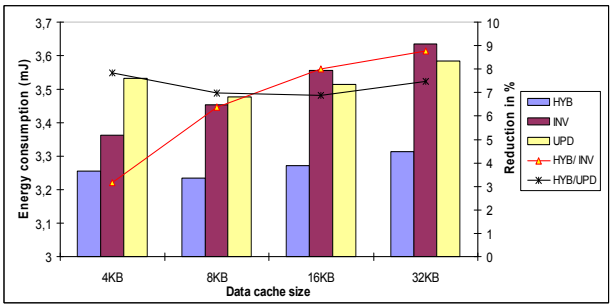
Figure 5 gives the experimental results obtained with the first parallelized version of the FFT. Figure 5.a proves that the proposed hybrid protocol reduced the miss ratio compared to the invalidation protocol. The improvement rate varies from 59% for a 4KB D-cache towards 82% for 32 KB D-cache. This improvement is due to the reduction in the number of cache misses caused by invalidating frequently read-written shared data. The reduction of the miss ratio involves on the one hand the reduction of the execution time in cycles (figure 5.b). In fact, the reduction in the number of cache misses decreases significantly the traffic in the NoC consequently the execution time is reduced. On the second hand, the energy consumption is also reduced with the hybrid protocol (figure 5.c). This reduction varies from 3% for 4 KB D-cache up to 9% for 32 KB D-cache. The improvement rate becomes more significant by increasing the cache size; this is due to the fact that the number of cache misses caused by invalidation increases by increasing the cache size.



a. Miss ratio function of data cache size



b. Execution time function of data cache size



c. Energy consumption function of data cache size

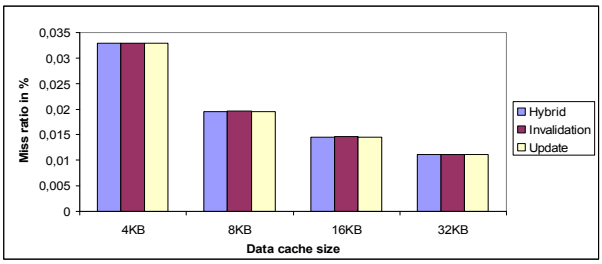
Figure 5: Performance comparison for the FFT application with mixed shared data on a 4 processor-MPSoC

The improvement made by the hybrid protocol is also compared to the update protocol. Figure 5.c shows that the energy consumption is reduced with the proposed protocol; the improvement rate varies between 8% and 7.5% for the different cache sizes. The reason behind this reduction is the reduction of unnecessary updates of mostly-read shared data with hybrid protocol. Nevertheless, this gain does not appear in term of execution time reduction in this FFT

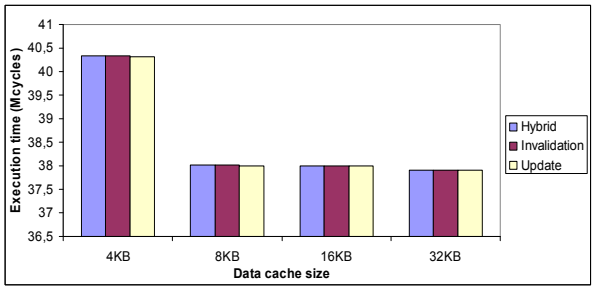
version, because the unnecessary updates are done by the bus and not by the crossbar.

1.2. FFT with mostly-read shared data

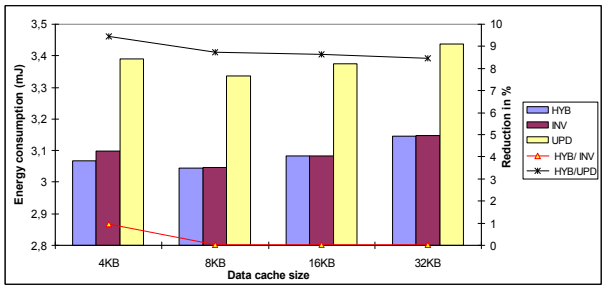
Figures 6.a and 6.b show the performances in terms of miss ratio and execution time for the second version. The obtained values are almost comparable for the three protocols (hybrid, invalidation and update). Indeed, in this version frequently read-written shared data to be updated is not present. Therefore, the energy consumption with the hybrid protocol and the invalidation protocol is the same one. On the other hand, according to figure 6.c hybrid protocol reduces the energy consumption compared to the update protocol. The improvement rate varies between 9.30% and 8.30% for the different cache sizes. This reduction is due to the fact that the hybrid protocol eliminates unnecessary updates of mostly-read shared data.



a. Miss ratio function of data cache size



b. Execution time function of data cache size



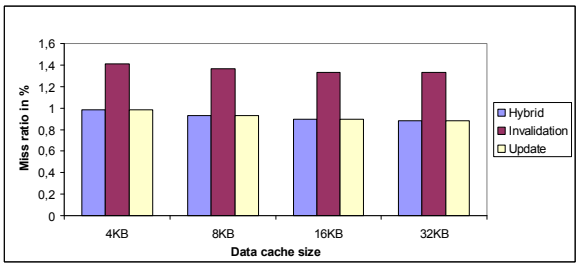
c. Energy consumption function of data cache size

Figure 6: Performance comparison for the FFT application with only mostly-read shared data on a 4 processor-MPSoC

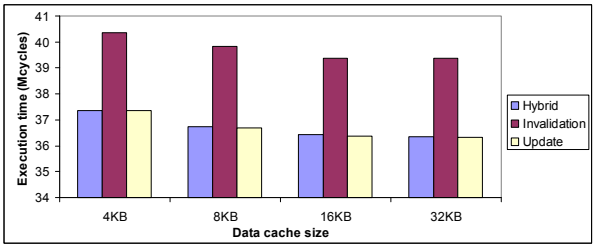
Although this version gives better results with the invalidation protocol than with the update protocol, we notice that with the hybrid protocol performances are high. This proves that the proposed hybrid protocol adapts dynamically with the data access patterns of the application and it does not make updates only if it is necessary.

1.3. FFT with frequently read-written shared data

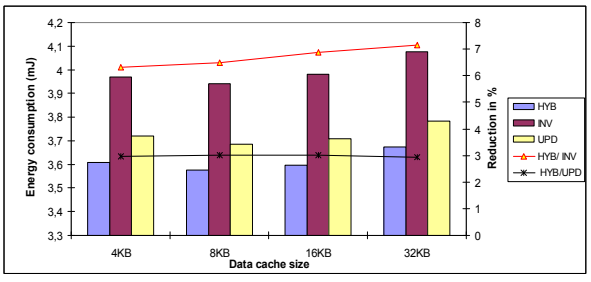
As indicated in figure 7, although this FFT version functions better with the update protocol, the results illustrated by figure 7 prove that performances with the hybrid protocol are almost similar with the update protocol. Nevertheless, the energy consumption with our hybrid protocol is lower than with the update protocol. The improvement rate is 3% (figure 7.c). This is due to the elimination of a certain number of unnecessary updates. Therefore, our new protocol realizes block invalidation only if it is necessary. So, it is able to adapt dynamically with the data access patterns of the application. This is also proven by figure 7.a which indicates that the miss ratio is improved by the hybrid protocol compared to the invalidation protocol. The improvement rate varies between 30% for a 4 KB D-cache up to 33.50% for 32 KB D-cache. In the same time, execution time is reduced by up to 8% (figure 7.b) and energy consumption is reduced by up to 7% (figure 7.c).



a. Miss ratio function of data cache size



b. Execution time function of data cache size



c. Energy consumption function of data cache size

Figure 7: Performance comparison for the FFT application using only frequently read-written shared data on a 4 processor-MPSoC

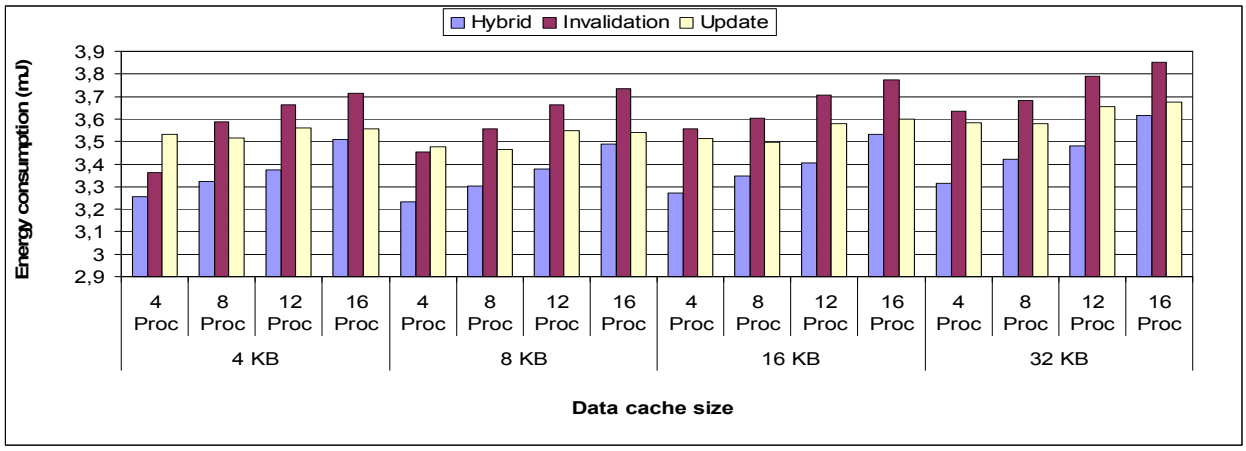


Figure 8: Energy consumption according to the data cache size and the number of processors for hybrid, invalidation and update protocols. Here we use the FFT application with mixed shared data

This section demonstrated that for a mixed shared data application the proposed hybrid protocol can reduce both cache misses and unnecessary update transactions. Moreover in extreme situations, i.e. when data are all of mostly-read type or all of frequently read-written shared data application, the protocol adapts its functioning mode at run-time.

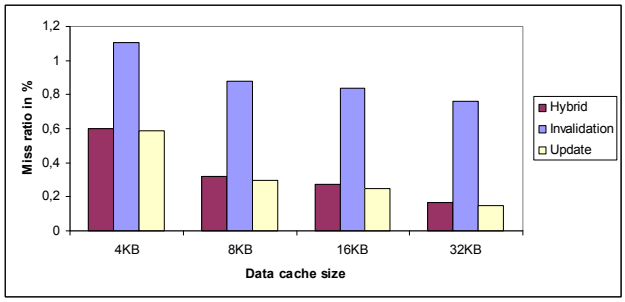
2. Performances comparison with increasing number of processors on FFT and matrix multiplication

To determine the effect of increasing the number of processors on the proposed protocol performances we parallelized the FFT application on 8, 12 and 16 processors. In this section, we compare the energy consumption for the hybrid, invalidation and update protocols. The results are given in figure 8 shows that, by increasing the number of processors, energy consumption increases for all the protocols. In addition, our hybrid protocol performs well. However the improvement is more interesting compared to the invalidation protocol than to the update protocol. This is due to the fact that by increasing the number of processors the number of cache misses also increases.

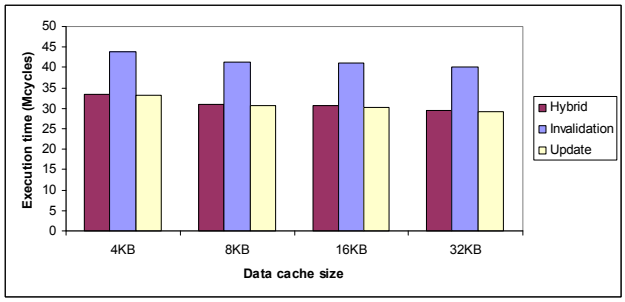
The results obtained with the parallelized version of matrix multiplication (MM) application on four processors are shown in figure 9. The upper part of this figure (figure 9.a) proves that the proposed hybrid protocol reduces the miss ratio compared to the invalidation protocol by up 80.6%. The reduction of the miss ratio allows a reduction of both the execution time (figure 9.b) and the energy consumption (figure 9.c). A gain of up 18.5% is obtained for the energy consumption. The improvement obtained by the hybrid protocol is also compared to the update protocol for the MM application. Figure 9.c indicates that the energy consumption is also reduced with the proposed protocol by up to 3.5% for 32 KB cache size.

3. Overhead reduction for the hybrid protocol

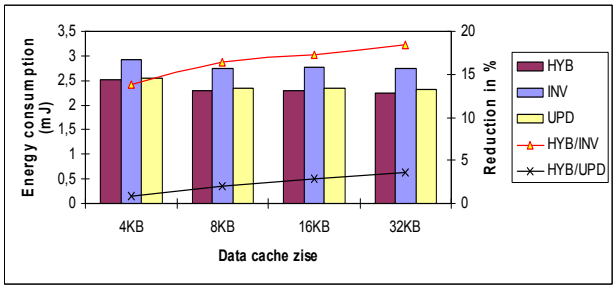
Using the CACTI [15] models, we measures the overhead of our protocol in terms of required area. Results indicate that 25% of the shared memory area is used by the proposed protocol. This overhead is quite important.



a. Miss ratio according to the data cache size



b. Execution time according to the data cache



c. Energy consumption according to the data cache size

Figure 9: Performance comparison for the matrix multiplication application on four processors

To reduce this cost simulations with MM application have been performed to measure the performance of the hybrid protocol when the number of counters within the shared

memory is reduced. In other words, instead of associating the "UC" and "W" counters with one block, these 2 counters will correspond to *b* consecutive blocks. In our experiment, *b* may have the values: 2, 4, 8, 16 and 32. Due to lack of space, we give here only the experimental result for the energy consumption gain (figure 10).

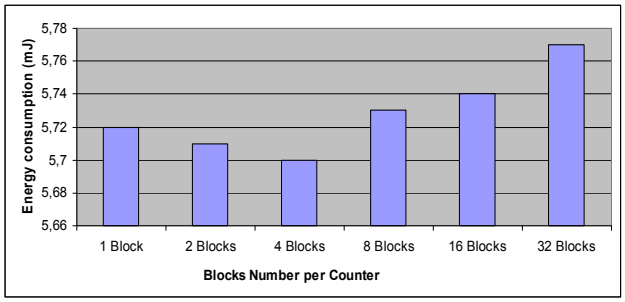


Figure 10: Energy consumption according to the number of blocks associated to each pair (W and UC) of counter for the MM application with four processors

This figure shows that in addition to area gain, there is a improvement in energy consumption when the "UC" and "W" counters are associated with 2 or 4 memory blocks. The reason behind this gain is the spatial locality, i.e. memory accesses to memory blocks in the same memory area have the same patterns and thus need the same protocol. By associating a counter for each 4 neighbor blocks the area of the proposed protocol is only 6% of the shared memory.

VII. CONCLUSION

In order to solve performance limits due to the use of unique protocol to maintain data coherency in MPSoC equipped with a non-shared-bus network-on-chip (NoC), we have designed a new dynamic hybrid protocol. The ESIO is based on a completely hardware solution and using a full bit-vector directory. We have evaluated this protocol using the SoCLib MPSoC simulation platform with two parallel applications; FFT and MM. The 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 9% for the FFT and 18.5% for the MM. More importantly, our protocol is able to adapt automatically to data access patterns at run-time.

REFERENCES

[1] H. Chtioui, R. Ben Atitallah, S. Niar, M. Abid, J. L. Dekeyser, Gestion de la cohérence des caches dans les architectures MPSoC utilisant des NoC complexes, Symposium en Architecture de machines (SympA '2008) February 2008.

[2] J. Archibald, J. L. Baer, Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model. ACM Transactions on Computer Systems, Nov. 1986.

[3] J. K. Archibald, A Cache Coherence Approach for Large Multiprocessor Systems, 2nd International Conference on Supercomputing, 1988.

[4] A .R. Karlin, M S. Manasse, L Rudolph, D. Sleator, Competitive Snoopy Caching, 27th Annual Symposium on Foundations of Computer Science, 1986.

[5] H. Grahn, P. Stenstrom, and M. Dubois, Implementation and Evaluation of Update-Based Cache Protocols Under Relaxed Memory Consistency Models, Future Generation Computer Systems, June 1995.

[6] F. M. Toussi and D. J. Lilja, The Potential of Compile-Time Analysis to Adapt the Cache Coherence Enforcement Strategy to the Data Sharing Characteristics, IEEE Transactions on Parallel and Distributed Systems, May 1995.

[7] J. R. Goodman, Using Cache Memory to Reduce Processor-Memory Traffic. 10th Annual International Symposium on Computer Architecture, June 1983.

[8] C. Anderson, A. R. Karlin, Two Adaptive Hybrid Cache Coherency Protocols, 2nd IEEE Symposium on High-Performance Computer Architecture, 1996.

[9] H. Grahn and P. Stenstrom, Evaluation of a competitive-update cache coherence protocol with migratory data detection, Journal of Parallel and Distributed Computing, 1996.

[10] D. Ivosevic, S. Srbljic, and V. Sruk, Time Domain Performance Evaluation of Adaptive Hybrid Cache Coherence Protocols, 10th Mediterranean Electrotechnical Conference, 2000.

[11] E. Bolotin, Z. Guz, I. Cidon, R. Ginosar and A. Kolodny, The Power of Priority: NoC based Distributed Cache Coherency, First International Symposium on Networks-on-Chip, May 2007.

[12] N. Eisley, L. S. Peh, L. Shang, In-Network Cache Coherence, International Symposium on Microarchitecture, 2006.

[13] L. M. Censier and P. Feautrier, A New Solution to Coherence Problems in Multicache Systems, IEEE Transactions on Computers, December 1978.

[14] R. Sendag, A. Yilmazer, J. J. Yi, and A. K. Uht, Quantifying and Reducing the Effects of Wrong-Path Memory References in Cache-Coherent Multiprocessor Systems, Parallel and Distributed Processing Symposium, April 2006.

[15] CACTI home page http://research.compaq.com/wrl/people/jouppi/CACTI.

[16] SoCLib project: An integrated system-on-chip modeling and simulation platform, 2003.Technical report, CNRS,

[17] W-D. Weber and A. Gupta, Analysis of Cache Invalidation Patterns in Multiprocessors, Third International Conference on Architectural Support for Programming Languages and Operating Systems, 1989.

[18] A. Gupta and W-D. Weber, Cache Invalidation Patterns in Shared-Memory Multiprocessors, IEEE Transactions on Computers, 1992.

[19] R. Ben Atitallah, S. Niar, A. Greiner, S. Meftali, and J. L. Dekeyser, Estimating energy consumption for an MPSoC architectural exploration, Architecture of Computing Systems, March 2006.