14th European Signal Processing Conference (EUSIPCO 2006), Florence, Italy, September 4-8, 2006, copyright by EURASIP

AUTOMATIC DSP CACHE MEMORY MANAGEMENT AND FAST PROTOTYPING FOR MULTIPROCESSOR IMAGE APPLICATIONS

F URBAN[1,2], *M RAULET*[2], *JF NEZAN*[2], *O DEFORGES*[2]

(1) THOMSON RD FRANCE,
Video Compression Lab
1 av. de belle fontaine, CS 17616
35576 Cesson Sévigné CEDEX, France
fabrice.urban@thomson.net

(2) IETR/Image group Lab
UMR CNRS 6164/INSA
20 av. des Buttes de Coësmes
35043 RENNES Cedex, France
mraulet,jnezan,odeforge@insa-rennes.fr

ABSTRACT

Image and video processing applications represent a great challenge in terms of real-time embedded systems. Programmable multicomponent architectures can provide suitable target solutions combining flexibility and computation power. On-chip memory size limitation forces the use of external memory. Its relatively small bandwidth induces performance loss which can be limited by the use of an efficient data access scheme. Cache mechanism offers a good tradeoff between efficiency and programming complexity. However in a multicomponent context data consistency has to be ensured. The aim of our work is to develop a fast and automatic prototyping methodology dedicated to deterministic signal processing applications and their multicomponent implementations. This methodology directly generates distributed executives for various platforms from a high level application description. This paper presents the improvement provided by cache memory in the design process. Cache is automatically managed in multicomponent implementations, thus reducing development time. The improvement is illustrated by image processing applications.

1. INTRODUCTION

Today's image processing applications such as video or still image codecs require not only a lot of processing power but also a large amount of memory. Specific circuits overcome these constraints; nevertheless it is not compatible with short time to market and the need for early and evolutive demonstration prototypes. An alternative can be provided by programmable software (DSP: Digital Signal Processor, RISC: Reduced Instruction Set Computer, CISC: Complex Instruction Set Computer) or programmable hardware (FPGA: Field Programmable Gate Arrays) components since they are more flexible. Hard real-time constraints are satisfied by parallel calculations on multicomponent architectures.

Embedded memory size limitation in both specific and programmable components is overcome by external memory in spite of its relatively small bandwidth compared to on-chip memory. Thus to limit the performance loss due to the external memory access bottleneck, an appropriate mechanism has to be set up to efficiently move data into on-chip memory when required.

A hand made data pre-fetch scheme can be used to temporarily store data into on-chip memory before it is needed, in a scratchpad way [1]. However this involves complex development and is not evolutive. Instead most DSPs embed a cache controller unit which allows internal memory to be used as cache to enhance external memory access.

The parallel aspect of multicomponent architectures raise problems in terms of application distribution: handmade data transfers and synchronizations quickly become very complex and result in lost time and potential deadlocks. A suitable design process solution consists of using a rapid prototyping methodology. The aim is then to go from a high level description of the application to its real-time implementation on a target architecture as automatically as possible.

This paper presents an improvement of a rapid prototyping methodology [2] suitable for signal processing systems and heterogeneous multicomponent architectures. Video applications are characterized by large data amount compared to on-chip capacities requiring the use of external memory. Its access has therefore to be enhanced. A solution for DSP devices is to use cache which involves dealing with memory conflicts known as cache coherence in multicomponent designs. The contribution of this paper is to automatically use cache to enhance external memory access and ensure data consistency for multicomponent applications in a fast prototyping process. The distributed executive containing cache management primitives is generated automatically. This automation saves development time and prevents from conflicts. It ensures processing safety and hence reduces validation tests. This is illustrated by the implementation of image processing applications over a multicomponent architecture composed of a computer and C64x DSPs.

This paper organisation is as follows: in section 2 the prototyping methodology is introduced, then cache memory and inherent issues is presented in section 3. Section 4 explains how to integrate cache in the design process. Results are given section 5, finally section 6 concludes.

2. RAPID PROTOTYPING

2.1 Context

The IETR Image Group laboratory is studying a rapid prototyping design process based on the AAA methodology (Adequation Algorithm Architecture [3]). The goal is to go from a high level description of the application to its real-time distributed implementation in a multicomponent prototyping platform. The algorithm is distributed as efficiently as possible to match the architecture in order to reduce execution time.

The prototyping is based on SynDEx tool use [4]. It is an academic system level CAD (Computer Aided Design) tool developed with INRIA Rocquencourt, France. It supports the AAA methodology for distributed real-time processing.

2.2 Automatic prototyping

The aim of automatic prototyping is to directly achieve an optimized implementation from a description of an algorithm and of an architecture. The process involves two steps: matching the algorithm and the multiprocessor platform and generating the programming code.

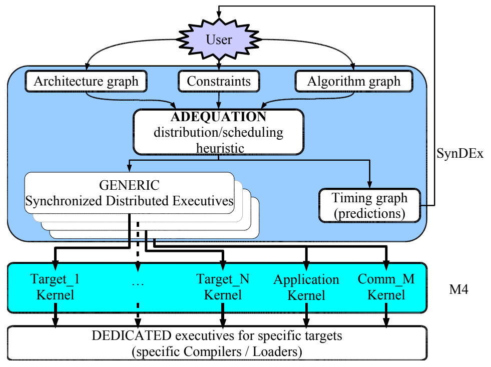
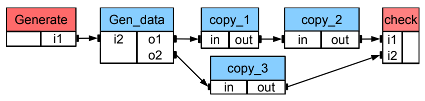
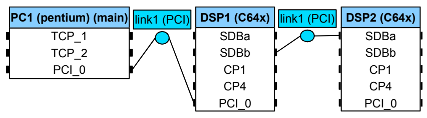


Figure 1: Automatic prototyping global graph

The first step requires the use of SynDEx (Fig. 1): the application is described with an algorithm graph (operations that the application has to execute) defined as a Data Flow Graph (DFG - Fig. 2-a), which specifies the potential parallelism, and an architecture graph (multicomponent target, i.e. a set of interconnected processors and specific integrated circuits) (Fig. 2-b), which specifies the available parallelism. The tool achieves the ”Adequation”: it maps as efficiently as possible the different parts of the algorithm onto the available processing units thanks to optimization heuristics [3]. These heuristics aim to minimize the total execution time of the algorithm running on the multicomponent architecture taking into account execution times, communication bandwidth and user constraints.



(a) Algorithm graph



(b) Architecture graph

Figure 2: SynDEx application description

SynDEx provides with both an optimized distribution (allocating parts of the algorithm on components) and scheduling (giving a total order for the operations distributed onto a component) of the algorithm on the architecture. The result is a generic SYNchronized Distributed EXecutive, which is independent of the processor target, into several source files (Fig. 1), one for each processor. A generic executive is a macro-code including memory allocations, a calculation and communication schedulers, and interprocessor synchronizations. In this way, the generated executives can be seen as an off-line static operating system that is suitable for setting data-driven scheduling, such as signal processing applications.

The second step is the transformation of the macro-code into specific compilable code (i.e. C code for PC and DSP, VHDL for FPGA) (fig. 1). This is done through the M4 macro processor and code generation kernels. It replaces macro-calls by their definition given in the corresponding executive kernel, which is dependent on a processor target and/or a communication medium. Previous works were about SynDEx kernels for several processors such as GPP (General Purpose Processors), Texas Intruments TMS320C6x (C62x, C64x) and Virtex FPGA families [2].

The executives are thus automatically generated. The user concentrates only on application related code as complex multiprocessor programming is handled by the tools. Effectively, memory allocations, operation scheduling and inter-processor communications and synchronisations are automatically inserted. The automation of the code generation includes formal verifications during the adequation which avoids deadlocks in the communication scheme. Moreover it ensures processing safety thanks to semaphores. Part of the tests are consequently eliminated, decreasing the development lifecycle.

2.3 Design Process

Automatic prototyping enable full rapid implementation of the application DFG on a multiprocessor platform. It allows easy architectural exploration from formal verification of the algorithm on a single PC, to the final multiprocessor implementation. Adding (or removing) a processor to the architecture is simple because most of the tasks performed by the user concern the description of an application and a compiling environment.

In the same way a modification of the algorithm (involving additionnal operations, new data dependance or simply an execution time evolution) can lead to a new distribution and scheduling. Here again this is handled by the tools.

The approach is evolutive as complex tasks (adequation, synchronization, data transfers and chronometric reports) are executed automatically.

3. CACHE MEMORY OVERVIEW

In this section we present cache memory mechanism and faced issues in a multi-DSP context.

3.1 Processor memory architecture

A simplified architecture of a common DSP is given figure 3. The CPU (Central Processing Unit) is the core. It accesses internal memory through a fast bus. A DMA (Direct Memory Access) controller manages memory transfers. Next to the processor is the external memory connected through an external interface with a slow bus. Additional peripherals such as communication links are connected to the external interface.

External memory is needed in most image processing applications caracterized by large data manipulations. Its main drawback is its relatively small bandwidth. In order to avoid a memory access bottleneck data must be efficiently fetched prior to be used. This can be done with the DMA, allowing concurrent CPU processings and memory transfers. Such

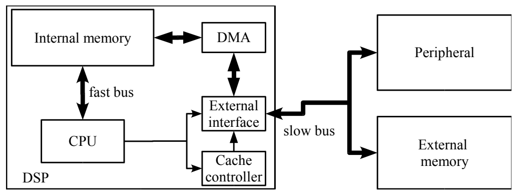


Figure 3: DSP Memory architecture

mechanism involves complex programing and is not evolutive. Indeed its scheme depends on the type of calculation, memory size and data. Instead, development time can be saved using cache memory. It offers an automatic way to enhance data access and is more suitable to be integrated in a fast prototyping process.

3.2 Cache mechanism

Most DSPs embed a cache controller improving drastically performances for external memory data processing. The internal memory is then set as cache. Some DSP (e.g. TI C64x) allow to set only a part of internal memory to be used as cache. Remaining memory can be used to store the program and small data. When the cache is enabled the CPU doesn't directly access external memory. Instead data requests are sent to the cache controller which uses the DMA to move data from external memory into the cache. The main advantage is that cache mechanism is automatic i.e. there is no need to modify the programming code.

The cache controller manages all external memory accesses from the CPU. It maps data requests to corresponding cache locations, fetch data from external memory whenever it is not already in cache. When data is removed from cache, it is either just deleted (invalidated) or written back to external memory depending on its modification.

This mechanism accelerates external memory processings. Considering an optimal case, a piece of data is fetched only once from external memory and it is reused from cache. Computation time is then drastically reduced compared to full external accesses.

3.3 Cache coherence and multiprocessor applications

With the cache memory mechanism, two copies of the data coexist (one in external memory and one in cache). When it is modified in either one of the locations, cache and external memory differ, they are said to be incoherent. A protocol must be followed to ensure no outdated data is accessed [5].

Data involving a dependance on the Data Flow Graph are taken into account by the prototyping tool. Remaining data is considered as program or contained in the stack. It should anyway be only small internal constants or temporary storage. The prototyping tool handles the allocation of the first type. These data locations are also protected with semaphores automatically generated in the executive. As a consequence, only one DSP has access to a memory location at a time. Only data visible on the DFG is considered relevant for cache use and hence incoherence risk. This is reliable because remaining data is internal and can't have a dependance with the rest of the architecture.

Automatic prototyping had been validated without cache activation. It generates reliable compilable multiprocessor executives. Nevertheless cache use brings data consistency issues that have to be resolved. We identified two cache incoherence configurations:

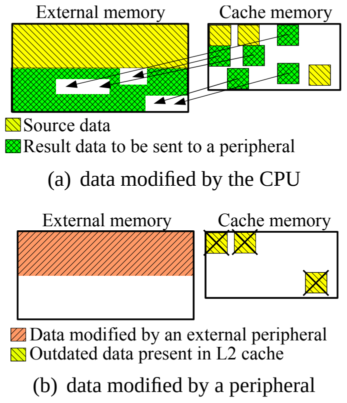

External memory
Cache memory
Source data
Result data to be sent to a peripheral
(a) data modified by the CPU
External memory
Cache memory
Data modified by an external peripheral
Outdated data present in L2 cache
(b) data modified by a peripheral

Figure 4: Cache coherence management

1. **When cached data is modified by the CPU and is accessed by an other processor**: The output data is located in cachable external memory (fig. 4-a). Cache is used to temporary store results and source data. At the end of processing, not all the output data has been writen to external memory. Consequently it has to be updated before it is accessed by a peripheral.
2. **When cachable data is modified by a peripheral**: A peripheral updated an external memory location (fig. 4-b). Corresponding outdated data must be evicted from cache to prevent the CPU to use it.

Cache memory configuration and management is usually done through libraries provided by the DSP manufacturer. The most common way to apply cache management on a DSP is the use of *"writeback"* and *"invalidate"* functions for configurations 1 and 2 respectively. Before cachable memory is accessed by a peripheral cache coherence must be ensured to prevent from data conflicts.

In multiprocessor applications designed with AAA most of the development concerning system-level operations are performed automatically. Our goal is to automatically manage cache coherence as well in order to provide cache capabilities.

4. AUTOMATIC CACHE MEMORY COHERENCE MANAGEMENT WITH AAA

The prototyping tool realizes static memory allocations and operation scheduling. It also handles interprocessor transfers and synchronizations. The knowledge off this context enables the cache coherence management to be handled automatically to maintain proper execution of the application.

4.1 Synchronized executive generation

The result of an adequation is a synchronised distributed executive. An example concerning one processor of the platform is represented with a Petri graph on figure 5.

This graph contains two schedulers (computation and communication) synchronized one another with semaphores. They are manipulated with P() and V() operations: P() is a request to a semaphore unit that generates a waiting state in a scheduler and V() is a semaphore token release.

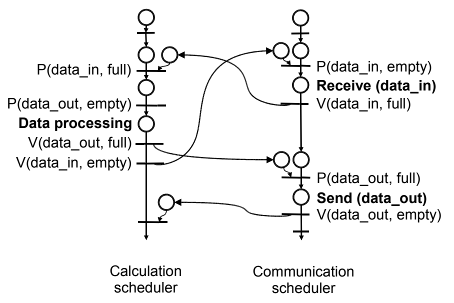


Figure 5: Executive's synchronisation

Memory locations concerned by a potential cache coherence issue are in cachable external memory area. If a location is accessed by both the CPU and the DMA then cache coherence has to be ensured because the DMA bypasses the cache controller. We consider that all interprocessor transfers are optimized and hence done via DMA which should always be the case considering the performance loss of a data polling technique. Cache coherence has then to be ensured when switching between the calculation scheduler and a communication scheduler. Consequently coherence management will be needed when unlocking the communication scheduler, which happens when issuing a V() operation from the calculation scheduler (Fig. 5). To ensure cache coherence V() operations must be coupled by proper cache management operation (*"writeback"* or *"invalidate"* request).

These suppositions are not restrictive: first if an interprocessor transfer uses a polling technique anyway an unnecessary cache management operation might be issued without any impact on operation result. The only drawback would be a slight decrease in performance. Secondly cache management macros have been developped to ensure cache coherence around operations using DMA (to enhance further memory access).

4.2 User's interaction

As the design process is automatic, only a limited knowledge of the hardware is required when prototyping an application onto a DSP. Simple macros have to be called from the application specific kernel to activate cache. Then the executive code generation automatically carries out cache memory allocation and coherence for inter-processors transfers. Cache coherence macros might be also used in the application specific kernel to automatically insert coherence operations around a function that uses DMA on a cachable memory location.

This allows a novice to prototype an application on a multi-DSP platform. The same programming C code as the one used on a PC can be compiled and executed on the DSPs. Indeed on a PC memory size is not an issue; a cache mechanism is implicitly used to efficiently access RAM.

The user can consequently safely use cache to enhance performances with only a limited knowledge of a DSP architecture. Previous work [2] allowed to easily program a multiprocessor platform. This improvement provides an automatic perfomance gain. Moreover, when external memory is used without cache, data localisation has a great impact on performance. The distribution of data between external or internal memory is crutial. With automatic cache management the impact of data localisation is reduced because every buffer in external memory can benefit of cache.

4.3 TI C64x DSP

C64x DSP from TI embed internal memory that can be used to cache external memory accesses [6]. The corresponding SynDEx code generation kernel had been updated to automatically activate and manage cache. Semaphore and memory allocation macros had been modified to use cache and ensure coherence. Results showing cache improvements are presented in the next section.

5. RESULTS

The automatic cache management had been tested and validated on a few applications involving a computer and C6416 DSPs at 1 GHz (fig. 2-b). These applications have been developed using the design process described above. The generated executive for the TI's C64x DSPs includes cache activation and management operations that have been automatically inserted during code generation process. Timing results are presented below for the LAR image compression method [7] and an MPEG-4 part. 2 video decoder developed at IETR, and a motion estimation algorithm [8].

An image compression application (LAR) had been prototyped onto the multicomponent architecture. The C code for PC was directly implemented into DSPs, thus without any algorithm optimisation. Compression - decompression timing results are given table 1 for different data locations and two image sizes. For a CIF picture (352x288) all data buffers can be contained in internal memory, giving optimal performances. Unfortunately bigger images (SD: 720x576) data do not fit in C6400's internal memory. Using cache accelerates by 10 the compression/decompression process when using external memory; it is only a 16% increase compared to optimal case. Cache is used and automatically managed, offering a good trade-off between programming complexity and data access performances.

data access	SD image	CIF image
external	800 ms	310 ms
internal	doesn't fit	26 ms
Automatic cache management	80 ms	31 ms

Table 1: Image codec (LAR) application timings

The technique had also been tested with a hierarchical motion estimation for HD (1280x720) video encoder. The video encoder runs on a PC whereas the motion estimation algorithm runs on a DSP platform. Table 2 gives timing results for the estimation of one motion field. Different methods are used: first the code is not manually optimized (only compilator optimisations) and the algorithm is benchmarked for full external access and cache memory use. Then several optimizations for C64x are performed. Using cache memory accelerates automatically by a factor of 24. A handmade optimized scheme to access external memory prevents from cache misses and flushes. It is 22% faster then cache in this example considering optimized code for DSP. However it requires complex specific programming and sufficient available internal memory.

data access	Timings
(1): full external accesses	5350 ms
(2): Automatic cache management	221 ms
(3): 2 + optimized code for DSP	100 ms
(4): 3 + handmade DMA accesses	78 ms

Table 2: Motion estimation application timings

Thirdly the automatic cache management had been applied on an MPEG-4 Part. 2 video decoder. It has been implemented by the IETR laboratory. Decoding times for I and P frames are given table 3 for a CIF (352x288) image sequence. This resolution allows all data buffers to be allocated in internal memory, which gives an optimal case for efficiency comparison purposes. Higher resolutions require the use of external memory. Here again the cache reduces drastically the execution time when using external memory: an acceleration factor of 5 for I frames and 7 for P and B frames. The loss due to the use of external memory and cache versus internal memory is 23%.

data location	I frame	P-B frame
internal memory	4.3 ms	5.5 ms
External memory without cache	22 ms	50 ms
External memory with automatic cache	4.5 ms	7 ms

Table 3: MPEG-4 decoder timings

The automatic cache management ensures data coherence. The MPEG-4 decoder represents 72 different data buffers whose consistency has to be checked. Allocating and ensuring coherence of these buffers implies lots of development time. The automatic code generation is achieved in seconds, allowing fast changes in the application.

The use of cache mecanism to enhance external memory accesses provides an acceleration of 5 to 24 for the given examples. These results are obtained automatically with the prototyping methodology. The multicomponent implementations are straitforward and cache is managed with no user interaction. A modification of either the algorithm or the platform can involve a new distribution and scheduling of the operations. This is automatically handled by the tools. As a result interprocessor communications are safe and cache coherence is ensured. A handmade access scheme using DMA enhances further external memory access, however it is not evolutive as it depends on the target, available space, computations and data.

These results confirm that cache is a good trade-off between programming complexity and calculation performances. It offers the flexibility needed in a prototyping process. The use of cache in a multicomponent application designed with SynDEx is now fully automatic. This saves development time and guaranties cache coherence.

6. CONCLUSION

We provide a way to easily use cache and automatically ensure cache coherence in a multiprocessor architecture. The design process is based on statically distributed and scheduled applications, which is well suited to deterministic signal processing.

The prototyping methodology includes functional checking, multiprocessor platform simulation and automatic distribution and scheduling of the operations onto the target architecture. Image processing applications imply large data amouts and the use of external memory. Bandwidth becomes then a bottleneck that can be reduced by an efficient data access scheme. A handmade technique can be very efficient; nevertheless it doesn't offer the flexibility needed by a fast prototyping methodology. Cache mechanism on the contrary is a good trade-off between efficiency and programming complexity. Moreover its management including activation, allocation and coherence is now automated by the prototyping tools.

Applications designed and validated with SynDEx can be directly prototyped onto a multi-DSP architecture, memory size and bandwidth issues being reduced with cache use while data coherence is automatically maintained. Thanks to the automatic cache management a novice can rapidly prototype a multiprocessor application and get good performances with only a limited knowledge of DSPs. Further specific optimisations are possible to enhance the implementation. This work extends the prototyping methodology to HD-TV applications which require a large memory size.

REFERENCES

[1] P. Francesco, P. Marchal, D. Atienza, L. Benini, F. Catthoor, J.M. Mendias, "An integrated hardware/software approach for run-time scratchpad management," in *Design Automation Conference*, 2004, pp. 238 – 243.

[2] M. Raulet, F. Urban, J.-F. Nezan, C. Moy, and O. Déforges, "SynDEx executive kernels for fast developments of applications over heterogeneous architectures," in *Proceedings of 13th European Signal Processing Conference*, Antalya, Turkey, 2005.

[3] T. Grandpierre, C. Lavarenne, and Y. Sorel, "Optimized Rapid Prototyping For Real-Time Embedded Heterogeneous multiprocessors," in *Proceedings of 7th International Workshop on Hardware/Software Co-Design, CODES'99*, Rome, Italy, May 1999.

[4] JF.Nezan, O.Deforges, M.Raulet, "Fast prototyping methodology for distribued and heterogeneous architectures: application to Mpeg-4 video tools," *Design Automation for Embedded Systems*, pp. 141–154, 2004.

[5] M. Tomasevic and V. Milutinovic, "A survey of hardware solutions for maintenance of cache coherence in shared memory multiprocessors," in *Proceeding of the Twenty-Sixth Hawaii International Conference on System Sciences*, August 1993, vol. 1, pp. 863 – 872.

[6] Texas Instruments, "TMS320C6000 DSP Cache User's guide," spru656a.

[7] M. Raulet, M. Babel, J.-F. Nezan, O. Déforges, and Y. Sorel, "Automatic coarse-grain partitioning and automatic code generation for heterogeneous architectures," in *Proceedings of IEEE Workshop on Signal Processing Systems, SiPS'03*, Seoul, Korea, August 2003.

[8] C. Stiller and J. Konrad, "Estimating Motion in Image Sequences, A tutorial on modeling and computation of 2D motion," 1999.