SESAM/Par4All: A Tool for Joint Exploration of MPSoC Architectures and Dynamic Dataflow Code Generation

N. Ventroux, T. Sassolas, A. Guerre
CEA, LIST,
Embedded Computing Laboratory
91191 Gif-sur-Yvette CEDEX, France;
nicolas.ventroux@cea.fr

B. Creusillet, R. Keryell
HPC Project
9 route du Colonel Marcel Moraine
92360 Meudon la Forêt, France
beatrice.creusillet@hpc-project.com

ABSTRACT

Due to the increasing complexity of new multiprocessor systems on chip, flexible and accurate simulators become a necessity for exploring the vast design space solution. In a streaming execution model, only a well-balanced pipeline can lead to an efficient implementation. However with dynamic applications, each stage is prone to execution time variations. Only a joint exploration of the application space of parallelization possibilities, together with the possible MPSoC architectural choices, can lead to an efficient embedded system. In this paper, we associate a semi-automatic parallelization workflow based on the Par4All retargetable compiler, to the SESAM environment. This new framework can ease the application exploration and find the best tradeoffs between complexity and performance for asymmetric homogeneous MPSoCs and dynamic streaming application processing. A use case is performed with a radio sensing application implemented on a complete MPSoC platform.

Categories and Subject Descriptors

C.0 [**General**]: Modeling of computer architecture; C.4 [**Per-formance of systems**]: Modeling techniques; I.6.4 [**Com-puting and modeling**]: Simulation and modeling—*Model Validation and Analysis*; D.3 [**Software**]: Programming Languages; D.3.4 [**Programming Languages**]: Processors—*compilers, code generation, retargetable compilers*

General Terms

Design, Performance

Keywords

MPSoC, processor modeling, TLM, SystemC, simulation, performance analysis, source-to-source compilation

RAPIDO '12 January 23 2012, Paris, France

1. INTRODUCTION

The emergence of new embedded applications for telecom, automotive, digital television and multimedia applications, has fueled the demand for architectures with higher performances, and better chip area and power efficiency. These applications are usually computation-intensive, which prevents them from being executed by general-purpose processors. In addition, architectures must be able to simultaneously manage concurrent information flows; and they must all be efficiently dispatched and processed. This is only feasible in a multithreaded execution environment. Designers are thus showing interest in System-on-Chip (SoC) paradigms composed of multiple computation resources connected through networks that are highly efficient in terms of latency and bandwidth. The resulting new trend in architectural design is the MultiProcessor SoC (MPSoC) [1].

Another very important feature of future embedded computation-intensive applications is the dynamism. Algorithms become highly data-dependent and their execution time depends on their input data, since decision processes must also be accelerated. Consequently, on a multiprocessor platform, optimal static partitioning cannot exist since all the processing times depend on the given data and are prone to non-uniform data accesses. In [2], it is shown that the solution consists in dynamically allocating tasks according to the availability of computing resources. Global scheduling should maintain a balanced system load and support workload variations that cannot be known off-line. Moreover, the preemption and the migration of tasks dynamically balance the computation power between concurrent processes. Only an asymmetrical approach can implement a global scheduling and efficiently manage dynamic applications.

An asymmetric MPSoC architecture consists of one (sometimes several) centralized or hierarchized control core, and several homogeneous or heterogeneous cores for computing tasks. The control core handles the tasks scheduling. In addition, it performs load balancing through task migrations between the computing cores when they are homogeneous. The asymmetric architectures usually have an optimized architecture for control. This distinction between control and computing cores renders the asymmetric architecture more transistor/energy efficient than the symmetric architectures.

One possible approach to parallelize an application is to pipeline its execution. This programming and execution model suits well data oriented applications that consider a

continuous flow of data. An asymmetric MPSoC can dynamically distribute the pipeline stages among computing resources. Only a well-balanced pipeline application will lead to a good efficiency.

In previous works [3, 4], we developed the SESAM tool to help the design of new asymmetric MPSoC architectures. This tool allows the exploration of MPSoC architectures and the evaluation of many different features (effective performance, used bandwidth, system overheads...). In this paper, we associate the SESAM environment to a semi-automatic code generation workflow using Par4All [5]. For the first time, two exploration tools, one for the architecture, one for the task code generation of dataflow applications, are associated to create a complete exploration environment for embedded systems. With the implementation of a significant application from radio telecommunication domain (radio sensing) on a complete asymmetric MPSoC architecture, we will validate our work and show how the association of our tools can really help tune both the application and the architecture.

This paper is organized as follows: Section 2 covers related works on MPSoC simulators from both industrial and academic worlds, as well as related works on compilation of streaming applications. Then, section 3 gives an overview of the initial SESAM environment. Section 4 presents its programming model and the available primitives to support pipelined dataflow applications, while Section 5 depicts the code generation tool using Par4All. Section 6 presents the whole framework that associates both the SESAM environment and Par4All. Section 7 illustrates the performance results obtained by running a real case embedded application on a complete MPSoC architecture implemented with SESAM. Finally, section 8 concludes the paper by discussing the presented work.

2. RELATED WORK

Lots of works have been published before on single-processor, multiprocessor and full-system simulators [6, 7]. Some of them focus on the exploration of specific resources. For instance, Flash [8] eases the exploration of different memory hierarchies, or SICOSYS [9] studies only different Network-on-Chips (NoCs). Taken separately, these tools are very interesting but a complete MPSoC exploration environment is needed in order to analyze all architectural aspects under real application processing case.

Among complete MPSoC simulators, MC-Sim [7] uses a variety of processors, memory hierachies or NoC configurations but remains cycle-accurate. On the contrary, simulators like STARSoC [10] offer a rapid design space exploration but only consider functional level communications. To study network contentions and the impact of communication latencies, a timed simulation is necessary. Others, like ReSP [11], use generic processors and cannot take into account instruction set specificities. This does not allow to size and to validate MPSoC architectures. On the contrary, some simulators, like MPARM [12], are processor specific and do not allow the exploration of different memory system architectures or different processors, and hence lack flexibility.

Some of the simulators benefit from the genericity of a very high description level, like Sesame [13] or CH-MPSoC [14]. They use a gradual refine Y-Chart methodology to explore the MPSoC design space. However, even if they remain very promising tools, they cannot support complex IPs or MPSoC structures with advanced networking solutions. Generated architectures remain very constrained. Less generic projects exist, like SoCLib [15], but their scope are too limited to fulfil MPSoC exploration and in particular they cannot support automatic MPSoC generation to analyze its parameters.

Some very interesting projects [16, 17, 18] make a model of a large set of MPSoC platforms. Nonetheless, these solutions do not propose a rich set of Network-on-Chips (NoCs), and it is not possible to easily integrate a centralized element to dynamically allocate tasks to resources. The programming model consists in statically allocating threads onto processors, and does not allow the design of architectures optimized for dynamic applications.

There is a lot of studies about the compilation of streaming applications for multicore systems. They first differ in the abstractions they provide to express task parallelism: specific languages [19, 20], extensions of subsets of standard languages such as C [21], or pragmas [22, 23]. Our approach belongs to the last category, with the difference that the user does not need to specify the input and output data. They are computed internally using array region analyzes [24].

Then, the studies address the difficulty of providing an optimal task scheduling for the target architecture, either at compile time (*static scheduling* [19]) or through a runtime (*dynamic scheduling* [22, 25]). Because asymmetric MPSoC architectures already come with a dynamic scheduling of tasks, Par4All has only to generate a simple task graph representing task dependencies to control the application pipeline. Besides, computation tasks rely on specific communication mechanisms to support a streaming execution.

Finally, to the authors' knowledge, there is no published work on a complete simulation tool chain that supports the exploration of asymmetric MPSoC architectures and associates a semi-automatic code generation for streaming applications.

3. SESAM OVERVIEW

SESAM is a tool that has been specifically built up to ease the design and the exploration of asymmetric multiprocessor architectures [3]. SESAM can also be used to analyze and optimize the application parallelism, as well as control management policies. This tool is made of various instruction set simulators (MIPS, PowerPC, Sparc), networks-on-chips (multibus, mesh, torus, multistage, ring), a DMA, a memory management unit, caches, memories and different control solutions to schedule and dispatch tasks. All simulator provided blocks can be timed.

This framework is described with the SystemC description language, and allows MPSoC exploration at the TLM level with fast and cycle-accurate simulations. Besides, SESAM uses approximate-timed TLM with explicit time to provide a fast and accurate simulation of highly complex architectures. This model, described in [26] uses the Transactional Level Modeling (TLM) approach coupled with timed communica-

tions. This solution allows the exploration of MPSoCs while reflecting the accurate final design. Regarding the communications, we point out a 90 % accuracy compared to a fully cycle-accurate simulator. Time information is necessary to evaluate performances and to study communication needs and bottlenecks. It supports co-simulation within the ModelSim environment [27] and takes part in the MPSoC design flow, since all the components are described at different hardware abstraction levels.

To ease the exploration of MPSoCs, all the components and system parameters are set at run-time from a parameter file without platform recompilation. It is possible to define the memory map, the applications that must be loaded, the number of processors and their type, the number of local memories, their size, the parameters of the instruction and data caches, memory latencies, network latencies, network topologies (torus, ring, mesh...) etc. More than 160 parameters can be modified. For instance, we can study the pipeline length impact of processing elements [28]. Moreover, each simulation brings more than 250 different platform statistics. That helps the designer size the architecture. For example, SESAM collects the miss rate of the caches, the memory allocation history, the processor occupation rate, the number of preemptions, the time spent to load or save the context of tasks, or the effective used bandwidth of each network.

Energy consumption is a very important parameter to be considered at each step of the design process. Different solutions have been implemented into the SESAM framework to allow the exploraiton of different energy consumption strategies based on DPM an DVFS modes. These strategies can exploit filling rate of shared buffers to dynamically balance the streaming flow [29]. In addition, in order to estimate the energy consumption of processors according to applications, PowerArchC has been developed and implemented into SESAM [30].

A script can be used to automatically generate several simulations by varying different parameters in the parameter file, as well as different applications. An Excel macro imports these statistics to study their impact on performances. In addition, SESAM offers the possibility to automatically dispatch all the simulations to different host PCs. For example, 400 simulations can be carried out with 12 hosts in less than one hour and a half [3].

Debugging the architecture is possible with a specific GNU GDB [31] implementation. In the case of a dynamic task allocation modeling, it is not possible to know off-line where a task will be executed. Therefore, we build up a hierarchical GDB stub that is instantiated at the beginning of the simulation. A GDB instance, using the remote protocol, sends specific debug commands to dynamically carry out breakpoints, watchpoints, as well as step by step execution, on an MPSoC platform. This unique multiprocessor debugger allows the task debugging even with dynamic migration between the cores. Moreover, it is possible to simultaneously debug the platform and the code executed by the processing resources.

4. SESAM PROGRAMMING MODEL

The programming model of SESAM is specifically adapted to dynamic applications and global scheduling methods. Obviously, it is inconceivable to carry out a generic programming model for all asymmetrical MPSoCs. Nonetheless, it is possible to add new programming models. The programming model is based on the explicit separation of the control and the computation parts.

The control task is a Control Data Flow Graph (CDFG) extracted from the application, which represents all control and data dependencies. The control task handles the computation task scheduling and other control functionalities, like synchronizations and shared resource management. It must be written in a dedicated and simple assembly language. Each control task, for each different application, needs to define: the number of computation tasks, the binary file names corresponding to these tasks, and their necessary stack memory size. Then, we must specify which are the first and last tasks of the application. Finally, for real-time task scheduling, the deadline of the application, as well as the worst case execution time of each task, must be defined. The processor type of each task is also specified and this information is used during the allocation process. A specific compilation tool is used for the binary generation.

A computation task is a standalone program, which can use the SESAM Hardware Abstraction Layer to manage shared memory allocations and explicit synchronizations. This HAL is summarized in Table 1. It can be extended to explore other memory management strategies. In the SESAM framework, the memory space can be implemented with several banks or a single memory. The Memory Management Unit (MMU) manages the memory space and shares it between all active tasks.

HAL functions	Description
Memory allocation functions	
sesam_reserve_data()	reserve pages
sesam_data_assignation()	allocate the data
sesam_free_data()	deallocate the data
sesam_chown_data()	change the data owner
Data access functions	
sesam_read(), sesam_write()	read or write a data
sesam_read_burst()	read a finite nb of bytes
sesam_write_burst()	write a finite nb of bytes
sesam_read_byte()	read a byte
sesam_write_byte()	write a byte
Debug function	
sesam_printf()	display debug
Page synchronization functions	
sesam_wait_page()	wait for a page
sesam_send_page()	page is ready

Table 1: Hardware Abstraction Layer of SESAM

This HAL provides memory allocation, read/write shared memory access, debugging and page synchronization functions. Each data is defined by a data identifier, which is used to dialog between the memory management unit and the computation tasks. For instance, the function call *sesam_-data_assignation(10,4,2)* allocates *4* pages for the data ID *10* with *2* consumers for this data. The function call *sesam_write-_data(10,&c,4)* writes the word *c* starting from the 4^{th} byte of the data ID *10*. The *sesam_wait_page* func-

tion is a blocking wait method. The task waits for the availability of a page in only read or write mode. When all consumers have sent a write availability, the *sesam_send_page* function is used to inform the memory management unit that the content of the page is ready to be read, or that its content has become useless for the consumer task. The memory management unit can then release the page access rights and accept future writes. This hand-shake protocol is a semaphore-like processing and guarantees the data consistency.

When a *sesam_send_page* is sent to the MMU, the status of the page is updated. If the page was in a *write* mode, the consumer number is checked and updated. To distinguish multiple requests of a single task from multiple consumers' requests, a consumer list is maintained for each page. When all consumers have read the page, the page status changes and it becomes possible to write again into it. When a *sesam_wait_page* is sent to the MMU, the request is pushed into a *wait_dispo list request* and the information is sent to the controller. As soon as the page becomes available, the MMU sends to the processor an answer that unlocks the waiting *sesam_wait_page* function. Because a task can dynamically be preempted by the controller and migrated to another processing element, the MMU must be able to address the processor executing the waiting task. Thus, a *sesam_wait_page* is sent again when the task is resumed on the new processor in order to update the processing element address. This protocol is more depicted in [4].

With a streaming execution, the stages of the application pipeline must communicate through these synchronization primitives to access their shared buffers. A consumer must wait for the shared data to be written before reading it, in order to keep consistent data. To maximize the parallelism in the pipeline and ensure sufficient concurrent executions, the granularity of data synchronizations must be well-sized. A fine-grain synchronization level generates an important hardware and control overhead to implement all semaphores used to store the access status information. Thus, all shared data accesses are at the page level.

5. CODE GENERATION USING PAR4ALL

To complete the simulation tool chain, the Par4All retargetable compiler provides a source-to-source code generator for the SESAM HAL (see Figure 1). It relies on the PIPS parallelizing compiler [32] and on a specific runtime to relieve the programmer from generating each task code and the corresponding communications through the SESAM buffers. Hence, Par4All provides the possibility of programming in the usual sequential way (as in the code of Figure 3), and to focus on the choice of the computation kernels which will form the bases of the final application tasks.

The input code must be written in C and meet the Par4All coding rules, which mainly restrict the use of pointers. The computation kernels must have a structured control flow, be all declared in the `main` function, and must not be nested. Hence, the pragmas which designate the computation kernels can be placed before any structured statement such as a loop nest or a function call for instance.

Isolating the data of a pre-determined computation kernel

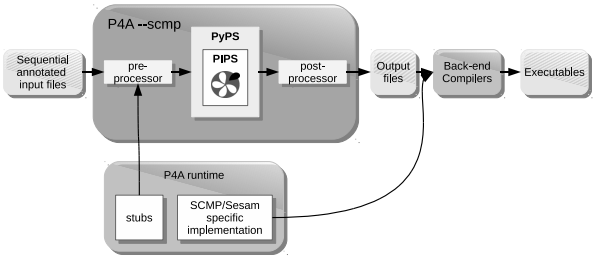

Figure 1: Par4All Compilation workflow

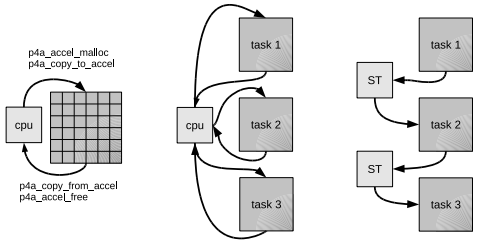

Figure 2: Programming models

from its environment to externalize its memory space on a distant medium, is the fundamental operation used in Par4All to generate code for GPUs [33, 34]. It includes the allocation of local data and the generation of communications to retrieve the original values from the source medium, and to send back the computed values. Hence the idea to reuse this component to generate code for the different tasks, including inter-tasks communications.

However, the original model for a CPU/GPU couple enforces that the communications are performed from/to a unique CPU process which collects the data (Figure 2(a)), ensuring the consistency of the latter. But keeping a unique process to gather the data to/from the tasks would sequentialize the whole application (Figure 2(b)). Therefore, we have generalized the model by introducing one *server task* for each datum implied in inter-tasks communications (Figure 2(c)). Because kernel tasks communicate only though server tasks, it ensures data consistency while preserving inter-tasks parallelism.

Another advantage of our model, is to avoid deadlocks, because, as shown in Figures 4 and 5, the codes finally generated for computation tasks and server tasks (here from the code of Figure 3) are completely symmetrical, and retain the control flow of the initial code. Thus, even if the execution conditions of communications are not known at compile time, they are the same on the server tasks and on the computation tasks during the execution, and the runtime ensures that there is always one, and solely one, consumer access per produced page of a communication buffer. This relieves the user from the painful debugging of a new parallel application every time he experiments a new task splitting strategy.

If we now look at the code generated for the first kernel task (Figure 4), we see that a new local variable (`P4A__a__0`) is allocated, and that the kernel performs its computation on this variable. Then, the values are copied back to the orig-

```
int main()
{
  int i, t, a[20], b[20];
  for (t=0; t < 100; t++) {
kernel_tasks_1:
    for(i=0; i <10; i++) a[i] = i+t;
kernel_tasks_2:
    for(i=10; i <20; i++) a[i] = 2*i+t;
kernel_tasks_3:
    for(i=10; i <20; i++) printf("a[%d] = %d\n", i, a[i]);
  }
  return(0);
}
```

Figure 3: Running example

inal variable (`a`) using the communication functions of the Par4All runtime, and the freeing function is called in case the local variable is not used anymore. Notice that the execution of the kernel is conditioned by a boolean value defined in the `scmp_buffers.h` header file. This file is automatically generated by Par4All and uses the value defined on the first line of the file (here `kernel_task_1`) to set the boolean values guarding the execution of the kernels and those describing how the tasks use the different buffers. For instance the value `kernel_task_1_p` is set to 1 if `kernel_task_1` is defined, and to 0 otherwise. Thus, the first kernel is executed by the first kernel task, and is skipped by all the other tasks. Similarly, `P4A__a__0_prod_p` is set to 1 if the task produces data in buffer `P4A__a__0`, and 0 otherwise. These values determine the behavior of the allocation and communications functions, which are not executed by the tasks which are not concerned.

The code of the server task corresponding to the original variable `a` (Figure 5) is the same, except for the communication functions which have specific server versions[1].

6. SESAM AND PAR4ALL

SESAM alone is a very efficient exploration framework to design asymmetric multiprocessors on chip. With the many parameters we can tune, it is possible to define the best trade-off to design a complete MPSoC. However, performances of an embedded system also depend on the application itself. The way it is implemented and parallelized can have a significant impact. With streaming applications, an unbalanced application pipeline can lead to very poor performances, even if the architecture parameters are well-sized. As a result, less computation-intensive tasks spend a long time waiting for available data. For this reason, we decided to associate Par4All to SESAM in order to also allow the exploration of the application, and so to design a complete efficient system. This new framework supports only homogeneous computing resources.

As shown in Figure 6, Par4All generates the control task, which is a CDFG graph, and all computation tasks source codes based on the SESAM HAL corresponding to the application pipeline, including kernel and server tasks. Par4All also generates an initial and a final task, as well as the Makefile to build the executables. The computation task executables are generated using a C cross-compiler corresponding to the computing resource type. A specific compiler provided by the SESAM framework is used for the control task. Depending on the execution results, it is then possible to change the kernel tasks by modifying the pragmas used in the input application, and run it again through Par4All, to better adapt the different stage lengths in the application pipeline.

[1]More details on the Par4All runtime implementation for SESAM can be found in [35].

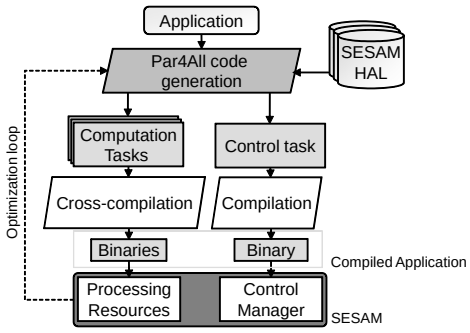

Figure 6: SESAM and Par4All exploration tool

With asymmetric MPSoCs, irregular processing length can be partially hidden. Because they have been designed to deal with dynamism, they can preempt and migrate tasks to balance computing resource's load between homogeneous resources. A waiting task can be preempted to be replaced by a ready one in order to increase the resource occupation. Thus, the greater the number of extracted independent and parallel tasks, the more the application can be accelerated at runtime. Therefore, in such a dynamic execution environment, if the pipeline is long enough to always have ready tasks to execute it is not necessary to get an equal length between the stages. On the same time, increasing the number of pipeline tasks increase the memory need and control overhead. Indeed, the number of shared buffers depends on the number of stages. In addition, increasing the number of synchronizations generates extra control overhead, since we have a centralized controller.

With these new capabilities, the SESAM/Par4All framework can ease the analysis of the impact of different parallelized code versions on performances, and efficiently size the MPSoC platform. It can help to explore the design space in order to build an efficient complete system. We now have to analyze and quantitatively evaluate this new framework on a real application in order to demonstrate its interest.

7. RESULTS

Our objectives here are two-fold: demonstrate the SESAM's capabilities to support the execution of dataflow applications on asymmetric MPSoC architectures, and show the advantages of using Par4All to port these applications to the SESAM environment. To do this we have considered a generic asymmetric MPSoC representative of the SCMP architecture [36].

As shown in Figure 7, this architecture is made of homogeneous MIPS32 processing elements (MIPS32R2 ISA compliant functional ISSs) and two DMAs for the communication with the external data memory. These devices communicate with 32 shared memories through a 64-bit multibus data network, where each memory bank is 128KB. These memories are locally shared and physically distributed. The data network and memory latencies are 2 cycles. All the processors use two private 4 KB instruction and 8 KB data

```
#define kernel_task_1 1
#include "scmp_buffers.h"
...
int main()
{
  P4A_scmp_reset();
  int i, t, a[20], b[20];
  for(t = 0; t <= 99; t += 1) {
    {
      int (*P4A__a__0)[10] = (int (*)[10]) 0;
      P4A_accel_malloc((void **) &P4A__a__0, sizeof(int)*10,
          P4A__a__0_id, P4A__a__0_prod_p || P4A__a__0_cons_p,
          P4A__a__0_prod_p);

      if (kernel_task_1_p) // true in this task
        for(i = 0; i <= 9; i += 1) (*P4A__a__0)[i-0] = i+t;

      P4A_copy_from_accel_1d(sizeof(int),20,10,0,
          P4A_sesam_server_a_p?&a[0]:NULL, *P4A__a__0,
          P4A__a__0_id, P4A__a__0_prod_p || P4A__a__0_cons_p);
      P4A_accel_free(P4A__a__0, P4A__a__0_id,
          P4A__a__0_prod_p || P4A__a__0_cons_p,
          P4A__a__0_prod_p);
    }
...
}
```

Figure 4: Kernel task code

```
#define P4A_sesam_server_a 1
#include "scmp_buffers.h"
...
int main()
{
  P4A_scmp_reset();
  int i, t, a[20], b[20];
  for(t = 0; t <= 99; t += 1) {
    {
      int (*P4A__a__0)[10] = (int (*)[10]) 0;
      P4A_accel_malloc((void **)&P4A__a__0, sizeof(int)*10,
          P4A__a__0_id,P4A__a__0_prod_p || P4A__a__0_cons_p,
          P4A__a__0_prod_p);

      if (kernel_task_1_p) // false in this task
        for(i = 0; i <= 9; i += 1) (*P4A__a__0)[i-0] = i+t;

      P4A_copy_from_accel_1d_server(sizeof(int),20,10,0,
          P4A_sesam_server_a_p?&a[0]:NULL,*P4A__a__0,
          P4A__a__0_id,P4A__a__0_prod_p||P4A__a__0_cons_p);
      P4A_accel_free(P4A__a__0, P4A__a__0_id,
          P4A__a__0_prod_p||P4A__a__0_cons_p,
          P4A__a__0_prod_p);
    }
...
}
```

Figure 5: Server task code

caches. Each cache is a 4-way set associative and implements a write-back protocol. The external network-on-chip (NoC) is a 2-cycle-latency 32-bit simple bus with a round-robin arbiter. External memories have a latency of 4 cycles. All latencies have been evaluated through partial TSMC 45 nm ASIC synthesis and normalized in relation to the PE frequency. The central control manager is a 5-stage RISC processor named AntX [37] with two instruction and data caches of 2 KB each. It executes a microkernel, which supports the dynamic scheduling and allocation of tasks on PEs. An interrupt controller is used to communicate with the computing resources. The architecture is also composed of a code loading unit (CLU) that can prefetch task instruction codes into shared memories, as well as a memory management unit (MMU). All devices are timed and only communications are approximate-timed transactions.

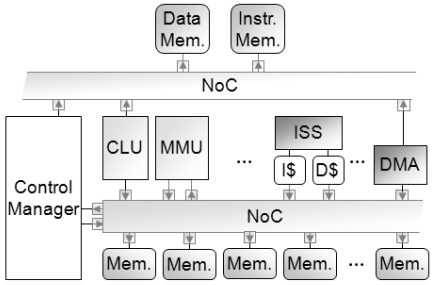

Figure 7: SESAM infrastructure

The original application is a radio sensing function developped by Thales Communication France, which aims to detect the unused spectra and to share them without interference with other users. In other words, the already used spectra are detected in order to identify spectrum holes. This application is used in spectrum monitoring and electronic warfare devices. To validate our approach, four parallelized versions have been obtained from the original sequential application:

- *Manual* is a manually parallelized version, very close to the original sequential code.
- *Semi-auto v1* is a first parallelized version obtained with Par4All using 8 initial kernel tasks.
- *Semi-auto v2* is a second parallelized version obtained with Par4All using 17 initial kernel tasks which approximately correspond to the tasks of the manual version.
- *Manual opt* is the last manually optimized version.

Table 2 summarizes their complexity and the time spent to develop them from the sequential application and then from the first manual or semi-automatic parallel version. In the manual versions, a double buffering communication scheme is used to maximize the concurrency between pipelined tasks. With Par4All, communications between two kernel tasks through a server task involve two shared buffers. Thus, tasks can independently execute the next frame from the previous pipeline stage results.

Applications	nb computation tasks	time
Manual	18	7 days
Semi-auto V1	19 (8 kernel + 9 server tasks)	1/2 day
Semi-auto V2	31 (17 kernel + 12 server tasks)	30 min
Manual opt	40	5 days

Table 2: Application description and time to developped them.

Figure 8-a shows that the total execution time can be heavily impacted by the total data wait time. This overhead is representative of an unbalanced pipeline. It even reaches 77 % with the first version obtained with Par4All. The second version improves the workload balancing in extending the application pipeline, and reducing in the same time the impact of longer stages. Results are even close to the first manual version, and this with much less development effort. We can see on Figure 8-b that the dynamic control of the architecture even partially hides the unbalance with 4 PEs. Unfortunately, the number of tasks remains limited for 8 PEs. Besides, Figure 8-a shows that the server task impact on the computation length remains moderate, and the Par4All strategy does not influence too much the overall performances compared to a manual implementation.

In Figure 8-c, the maximum memory use is depicted. As expected, due to server tasks, versions obtained with Par4All

double the memory use. But, even manual versions need to increase the number of buffers to get sufficient parallelism. However, with an equivalent number of kernel tasks, Par4All applications still increase the necessary memory of 65 %.

Finally, Figure 8-d shows the results obtained with the last manual optimized version. The streaming description of the application and the use of the streaming protocol to access shared data can have a non-negligible cost. Many accesses to a central device, such as the MMU, to get the authorization to write or read each page of a buffer, could have a very negative impact on performances. Only simulations help the evaluation of the potential benefit. However, we demonstrate that the parallelism obtained with a streaming execution can be important. We get an acceleration of 6.23 and an occupation rate beyond 97 % with 8 PEs. These results depend on the control overhead, which must be minimized. This is why, in our architecture, we use a microkernel especially developed to optimize the reactivity of the control.

These results show interesting benefits when using our mixed SESAM/Par4All framework. Par4All brings a very convenient way to generate multiple parallelized version of the application in order to find the right balance between the tasks, with a moderate impact on performances. Then, the SESAM environment brings to the designer the possibility to guide the application optimization with Par4All, and to size the architecture to the application needs. We also demonstrate that a streaming execution can be very efficient with a dynamic control of the pipeline execution.

8. CONCLUSION

This paper presented the association of two exploration tools, one for the architecture, one for the task code generation of dataflow applications, to create a complete exploration environment for embedded systems.

The efficiency of our new framework were studied through the simulation of a complete asymmetric MPSoC architecture running a radio sensing application. We demonstrated that SESAM can bring to the designer the possibility to guide the application optimization with Par4All, and that Par4All brings a very convenient way to generate multiple parallelized version of the application in order to find the right balance between the tasks, with a moderate impact on performances.

The next step could be to enhance Par4All to avoid generating server tasks when this is decidable at compile time, to reach the same efficiency as manually generated applications. Another interesting extension would consist in synthesizing information provided by SESAM in order to automatically make Par4All converge on the right adequation between the MPSoC platform and the application.

Acknowledgements

The authors would like to thanks G. Blanc and R. David for their contributions on SESAM, and the members of the PIPS team for their help, and in particular F. Irigoin for his careful reading of a previous report. The authors would also like to thank F. Broekaert from Thales Communications France for providing the radio-sensing application. Part of the research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement No. 100029.

9. REFERENCES

[1] A. A. Jerraya and W. Wolf. *Multiprocessor Systems-on-Chips*. Elsevier, 2005.

[2] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, April 2008.

[3] N. Ventroux, A. Guerre, T. Sassolas, L. Moutaoukil, C. Bechara, and R. David. SESAM: an MPSoC Simulation Environment for Dynamic Application Processing. In *IEEE International Conference on Embedded Software and Systems (ICESS)*, Bradford, UK, July 2010.

[4] N. Ventroux, T. Sassolas, R. David, G. Blanc, A. Guerre, and C. Bechara. SESAM Extension For Fast MPSoC Architectural Exploration And Dynamic Streaming Application. In *IEEE/IFIP International Conference on VLSI and System-on-Chip (VLSI-SoC)*, Madrid, Spain, September 2010.

[5] HPC Project. Par4All, automatic parallelization, http://www.par4all.org.

[6] J. J. Yi and D. J. Lilja. Simulation of computer architectures: simulators, benchmarks, methodologies, and recommendations. *IEEE Transactions on Computers*, 55(3):268–280, March 2006.

[7] J. Cong, K. Gururaj, G. Han, A. Kaplan, M. Naik, and G. Reinman. MC-Sim: An efficient simulation tool for MPSoC designs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 364–371, San Jose, USA, November 2008.

[8] J. Gibson et al. FLASH vs. (Simulated) FLASH: Closing the simulation loop. In *ACM ASPLOS*, Pittsburgh, USA, March 2000.

[9] V. Puente, J. Gregorio, and R. Beivide. SICOSYS: an integrated framework for studying interconnection network performance in multiprocessor systems. In *Euromicro Workshop on Parallel, Distributed and Network-based Processing*, Canary Islands, Spain, January 2002.

[10] S. Boukhechem and E.-B. Bouernnane. TLM Platform Based on SystemC For STARSoC Design Space Exploration. In *NASA/ESA Conference on Adaptive Hardware and Systems*, Noordwijk, The Netherlands, June 2008.

[11] G. Beltrame, C. Bolchini, L. Fossati, A. Miele, and D. Sciuto. ReSP: A non-intrusive Transaction-Level Reflective MPSoC Simulation Platform for design space exploration. In *Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 673–678, Seoul, Korea, January 2008.

[12] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. *VLSI Signal Processing Systems*, 41(2):169–182, 2005.

[13] A.D. Pimentel, C. Erbas, and S. Polstra. A Systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers*, 55(2):99–112, February 2006.

[14] H. Shen, P. Gerin, and F. Pétrot. Configurable Heterogeneous MPSoC Architecture Exploration Using Abstraction Levels. In *IEEE/IFIP International Symposium on Rapid System Prototyping*, Paris, France, June 2009.

[15] E. Viaud, F. Pêcheux, and A. Greiner. An efficient TLM/T modeling and simulation environment based on conservative parallel discrete event principles. In *DATE*, Nice, France, April 2009.

[16] A. Wieferink et al. A System Level Processor/Communication Co-Exploration Methodology for Multi-Processor System-on-Chip Platforms. In *International Conference on Design, Automation and Test in Europe (DATE)*, Paris, France, February 2004.

[17] P. Paulin, C. Pilkington, and E. Bensoudane. StepNP: A System-Level Exploration Platform for Network Processors. *IEEE Design & Test*, 19(6):17–26, November 2002.

[18] D. August, J. Chang, S. Girbal., D. Gracia-Perez., G. Mouchard, D. Penry, O. Temam, and N. Vachharajani. UNISIM: An Open Simulation Environment and Library for Complex Architecture Design and Collaborative Development. *Computer Architecture Letters*, 6(2):45–48, 2007.

[19] M. Gordon et al. A stream compiler for communication-exposed

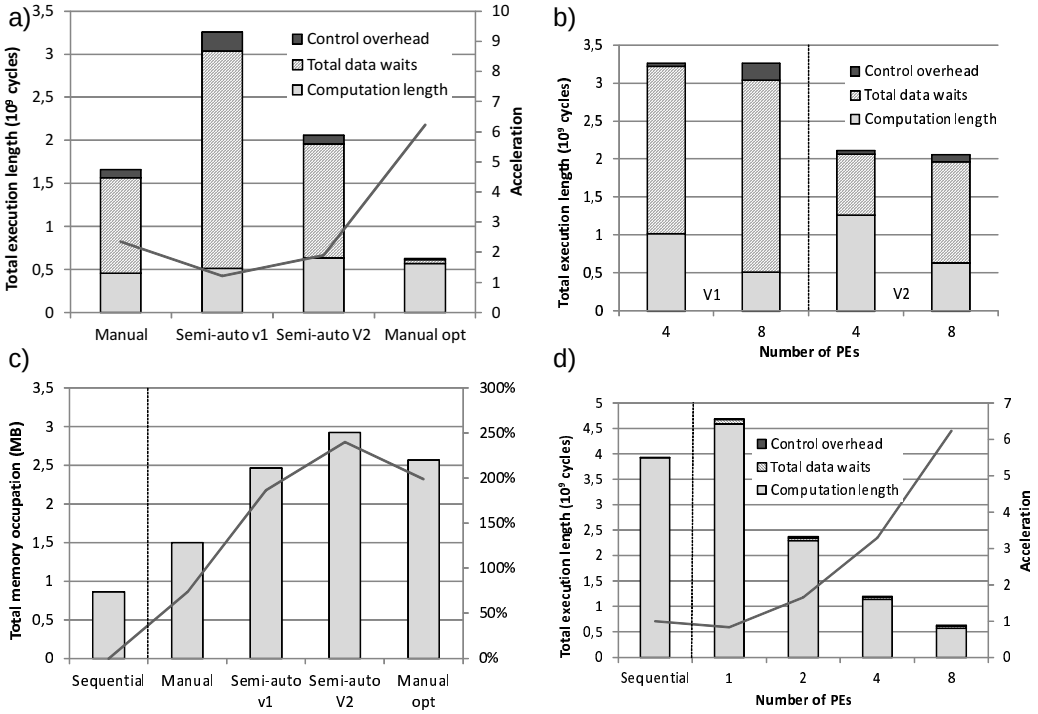

Figure 8: Radio sensing application results on a MPSoC platform with a streaming execution (except with the sequential version): (a) total execution time of the different parallelized version of the application on 8 processing elements (MIPS32 processors); (b) total execution time of semi-automatic parallelized version of the application on 4 and 8 processing elements; (c) total memory occupation of the different parallelized version of the application; and (d) total execution time of the manual last optimized version depending on the processing element (PE) number.

architectures. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ASPLOS-X, pages 291–303, New York, NY, USA, 2002. ACM.

[20] H. Vandierendonck, P. Pratikakis, and D.S. Nikolopoulos. Parallel programming of general-purpose programs using task-based programming models. In *Proceedings of the 3rd USENIX conference on Hot topic in parallelism*, HotPar'11, pages 13–13, Berkeley, CA, USA, 2011. USENIX Association.

[21] Y. Choi, Y. Lin, N. Chong, S. Mahlke, and T. Mudge. Stream Compilation for Real-Time Embedded Multicore Systems. In *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, CGO '09, pages 210–220, Washington, DC, USA, 2009. IEEE Computer Society.

[22] J.M. Perez, R.M. Badia, and J. Labarta. A dependency-aware task-based programming environment for multi-core architectures. In *2008 IEEE International Conference on Cluster Computing*, September 2008.

[23] E. Ayguade et al. The design of openmp tasks. *IEEE Transactions on Parallel and Distributed Systems*, 20(3), November 2009.

[24] B. Creusillet and F. Irigoin. Interprocedural Array Region Analyses. *International Journal of Parallel Programming (special issue on LCPC)*, 24(6):513–546, 1996.

[25] C. Augonnet, S. Thibault, R. Namyst, and P-A. Wacrenier. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 2010.

[26] A. Guerre, N. Ventroux, R. David, and A. Merigot. Approximate-Timed Transaction Level Modeling for MPSoC Exploration: a Network-on-Chip Case Study. In *Euromicro Conference on Digital System Design (DSD)*, Patras, Greece, August 2009.

[27] ModelSim. http://www.model.com/.

[28] C. Bechara, N. Ventroux, and D. Etiemble. Towards a Parameterizable Cycle-Accurate ISS in ArchC. In *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*, Hammamet, Tunisia, May 2010.

[29] T. Sassolas, N. Ventroux, N. Boudouani, and G. Blanc. A Power-Aware Online Scheduling Algorithm for Streaming Applications in Embedded MPSoC. In *IEEE International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Grenoble, France, September 2010.

[30] T. Gupta, C. Bertolini, O. Heron, N. Ventroux, T. Zimmer, and F. Marc. High Level Power and Energy Exploration using ArchC. In *IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Petrópolis, Brazil, October 2010.

[31] The GNU GDB project. http://www.gnu.org/software/gdb/.

[32] M. Amini, C. Ancourt, F. Coelho, B. Creusillet, S. Guelton, F. Irigoin, P. Jouvelot, R. Keryell, and P. Villalon. PIPS Is not (only) Polyhedral Software. In *First International Workshop on Polyhedral Compilation Techniques*, IMPACT, Chamonix, France, April 2011.

[33] S. Guelton. *Building Source-to-Source Compilers for Heterogeneous Targets*. PhD thesis, ENSTB, 2011.

[34] S. Guelton, R. Keryell, and F. Irigoin. Compilation pour cible hétérogènes: automatisation des analyses, transformations et décisions nécessaires. In *20ème Rencontres Francaises du Parallélisme*, Renpar, Saint Malo, France, May 2011.

[35] B. Creusillet. Automatic Task Generation on the SCMP architecture for data flow applications. http://www.par4all.org/documentation/publications, 2011.

[36] N. Ventroux and R. David. SCMP Architecture: An Asymmetric Multiprocessor System-on-Chip for Dynamic Applications. In *ACM International Forum on Next Generation Multicore/Manycore Technologies (IFMT)*, Saint-Malo, France, May 2010.

[37] C. Bechara, A. Berhault, N. Ventroux, S. Chevobbe, Y. Lhuillier, R. David, and D. Etiemble. A Small Footprint Interleaved Multithreaded Processor for Embedded Systems. In *IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, Beirut, Lebanon, December 2011.