The MOSART Mapping Optimization for multi-core ARchiTectures *

Bernard Candaele[1], Sylvain Aguirre[1], Michel Sarlotte[1], Iraklis Anagnostopoulos[2], Sotirios Xydis[2], Alexandros Bartzas[2], Dimitris Bekiaris[2], Dimitrios Soudris[2], Zhonghai Lu[3], Xiaowen Chen[3], Jean-Michel Chabloz[3], Ahmed Hemani[3], Axel Jantsch[3], Geert Vanmeerbeeck[4], Jari Kreku[5], Kari Tiensyrja[5], Fragkiskos Ieromnimon[6], Dimitrios Kritharidis[6], Andreas Wiefrink[7], Bart Vanthournout[7], Philippe Martin[8]

[1] THALES Communications, Colombes Cedex, France
[2] Institute of Communications and Computer Systems, Athens, Greece
[3] Royal Institute of Technology-KTH, Stockholm, Sweden
[4] IMEC, Interuniversity Micro-electronics Center, Leuven, Belgium
[5] VTT Communication Platforms, Oulu, Finland
[6] INTRACOM S.A. Telecom Solutions, Peania, Greece
[7] SYNOPSYS, Leuven, Belgium
[8] ARTERIS, Guyancourt Cedex France

Abstract
MOSART project addresses two main challenges of prevailing architectures: (i) The global interconnect and memory bottleneck due to a single, globally shared memory with high access times and power consumption; (ii) The difficulties in programming heterogeneous, multi-core platforms MOSART aims to overcome these through a multi-core architecture with distributed memory organization, a Network-on-Chip (NoC) communication backbone and configurable processing cores that are scaled, optimized and customized together to achieve diverse energy, performance, cost and size requirements of different classes of applications. MOSART achieves this by: (i) Providing platform support for management of abstract data structures including middleware services and a run-time data manager for NoC based communication infrastructure; (ii) Developing tool support for parallelizing and mapping applications on the multi-core target platform and customizing the processing cores for the application.

1 Introduction and Motivation

The widening gap between power and performance requirements of applications and what is afforded by technology scaling and architectural techniques clearly points to multi-processor architectures as the solution. As an example, even the present

* This work is supported by the E.C. funded FP7-215244 MOSART Project, www.mosart-project.org

day wireless standard 802.11a requires more than 5 GIPs (IST - Project E2R) of conventional DSP processing for its physical layer. The challenge going forward is to be able to sustain several applications that are at least an order of magnitude more demanding than the 802.11a/n/m.

Memory dominates the cost, power and performance of heterogeneous multiprocessor architectures. The need for large amount of storage and a high bandwidth access to it comes from two ends. The primary need comes from the applications becoming more complex and data intensive (high resolution, higher bandwidth communication etc.). The secondary need comes from the requirement to hide the latency of accessing slower off chip memory. To comprehensively optimize both the aspects, the challenge is to treat the memory question at system level where decisions are made about how to map complex and abstract data structures to efficient distributed memory hierarchy and provide runtime support for memory management and scheduling.

To address the memory and interconnect challenges, MOSART has developed a distributed memory architecture that is tightly integrated with a Network-on-Chip (NoC) interconnect backbone. Physically and architecturally NoC is an enabling technology that addresses the memory and interconnect challenge. Such NoC based distributed architecture enables arbitrary communication pattern among applications and also significantly lowers the interconnect latency, memory latency and energy requirement for accessing data. Developing appropriate design methods and tools, we have explored within affordable time budgets, various NoC interconnection topologies and multi-layer memory structures resulting into high performance and low energy NoC architecture.

To effectively utilize the distributed architecture and make the development cycle more modular, MOSART has developed middleware services for memory management for runtime data allocation and access scheduling. This middleware provides an abstract data type library offering optimized data types to the applications running on the platform. Additionally, a run-time data allocator is in charge of the data allocation over the distributed memory of the NoC platform. Present in the middleware are APIs that interface to the data transfer services (e.g., block transfers over the communication infrastructure).

Key characteristics of the developed architecture and the methodology are flexibility, scalability and modularity. The flexibility comes from a library of system level building blocks, both functional and infra-structural. The scalability comes from the ability to logically combine resources for increased performance, storage and/or bandwidth. The modularity comes from the way the building blocks are architected and harnessed at the chip level and how the design methodology models and abstracts them.

MOSART is the first attempt, to the best of our knowledge, that proposes the usage of NoC properties to actually solve some of the most vexing problems facing the SoC architectural and design community like a) design productivity, b) computational power, c) low power, d) domination of memory in terms of power and performance, e) global interconnect latency, f) bus scalability and g) managing arbitrary concurrency. MOSART is also the first serious attempt at developing a methodology

around it to be able to deploy it for real life applications. In MOSART, we also use NoC to provide us with scalability so that we can tune and customize the computational power, the interconnect bandwidth and the storage to the needs of applications at hand.

To summarize, the technical objectives are: a) to develop a multi-core architecture with distributed memory organization, a NoC communication backbone and configurable processing cores that are scaled, optimized and customized together to achieve diverse energy, performance, cost and size requirements of different classes of applications, b) to provide platform support for management of abstract data structures including middleware services and a run-time data manager for NoC based communication infrastructure, c) to develop tool support for parallelizing and mapping applications on the multicore target platform and customizing the processing cores for the application, and d) to validate and evaluate the architecture and tool support using applications from future high data rate wireless access.

2 Project description

The MOSART project has developed a flexible modular multi-core on-chip platform architecture and associated exploration design methods and tools. The overall system level methodology by MOSART is depicted in Fig. 1. The methodology steps are a)Applications and Performance Requirements (Section 2.1), b)Parallelization and System-Level exploration (Section 2.2), c)NoC Customization (Section 2.3) and d)ASIP exploration (Section 2.4).

2.1 Applications and Performance Requirements

The credibility of the MOSART approach is demonstrated by means of illustrative applications that demonstrate a high degree of usability for the existing design base. Two such applications have been chosen for the purposes of validation/evaluation.

The first one is the implementation of a part of the cognitive radio application on the MOSART platform. Cognitive radio is a new concept, employed in order to optimize the frequency band usage. It will be integrated into the next generation of post-SDR wireless terminal. This test case has already been used to demonstrate the interest of the ASIP approach in a first step, that is followed by the implementation of porting and execution of the parallelized code on a combination of multi-core and multi- ASIP architecture.

The second is an implementation on the MOSART platform of selected parts of the PHY layer of an experimental prototype of an IEEE 802.16e based broadband wireless system. The 802.16e standard has been defined to support broadband mobile connectivity in urban environments. The standard places heavier processing requirements than the earlier fixed WIMAX standard of 802.16d, coupled with

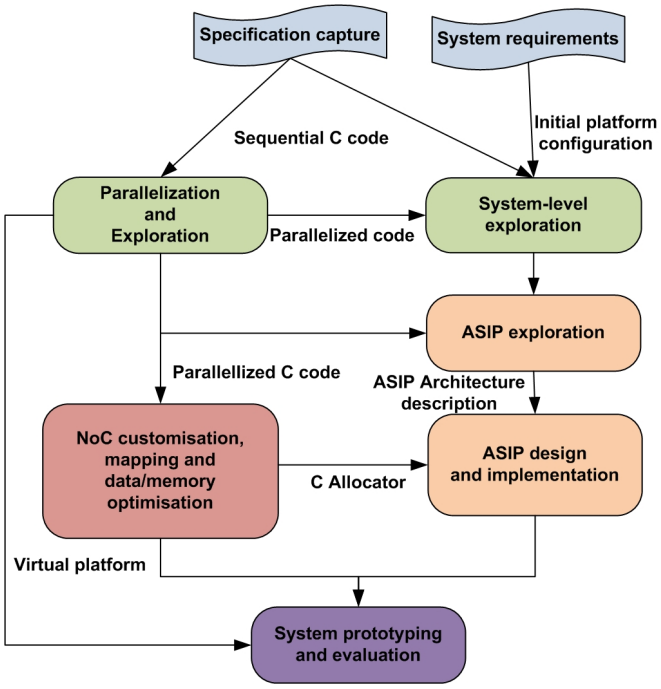

Fig. 1 MOSART framework overview

the ever-present need for low-power mobile terminals. The chosen application subset has been coded in C, and gone through the steps of parallelization. This step provides the necessary profiling information that will guide the ASIP exploration phase.

2.2 Parallelization and System-Level exploration

Extraction of parallelism from the sequential model of applications is conventionally used by algorithm developers. The MPSoC Parallelization Assist (MPA) tool [1] analyzes the application and generates parallel source code based on the directives specified by the designer. Then, MPA allows reporting performance (coarse-grain) obtained by simulating the parallelized application. The general idea of parallelization is that the designer identifies parts of the sequential code that are heavily executed and should be executed by multiple threads in parallel to improve the performance of the application. These pieces of code that will be parallelized are denoted as parallel sections (ParSec). Given the input code and the parallelization directives, the tool will generate a parallel version of the code and insert FIFOs and synchronization mechanisms where needed. Each time a flow dependency crosses a thread

boundary, the last definition has to be communicated from the producing thread to the consuming thread. This is done by inserting FIFO style communications channels into the generated partitioned code [1]. The parallelizations generated by the MPA need to be evaluated (fine grain) to find out the overall application gain in power and/or performance. To achieve this, an approach is required to evaluate system power and performance at a high-level.

The performance modeling and analysis approach is achieved with ABSOLUT [2] that is a model-based approach for system-level design. This approach takes service orientation into focus, and the execution platforms are modeled in terms of services provided (ASIPs, memories, interconnect, etc). The layered hierarchical workload models represent the computation and communication loads the applications cause on the platform when executed. The layered hierarchical platform models represent the computation and communication capacities the platform offers to the applications. The workload models are mapped onto the platform models and the resulting system model is simulated at transaction-level to obtain performance data. The approach enables performance evaluation early, exhibits light modeling effort, allows fast exploration iteration, reuses application and platform models, and provides performance results that are accurate enough for system-level exploration.

2.3 NoC Customization

MOSART has developed new technologies for future MPSoC based upon Network on Chip and distributed memory and computing cores for multimedia and wireless communications. In our McNoC, memories are distributed but shared among network nodes. An example is shown in Fig. 2. The system is composed of 16 Processor-Memory(PM) nodes interconnected via a packet-switched mesh network. A node can also be a memory node without a processor, pure logic or an interface node to off-chip memory. As shown in Fig. 2, each PM node contains a processor, for example, a LEON3, hardware modules connected to the local bus, and a local memory. The key module, which we introduce as an engine for memory and data management, is the *DME*, able to simultaneously serve various requests from the local core and the remote ones via the network. A Data Management Engine (DME) [3] has been designed and implemented to handle all on-chip memory and data management tasks for a distributed shared memory architecture. A set of data management methodologies for future McNoC platforms is proposed too. The first methodology that is developed is the abstract data type optimization (ADT). Employing this technique, the designers will be able to change the way the dynamic data of applications are stored and accessed (MTh-DMM). Also, the mapping of abstract data types to a distributed memory architecture is managed by the runtime memory management.

A novel asynchronous communication scheme (GRLS = Globally Ratiochronous-Locally Synchronous) [4] has been developed. The GRLS paradigm is based on the observation that in SoCs all on-chip clocks are normally derived from the same

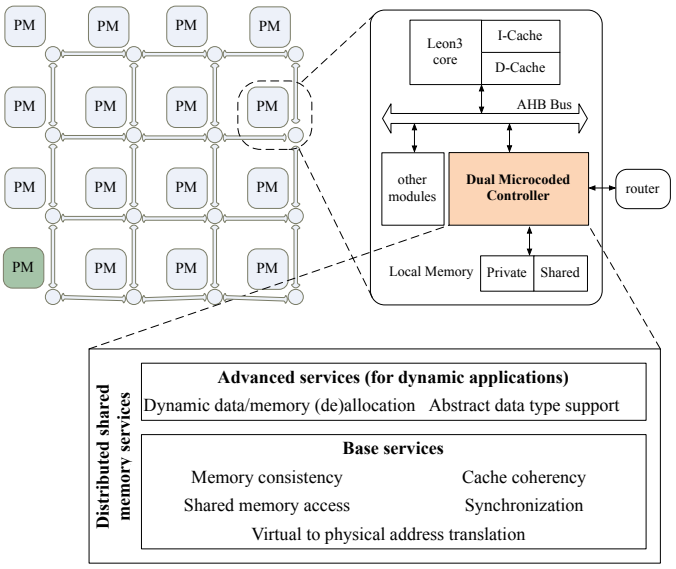

Fig. 2 A 16-node mesh McNoC; Processor-Memory (PM) node and Supported services.

master clock. The GRLS paradigm constrains all local frequencies to be rationally-related, and uses clock dividers for the generation of the local frequencies. The asynchronous communication problem is inherently more complex compared to the ratiochronous counterpart, and we used the periodic properties of rationally-related systems to build efficient latency-insensitive communication interfaces, allowing maximum throughput, low latency and low overhead, coupled with low complexity and high flexibility. We have shown how GRLS communication does not require handshake and has overhead and performance figures which are close to those of mesochronous interfaces, while keeping a flexibility close to that of GALS. We used the GRLS paradigm as the basis for the MOSART power management scheme, which partitions the SoC into different clock regions, which can be optimized independently from each other by means of Dynamic Voltage and Frequency Scaling (DVFS). Voltage Scaling is realized using a quantized approach, in which multiple supply voltages are distributed throughout the chip and the regions can dynamically select which voltage to use for power supply. We have developed fully-programmable Power Management Units to manage power services in the platform. The Power management Units allow to dynamically change the frequency and the supply voltage of any region and offers clock gating and shutoff services. Dynamic reconfiguration of the GRLS regions is also supported.

Table 1 Description of the five different allocators

Allocator	Description (free-lists)	Code size
Alloc1	free-list0($blockSize = 40$) free-list1($blockSize = 1460$) free-list2($blockSize = 1500$) Generic heap (holds blocks of other sizes)	4792 Bytes
Alloc2	free-list0($blockSize \in [0,40]$) free-list1($blockSize \in [1280,1460]$) free-list2($blockSize \in (1460,1500]$) Generic heap (holds blocks of other sizes)	4792 Bytes
Alloc3	free-list0($blockSize \in [0,40]$) free-list1($blockSize = 1460$) free-list2($blockSize = 1500$) free-list3($blockSize \in (40,92]$) free-list4($blockSize \in (92,132]$) free-list5($blockSize \in (132,256]$) free-list6($blockSize \in (256,512]$) free-list7($blockSize \in (512,1024]$) free-list8($blockSize \in (1024,1500)$) Generic heap (holds blocks of other sizes)	7728 Bytes
Alloc4	Similar to Alloc 1 with the addition of free-list3($blockSize = 92$)	5824 Bytes
Alloc5	Similar to Alloc 2 with the addition of free-list3($blockSize = 92$)	5824 Bytes

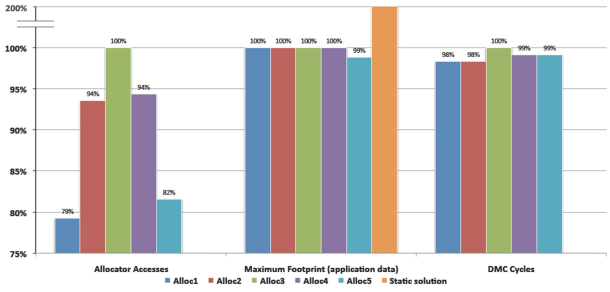

Fig. 3 (a) Description and (b) Comparison of the different memory allocators.

2.4 ASIP exploration

The ASIP design space can be very complex, and the performance estimations become very late in the design process in traditional approaches. The amount of manual work is considerable and the design cycle takes so much time that the exploration of the ASIP architecture design space remains very weak. The proposed methodology and prototype tool is according to our knowledge the first attempt to raise the ASIP design abstraction level above a standalone ASIP. Adding the ASIP architecture exploration to front of an existing ASIP design flow will allow for finding a good architecture for the actual design of an ASIP. It facilitates evaluation of the

ASIP performance early in the design process which results in a more systematic approach, increase automation and allow exploration of larger ASIP design space. The method and tool gives estimates of number of registers, number and types of functional units, number of pipeline stages and the instruction set of the ASIP core that would best satisfy the computational requirements of the types of algorithms it is targeted for. From the set of core models in the design space, the approach finds the most optimal for the given algorithm.

3 Experimental Results

In this Section, examples of MOSART's are presented. According to the aforementioned methodology we show the results in the field of *(a) Parallelization, (b) System level exploration, (c) Supporting distributed shared memory services and (d) ASIP exploration.*

3.1 Parallelization

The aforementioned application of selected parts of the PHY layer of an experimental prototype of an IEEE 802.16e based broadband wireless system (Section 2.1) was used as input to the parallelization tool. The first step towards code mapping was the modification of the C sources, so that the coding style is conformant with MPA syntax and semantics requirements. Once the C source was cleaned-up, the sequential code was executed on a conventional PC platform to verify that the application functionality had been preserved. It was then annotated with standard C syntax labels, to facilitate parallelism extraction from the MPA tool and conversion of the original sequential code into a multi-threaded version. Instrumentation code had also been added by the MPA suite, to facilitate gathering of vital program statistics which are displayed in textual and visual form once the modified code is compiled and run onto the targeted MPSoC platform. The parallelization and optimization process was thus guided towards production of multi-threaded code, matching the capabilities of the multi-core platform.

During the learning phase of MPA use, trivial parallelization scenarios were run, where a single thread executes all functionality of the labeled code segments. Subsequently, more elaborate parallelization scenarios were tried, initially extracting the "easy" parallelism that is suggested by the application code outline: the iFFT/FFT blocks were assigned to individual threads, with the rest of code functionality assigned to a couple of additional threads. During the process of code parallelization, additional "unsafe" code features were identified and removed from the original sequential code, that is always the starting point of the exploration effort. Issues such as arrays of structures and inconsistent use of declared multi-dimensional arrays, which are forbidden by MPA although allowed by C semantics and able to go through production compilers such as gcc, were removed from the code. All such

transformations of the sequential original sources were validated by runs on the PC platform.

3.2 System level exploration

The JPEG encoder was used to experiment and validate the GCC compiler-based workload generation tool in the context that takes MPA-parallelized source codes, creates respective workload models and maps them on the ABSOLUT platform model for transaction-level performance simulation in SystemC. The parallel versions of the JPEG encoder were created with the MPA tool. We generated four sets of workload models from the encoder. The first one was from the unmodified sequential application and the other three from parallelized versions of the application: Par-1 had two threads with the second thread executing Getblock and DCT algorithms. Par-2 consisted of three threads with the second and third one interleaving the execution of Getblock, DCT, and Quantization. Par-3 had also three threads with the second and third thread executing just Getblock and DCT in an interleaved manner.

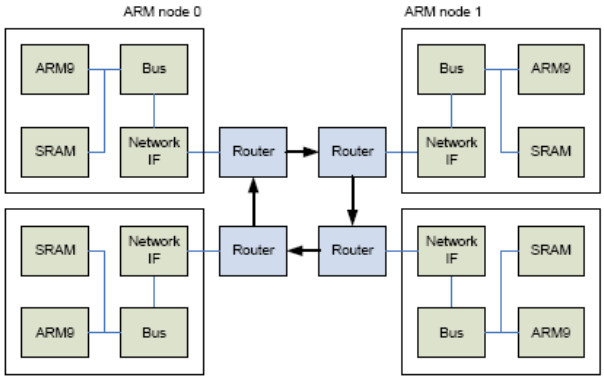

Fig. 4 Example platform consisting of four ARM nodes.

The execution platform model for the performance simulation of the JPEG application is depicted in Fig. 4. It consists of 4 ARM nodes connected by routers, which form a ring-like network. Each node has an ARM9 CPU, some local SRAM memory, a shared bus, and an interface to the other nodes. The accuracy of the ABSOLUT simulation approach has been evaluated with several case examples in [5, 6].

According to [6] both Par-1 and Par-3 have 100% utilization on the cpu of the ARM node 0. Par-1 has 44% cpu utilization in the second ARM node, whereas Par-3 has 21% utilization across nodes 1 and 2. Par-2 has 88% utilisation in the first node: it is idling at some point of simulation while waiting data from the other two

threads. Since Par-2 has a shorter execution time and more work for nodes 1 and 2, the cpu utilisation in those nodes is considerably higher at 53%

3.3 Supporting distributed shared memory services

3.3.1 Utilization of Base Services

We implemented two applications, matrix multiplication and 2D radix-2 DIT FFT, on the McNoC platform (See Fig. 2) with a range of sizes from 1 node to 64 (8×8) nodes. The matrix multiplication, which is computation intensive and does not involve synchronization, calculates the product of two matrices, $A[64,1]$ and $B[1,64]$, resulting in a $C[64,64]$ matrix. To vary the computation time, we consider both integer and floating point matrix multiplications. Fig. 5 shows the system speedup for the two applications. As the system size increases from 1 to 64 cores, the speedup rises from 1 to 36.494 for the integer matrix multiplication, from 1 to 52.054 for the floating point matrix multiplication, and from 1 to 48.776 for the 2D FFT. The speedup for the floating point matrix multiplication is higher than that for the integer matrix multiplication. This is as expected, because, when the computation takes more time, the portion of communication time becomes less significant, thus achieving higher speedup. That is to say, as the system size increases, communication becomes a more limiting factor for performance due to nonlinear increase in communication latency. In all cases, the DME overhead is insignificant.

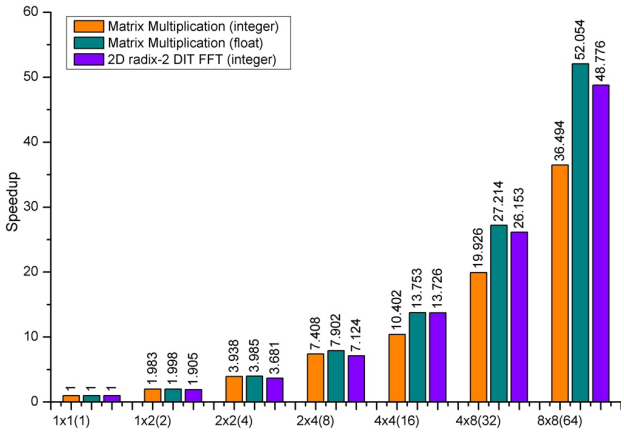

Fig. 5 Speedup of matrix multiplication and 2D DIT FFT.

3.3.2 Utilization of Advanced Services

The application we use as a test driver is a combination of several real-life kernels that are present in network applications [7, 8]. We triggered the system with a set of traces from a real wireless network. The software application is fully multi-threaded as it is increasingly common in computing systems: each kernel is executed in its own independent thread and communicates asynchronously with the other kernels through asynchronous FIFO queues. Through extensive application profiling we captured the allocation behavior of the application [8]. This information contains the block size distribution of the memory allocation requests. Based on the allocation behavior of the application the most appropriate allocator would be a pure-private one [9], offering the best performance in multi-processor environments. To evaluate our approach we use five different pure-private memory allocators, presented in Table. 1.

The results are presented in Fig. 3, where a comparison is performed in terms of number of memory accesses, maximum requested memory footprint and DME cycles. Out of the five memory allocators Alloc1 is the one that offers the smaller amount of memory accesses (21% less than Alloc3, which is the most complex one) and DME cycles, as it has the simplest internal structure and thus needing few memory accesses to service the allocation requests. Since all memory allocators have their free-lists and mapping functions to match the application's requirements, they all have similar behavior regarding the requested maximum memory footprint. However, when the static memory solution is compared against the dynamic one it requires 100% more memory footprint (Fig. 3).

3.3.3 Power Management Services

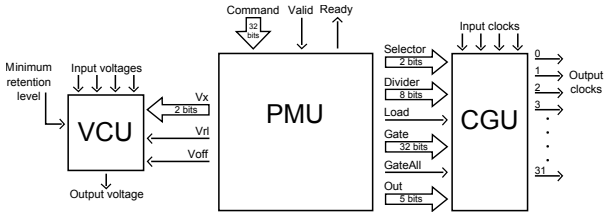

Fig. 6 Structure of the Power Management System

The power services are accessed by the Power Management Intelligence software (PMINT) through what we call Power Management System (PMS) (Fig. 6). The PMS is made up of three separate blocks: the Power Management Unit (PMU), the Clock Generation Unit (CGU) and the Voltage Control Unit (VCU). The Power Management Intelligence PMINT communicates with the PMU, which is a complex set of state machines giving access to the power services. The power services are

coordinated by the PMU and actuated by the CGU and the VCU, used respectively to generate the local clock(s) for the region and to regulate its supply voltage. While some of the power management services involve only CGU or VCU, the majority involve both units under the supervision of the PMU. The offered services are: a) changing frequency, b) changing voltage, c) changing DVFS point, d) clock gating, e) hybernation and f) power off. The maximum frequency (post layout) of the PMU is $1.25GHz$.

3.4 ASIP exploration

3.4.1 Initial profiling

There are three profiling level. The highest one (the least detailed) only simulate the total cycles required to run the whole application. On the other hand, the lowest one (the most detailed) profiles the time spent in each functions called in the C code (emulation function included). Both this two levels are activated by default while profiling. The third level of profiling is user-defined, and profiles the time spent in a (or several) user-defined part (called section) of the C code; usually, the detail level of this profiling is between the two other level. The initial profiling, with meaningful defined sections, on the VLIW architecture template provided by Processor Designer showed most of the cognitive radio application runtime were required to perform the wideband filter, as shown in Figure 7.The "not profiled" part is mainly composed of extra cycles introduced by the user-defined profiling section, for about 10% among 15%. The 5% last percents gathered mainly memory management (malloc, free, etc.), and the filter coefficient initialisation.

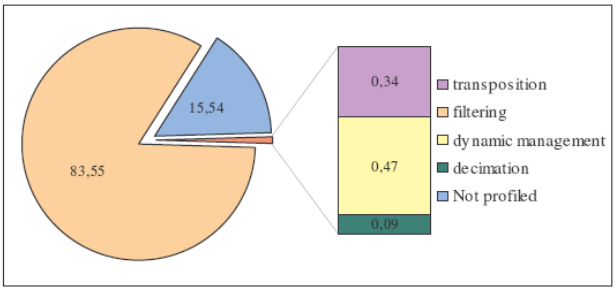

Fig. 7 Initial profiling results on the VLIW architecture template

3.4.2 MAC instruction

First, as the filtering required more than 85% of the total runtime, a MAC instruction has been added in the processor instruction set to speed this up. In order to not slow down too much the frequency, this MAC operation is implemented within two pipeline stages, i.e. within two clock cycles. During, the first cycle, operands are

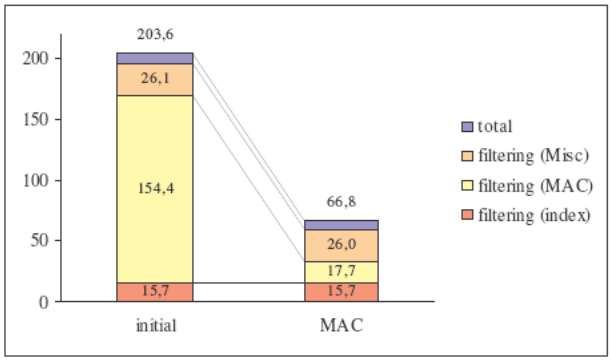

Fig. 8 Profiling result with the MAC instruction

read and the multiplication is computed. Then, during the second cycle, the multiplication result is accumulated in the destination result, i.e. added with the destination register previous value. The implementation of this operation has been automatically mapped to a multiplication-accumulation from the C to the assembly code tested and validated. Then, the application has been profiled again on this optimised processor, and its results are shown on Figure 8.

As a comparison, initial profiling results are shown on the left (figures are in millions of cycles). Moreover, the total runtime is cut in four different parts. The first one (”MAC”, represented in yellow) is the time spent computing the multiplication-accumulation. The second one (”index”, represented in red) is the time spent computing the memory address of the MAC operands (input sample and filter coefficient). The third one (”misc”, represented in orange) is the remaining time spent in the filtering function; it is mainly composed of extra (useless) cycles introduced by the user-defined profiling sections, and of loop branching instruction and index computing. Eventually, the last part (represented in blue), is the time left (transposition, decimation, etc.). Among these four parts, the MAC instruction obviously optimized the ”MAC” part; it speeds it up more than 8 times.

3.4.3 Branch prediction

The MAC instruction enhanced much the step 1 runtime. A second analyzing of these new performances showed many cycles were wasted in computing branch condition. As the application is based on nested loops, there are many conditional branch instructions, and each of them required a pipeline stall to compute whether the condition is true or not. The second ASIP optimization consists in implementing a loop-optimized branch prediction. It means that the processor would recognize a conditional branch that corresponds to a loop (defined by a branching address before the current program address). Then, it would automatically take the branch without computing the condition which is actually true most of the time. Then, the condition

is computed to check the branching was right; if not, the processor goes back the instruction right after the branch instruction. Performances for this new ASIP have then been profiled, and the results are shown in Figure 9.

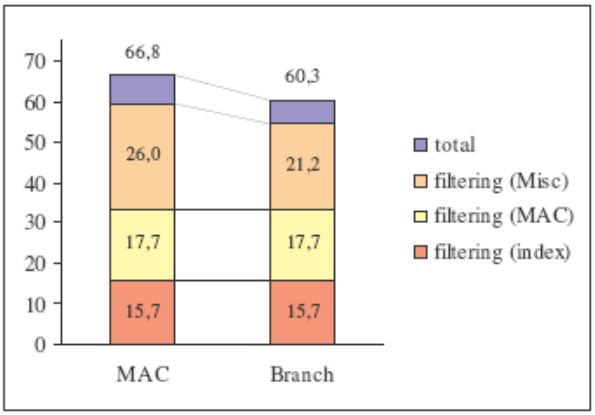

Fig. 9 Profiling result with the branch prediction

3.4.4 SW / HW performances trade-off

Figure 10 provides a summary of the achieved results about the software / hardware trade-off. It represents the different gains from the initial VLIW architecture. The first gain (cycles) represents the software gain, i.e. how faster (in comparison with the initial version) the application runs considering the same chip frequency; for example, a gain of 5 means than the application required fives times less cycles to run in comparison with the initial version. Then the next two figures (frequency and area) are related to the hardware impact. A figure greater than one means that the design is better than the initial one; for area, it means the chip is smaller, and for frequency, it means it run faster. Eventually, the last one (runtime) takes into account both software and hardware gains. Runtime represents the time (in seconds) required to run the application, and is determined from both the amount of cycles and the chip frequency. Actually, it shows than the great software gain afforded by the MAC instruction is partially counterbalanced by the frequency fall it introduces.

4 Conclusions

The objective of the MOSART project is to develop a flexible, modular multi-core on-chip platform architecture and associated exploration design methods and tools, to allow the scaling of the platform and optimization of its constituent elements for various embedded, multimedia and wireless communication applications. In MOSART, we have deployed a cluster of ASIPs to target a suite of applications and we enhance the efficacy of the MPSoC concept by using distributed memory ar-

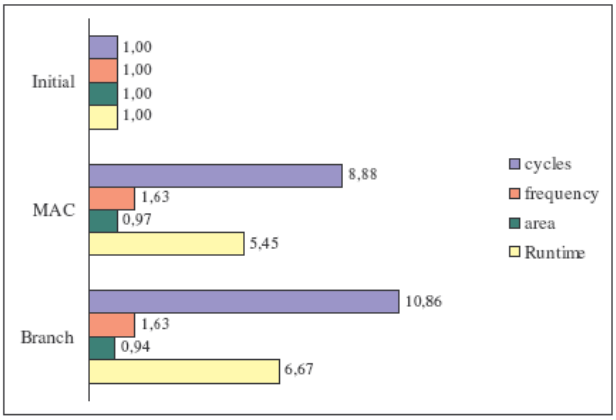

Fig. 10 Software / Hardware trade-off

chitecture and use of NoC. By adopting such an architecture, we claim that we not only gain flexibility, scalability and modularity, we also improve the computational efficiency to the extent that in the ladder of computational efficiency, the proposed architecture would be only one notch below hardwired ASICs and yet largely retain the flexibility of programmable solutions.

References

1. J.-Y. Mignolet *et al.*, "Mpa: Parallelizing an application onto a multicore platform made easy," *IEEE Micro*, vol. 29, no. 3, pp. 31–39, 2009.
2. J. Kreku *et al.*, "Combining uml2 application and systemc platform modelling for performance evaluation of real-time embedded systems," *EURASIP J. Embedded Syst.*, vol. 2008, pp. 1–18, 2008.
3. X. Chen *et al.*, "Supporting distributed shared memory on multi-core network-on-chips using a dual microcoded controller," in *Proc. of DATE*, 2010, pp. 39–44.
4. J.-M. Chabloz and A. Hemani, "A flexible communication scheme for rationally-related clock frequencies," in *Proc. of ICCD*, 2009, pp. 109–116.
5. J. Kreku *et al.*, "Workload simulation method for evaluation of application feasibility in a mobile multiprocessor platform," in *Proc. of DSD*. IEEE Computer Society, 2004, pp. 532–539.
6. ——, "Automatic workload generation for system-level exploration based on modified GCC compiler," in *Proc. of DATE*, 2010.
7. A. Bartzas *et al.*, "Enabling run-time memory data transfer optimizations at the system level with automated extraction of embedded software metadata information," in *Proc. of ASP-DAC*, 2008, pp. 434–439.
8. ——, "Software metadata: Systematic characterization of the memory behaviour of dynamic applications," *Journal of Systems and Software*, vol. In Press, Corrected Proof, pp. –, 2010.
9. P. R. Wilson *et al.*, "Dynamic storage allocation: A survey and critical review," in *Proc. of IWMM*. Springer-Verlag, 1995, pp. 1–116.