ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix

Bingfeng Mei[12], Serge Vernalde[2], Diederik Verkest[23], Hugo De Man[12], and Rudy Lauwereins[12]

[1] IMEC vzw, Kapeldreef 75, Leuven, B-3001, Belgium
[2] Department of Electrical Engineering, Katholic Universiteit Leuven, Belgium
[3] Department of Electrical Engineering, Vrije Universiteit Brussel, Belgium

Abstract. The coarse-grained reconfigurable architectures have advantages over the traditional FPGAs in terms of delay, area and configuration time. To execute entire applications, most of them combine an *instruction set processor*(ISP) and a reconfigurable matrix. However, not much attention is paid to the integration of these two parts, which results in high communication overhead and programming difficulty. To address this problem, we propose a novel architecture with tightly coupled *very long instruction word*(VLIW) processor and coarse-grained reconfigurable matrix. The advantages include simplified programming model, shared resource costs, and reduced communication overhead. To exploit this architecture, our previously developed compiler framework is adapted to the new architecture without much difficulty. The results show that the new architecture is very compiler-friendly.

1 Introduction

Coarse-grained reconfigurable architectures have become increasingly important in recent years. Various architectures were proposed [1][2][3][4]. These architectures often comprise a matrix of functional units (FUs), which are capable of executing word- or subword-level operations instead of bit-level ones found in common FPGAs. This *coarse* granularity greatly reduces the delay, area, power and configuration time compared with FPGAs, however, at the expense of flexibility. Other features include predictable timing, a small configuration storage space, flexible topology, etc.

However, the reconfigurable matrix alone is not capable of executing entire applications. Most coarse-grained architectures are coupled with processors, typically RISCs. The execution model of such hybrid architectures is based on the well-known 90/10 locality rule[5], i.e., *a program spends 90% of its execution time in only 10% of the code.* Some computational-intensive kernels are mapped to the matrix, whereas the rest code is executed by the processor. So far not much attention is paid to the integration of the two parts of the system. The coupling between the processor and the reconfigurable matrix is often loose, which is essentially two separated parts connected by a communication channel. This results in programming difficulty and communication overhead. In addition, the coarse-grained reconfigurable architecture consists of components which are similar to those used in processors. This represents a major resource-sharing and cost-saving opportunity, which is not extensively exploited in traditional coarse-grained architectures.

To address the above problems, in this paper we presents a novel architecture called ADRES (*Architecture for Dynamically Reconfigurable Embedded System*), which tightly couples a VLIW processor and a coarse-grained reconfigurable matrix. The VLIW processor and the coarse-grained reconfigurable matrix are integrated into one single architecture but with two virtual functional views. This level of integration has many advantages compared with other coarse-grained architectures, including improved performance, a simplified programming model, reduced communication costs and substantial resource sharing. Nowadays, new programmable architecture can not succeed without good support for mapping applications. In our previous work, we built a compiler framework for a family of coarse-grained architectures [6]. A novel modulo scheduling algorithm was developed to exploit the loop-level parallelism efficiently[7]. In this paper, we present how this compiler framework can be adapted to the ADRES architecture. In addition, some new techniques are proposed to solve the integration problem of the VLIW processor and the reconfigurable matrix.

The paper is organized as follow. Section 2 describes the proposed ADRES architecture and analyzes its main advantages. Section 3 discusses how the compiler framework is ported to the ADRES architecture and some considerations of the compilation techniques. Section 4 reports experimental results. Section 5 covers related work. Section 6 concludes the paper and presents future work.

2 ADRES Architecture

2.1 Architecture Description

Fig. 1 describes the system view of the ADRES architecture. It is similar to a processor with an execution core connected to a memory hierarchy. The ADRES core(fig 3) consists of many basic components, including mainly FUs and register files(RF), which are connected in a certain topology. The FUs are capable of executing word-level operations selected by a control signal. The RFs can store intermediate data. The whole ADRES matrix has two functional views, the VLIW processor and the reconfigurable matrix. These two functional views share some physical resources because their executions will never overlap with each other thanks to the processor/co-processor model. For the VLIW processor, several FUs are allocated and connected together through one multi-port register file, which is typical for VLIW architecture. Compared with the counterparts of the reconfigurable matrix, these FUs are more powerful in terms of functionality and speed. They can execute more operations such as branch operations. Some of these FUs are connected to the memory hierarchy, depending on available ports.

Thus the data access to the memory is done through the load/store operation available on those FUs.

For the reconfigurable matrix part, apart from the FUs and RF shared with the VLIW processor, there are a number of *reconfigurable cells*(RC) which basically comprise FUs and RFs too(fig. 2). The FUs can be heterogeneous supporting different operation sets. To remove the control flow inside loops, the FUs support predicated operations. The distributed RFs are small with less ports. The multiplexors are used to direct data from different sources. The configuration RAM can store a few configurations locally, which can be loaded on cycle-by-cycle basis. The configurations can also be loaded from the memory hierarchy at the cost of extra delay if the local configuration RAM is not big enough. Like instructions in ISPs, the configurations control the behaviour of the basic components by selecting operations and multiplexors. The purpose of the reconfigurable matrix is to accelerate the dataflow-like kernels in a highly parallel way. The matrix also includes the FUs and RF of the VLIW processor. The access to the memory of the matrix is also performed through the VLIW processor FUs.

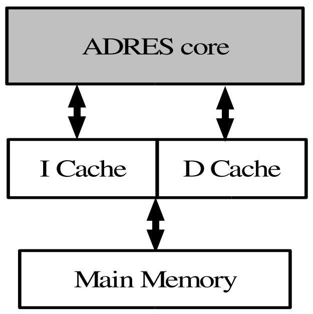

Fig. 1. ADRES system

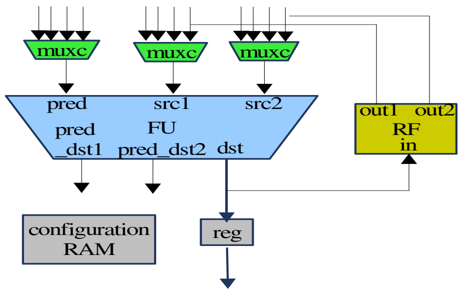

Fig. 2. Example of a Reconfigurable Cell

In fact, the ADRES is a template of architectures instead of a fixed architecture. An XML-based architecture description language is used to define the communication topology, supported operation set, resource allocation and timing of the target architecture [6]. Even the actual organization of the RC is not fixed, FUs and RFs can be put together in several ways, for example, two FUs can share one RF. The architecture shown in fig. 3 and fig. 2 is just one possible instance of the template. The specified architecture will be translated to an internal architecture representation to facilitate compilation techniques.

2.2 Improved Performance with the VLIW Processor

Many coarse-grained architectures consist of a reconfigurable matrix and a relatively slow RISC processor, e.g., TinyRisc in MorphoSys [1] and ARC in Chameleon [3]. These RISC processors execute the unaccelerated part of the application,

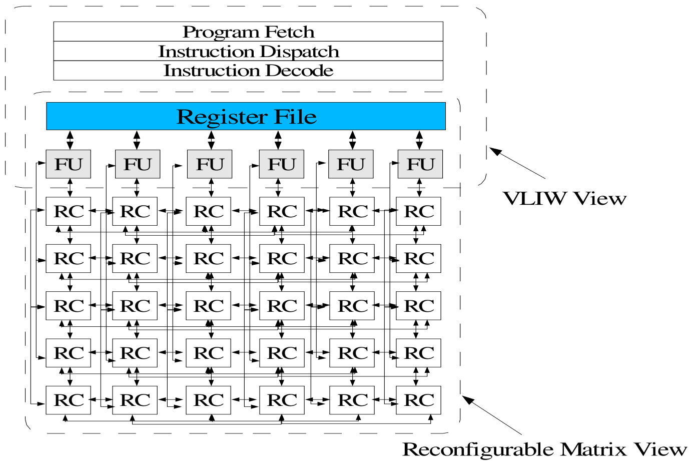

Fig. 3. ADRES core

which only represents a small portion of execution time. However, such a system architecture has some problems due to the huge performance gap between the RISC and the reconfigurable matrix. According to Amdahl's law [5], the performance gain that can be obtained by improving some portion of an application can be calculated as equation 1. Suppose the kernels, representing 90% of execution time, are mapped to the reconfigurable matrix to obtain 30 times of acceleration over the RISC processor, the overall speedup is merely 7.69. Obviously a high kernel speedup is not translated to a high overall speedup. The reason is that the unaccelerated part, which is often irregular and control-intensive, becomes a bottleneck. Speeding up this part is essential for the overall performance. Although it is hard to exploit higher parallelism for the unaccelerated part on the reconfigurable matrix, it is still possible to discover *instruction-level parallelism*(ILP) using a VLIW processor, where 2-4 times speedup over the RISC is reasonable. If we recalculate the speedup with the assumption of 3 times acceleration for the unaccelerated code, the overall acceleration is now 15.8, much better than the previous scenario. This simple calculation proves the importance of a balanced system. The VLIW processor can help to improve the overall speedup dramatically in certain circumstances.

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}} \tag{1}$$

2.3 Simplified Programming Model and Reduced Communication Cost

A simplified programming model and reduced communication cost are two important advantages of the ADRES architecture. These are achieved by making the VLIW processor and the reconfigurable matrix share access to the memory.

In traditional reconfigurable architectures, the processor and the reconfigurable matrix are essentially separated. The communication is often through explicit data copying. The normal execution steps are: (1) copy the data from the VLIW memory to that of the reconfigurable matrix; (2) the reconfigurable matrix part computes the kernel; (3) the results are copied back from the memory of the reconfigurable matrix to that of the VLIW processor. Though some techniques are adopted to reduce the data copying, e.g., wider data bus and DMA controller, the overhead is still considerable in terms of performance and energy. From the programming point of view, the separated processor and reconfigurable matrix require significant code rewriting. Starting from a software implementation to map kernels to the matrix, we have to identify the data structures used for communication and replace them with communication primitives. Data analysis should be done to make sure as few as possible data are actually copied. In addition, the kernels and the rest of the code have to be cleanly separated in such a way that no shared access to any data structure remains. These transformations are often complex and error-prone.

In the ADRES architecture, the data communication is performed through the shared RF and memory space. This feature is very helpful to map high-level language code such as C to the ADRES architecture without major changes. When a high-level language is compiled to a processor, the local variables are normally allocated in the RF, whereas the static variables and arrays are allocated in the memory space. When the control of the program is transfered between the VLIW processor and the reconfigurable matrix, those variables used for communication can stay in the RF or the memory as they were. The copying is unnecessary because both the VLIW processor and the reconfigurable matrix share access to the RF and memory hierarchy. From programming point of view, this *shared-memory* architecture is more compiler-friendly than the *message-passing* one. Moreover, the RF and memory are alternately shared instead of being simultaneously shared. This eliminates data synchronizing and integrity problems. Code doesn't require any rewriting and can be handled by compiler automatically.

2.4 Substantial Resource Sharing

Since the basic components such as the FUs and RFs of the reconfigurable matrix and those of the VLIW processor are basically the same, one natural thinking is that resources might be shared to have substantial cost-saving. In other coarse-grained reconfigurable architectures, the resources cannot be effectively shared because the processor and the reconfigurable matrix are two separated parts. For example, the FU in the TinyRisc of MorphoSys cannot work cooperatively with the reconfigurable cells in the matrix.

In the ADRES architecture, since the VLIW processor and the reconfigurable matrix are indeed two virtual functional views of the same physical entity, many resources are shared among these two parts. Due to its processor/co-processor model, only one of the VLIW processor and the reconfigurable matrix is active at any time. This fact makes the resource sharing possible. Especially, most components of the VLIW processor are reused in the reconfigurable matrix as shown in fig. 3. Although the amount of VLIW resources is only a fraction of those of the reconfigurable matrix, they are generally more powerful. For example, the FUs of the VLIW processor can execute more operations. The register file has much more ports than the counterparts in the reconfigurable matrix. In other words, the resources of the VLIW processor are substantial in terms of functionality. Reusing these resources can help to improve the performance and increase the schedulablity of kernels.

3 Adaptations of Compilation Techniques

Given the ever-increasing pressure of time-to-market and complexity of applications, the success of any new programmable architecture is more and more dependent on good design tools. For example, VLIW processors have gained huge popularity among DSP/multimedia applications although they are neither the most power- or performance-efficient ones. One important reason is that they have mature compiler support. An application written in a high-level programming language can be automatically mapped to a VLIW with reasonable quality. Compared with other coarse-grained reconfigurable architectures, the ADRES architecture is more compiler-friendly due to the simplified programming model discussed in section 2.3. However, some new compilation techniques need to be adopted to fully exploit the potential of the architecture.

3.1 Compilation Flow Overview

Previously, we have developed a compiler framework for a family of coarse-grained reconfigurable architectures [6]. A novel modulo scheduling algorithm and an abstract architecture representation were also proposed to exploit loop-level parallelism [7]. They have been adapted to the ADRES architecture. The overall compilation flow is shown in fig. 4. We use the IMPACT compiler framework [8] as a frontend to parse C source code, do some optimization and analysis, and emit the intermediate representation (IR), which is called *lcode*. Taking *lcode* as input, the compiler first tries to identify the pipelineable loops, which can be accelerated by the reconfigurable matrix. Then, the compilation process is divided into two paths that are for the VLIW processor and the reconfigurable matrix respectively. The identified loops are scheduled on the reconfigurable matrix using the modulo scheduling algorithm we developed [7]. The scheduler takes advantage of the shared resources, e.g., the multi-port VLIW register file, to maximize performance. The remaining code is mapped to the VLIW processor using regular VLIW compilation techniques, including ILP scheduling and register allocation. Afterwards, the two parts of scheduled code are put together, ready for being executed by the ADRES architecture.

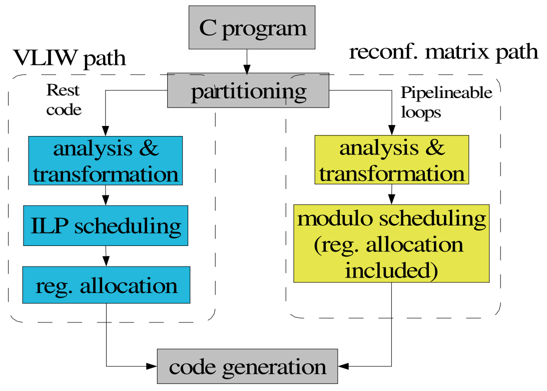

Fig. 4. Compilation Flow for the ADRES architecture

3.2 Interface Generation

The compilation techniques for the VLIW architecture are already mature and the main compilation techniques for the coarse-grained architecture were developed in our previous work. Adapted to the ADRES architecture, the most important problem is how to make the VLIW processor and the reconfigurable matrix work cooperatively and communicate with each other. Thanks to ADRES's compiler-friendly features, interface generation is indeed quite simple(fig. 5).

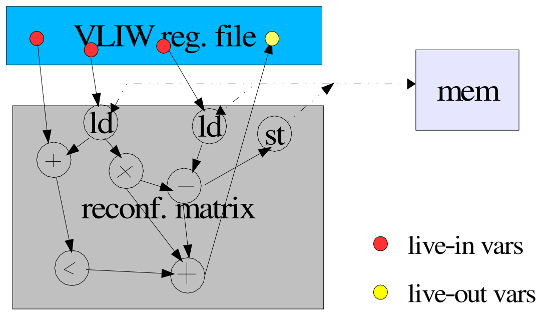

Fig. 5. Interfacing between the VLIW processor and the Reconfigurable matrix

Each loop mapped to the reconfigurable matrix has to communicate with the rest of application, e.g., taking input data and some parameters, and writing back results. As mentioned in section 2.3, the communication of the ADRES architecture is performed through shared register file and shared memory. Using dataflow analysis, the *live-in* and *live-out* variables are identified, which represents the input data and output data communicated through the shared register file. These variables will be allocated in the VLIW register file. Since these variables will occupy some register space throughout the lifetime of the loops, they are subtracted from the capacity of the VLIW register file. Therefore the scheduler won't overuse the VLIW register file for other tasks. As to the variables mapped to the memory, we don't need to do anything. The mapped loop can access the correct address through the load/store operations available on some FUs of the VLIW processor.

4 Experimental Results

For the purpose of experiment, an architecture resembling the topology of MorphoSys [1] is instantiated from the ADRES template. In this configuration, a total of 64 FUs are divided into four tiles, each of which consists of 4x4 FUs. Each FU is not only connected to the 4 nearest neighbor FUs, but also to all FUs within the same row or column in this tile. In addition, there are row buses and column buses across the matrix. The first row of FUs is also used by the VLIW processor, and are connected to a multi-port register file. Only the FUs in the first row are capable of executing memory operations, i.e., load/store operations.

The testbench consists of 4 programs, which are all derived from C reference code of TI's DSP benchmarks [9]. The *idct* is a 8x8 inverse discrete cosine transformation. The *fft* refers to a radix-4 fast Fourier transformation. The *corr* computes 3x3 correlation. The *latanal* is a lattice analysis function. They are typical multimedia and digital signal processing applications with abundant inherent parallelism.

kernel	no. of ops	live-in vars	live-out vars	II	IPC	sched. density
idct	86	18	2	3	28.7	44.8%
fft	70	19	0	3	23.3	36.5%
corr	56	25	0	2	28	43.8%
latanal	12	7	1	1	12	18.8%

Table 1. Schedule results

The schedule results are shown in table 1. The second column refers to the total number of operations within the pipelined loop body. The II is *initiation interval*, meaning the loop starts a new iteration every II cycles [10]. The live-in variables and live-out variables are allocated in the VLIW register file. For all loops, the amount of the live-in variables is quite considerable, e.g., used as coefficients for the *idct*. The live-out variables are very rare because a loop doesn't usually keep writing the same variable. The instructions-per-cycle (IPC) reflects how many operations are executed in one cycle on average. Scheduling density is equal to $IPC/No.ofFUs$. It reflects the actual utilization of all FUs for computation. The results show the IPC is pretty high, ranging from 12 to 28.7. It is well above any typical VLIW processor. The FU utilization is around 40% except *latanal*, which already achieves the minimal II.

We are still not able to show the overall speedup for the entire application because our architecture simulator is not ready yet. However, the results reflect the integration impact of the VLIW processor and the reconfigurable matrix. For example, the live-in and live-out variables are allocated on the VLIW register file, and the FUs and RF of the VLIW processor are used by the scheduler to more efficiently map kernels to the reconfigurable matrix.

5 Related Work

Many coarse-grained reconfigurable architectures have been proposed in recent years. MorphoSys [1] and REMARC [4] are typical ones consisting of a RISC processor and a fabric of reconfigurable units. For MorphoSys the communication is performed through a DMA controller and a so-called frame buffer. In REMARC, the coupling is tighter. The matrix is used as a co-processor next to the MIPS processor. Neither of these architectures has compiler support for the matrix part. Chameleon [3] is a commercial architecture that comprises an ARC processor and a reconfigurable processing fabric as well. The communication is through a 128-bit bus and a DMA controller. The data has to be copied between the two memory spaces. Compiler support is limited to the processor side. Recent work [11] integrates an open source processor and a commercial reconfigurable IP core. The communication is also through a shared bus.

Another category of reconfigurable architectures presents much tighter integration. Examples are ConCise [12], PRISC [13] and Chimaera [14]. In these architectures, the reconfigurable units are deeply embedded into the pipeline of the processor. Customized instructions are built with these reconfigurable units. The programming model is simplified compared with the previous category because resources such as memory ports and register file are exposed to both the processor and the reconfigurable units. This leads to good compiler support. However, these architectures do not have much potential for performance, constrained by limited exploitable parallelism.

6 Conclusions and Future Work

Coarse-grained reconfigurable architectures have been gaining importance recently. Many new architectures are proposed, which normally comprise a processor and a reconfigurable matrix. In this paper, we address the integration problem between the processor and the reconfigurable matrix, which has not received enough attention in the past. A new architecture called ADRES is proposed, where a VLIW processor and a reconfigurable matrix are tightly coupled in a single architecture and many resources are shared. This level of integration brings a lot of benefits, including increased performance, simplified programming model, reduced communication cost and substantial resource sharing.

Our compiler framework was adapted to the new architecture without much difficulty. It proves that the ADRES architecture is very compiler-friendly. The

VLIW compilation techniques and the compilation techniques for the reconfigurable matrix can be applied to the two parts of the ADRES architecture respectively. The partitioning and interfacing of the accelerated loops and the rest of code can be handled by the compiler without requiring code rewriting.

However, we have not implemented the ADRES architecture at the circuit level yet. Therefore, many detailed design problems have not been taken into account. And we do not have concrete figures for the area, power, etc to show the strength of the ADRES architecture in details. Hence, to implement the ADRES design is in the scope of our future work. On the other hand, we believe the compiler is even more important than the architecture. We will keep developing the compiler to refine the ADRES architecture from the compiler point of view.

References

1. Singh, H., Lee, M.H., Lu, G., Kurdahi, F.J., Bagherzadeh, N., Filho, E.M.C.: Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications. IEEE Trans. on Computers **49** (2000) 465–481
2. C. Ebeling, D. Cronquist, P.F.: RaPiD - reconfigurable pipelined datapath. In: Proc. of International Workshop on Field Programmable Logic and Applications. (1996)
3. : Chameleon Systems Inc. (2002) http://www.chameleonsystems.com.
4. Miyamori, T., Olukotun, K.: REMARC: Reconfigurable multimedia array coprocessor. In: FPGA. (1998) 261
5. Patterson, D.A., Hennessy, J.L.: Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, Inc. (1996)
6. Mei, B., Vernalde, S., Verkest, D., Man, H.D., Lauwereins, R.: DRESC: A retargetable compiler for coarse-grained reconfigurable architectures. In: International Conference on Field Programmable Technology. (2002)
7. Mei, B., Vernalde, S., Verkest, D., Man, H.D., Lauwereins, R.: Exploiting loop-level parallelism for coarse-grained reconfigurable architecture using modulo scheduling. In: Proc. Design, Automation and Test in Europe (DATE). (2003)
8. Chang, P.P., Mahlke, S.A., Chen, W.Y., Warter, N.J., Hwu, W.W.: IMPACT: An architectural framework for multiple-instruction-issue processors. In: Proceedings of the 18th International Symposium on Computer Architecture (ISCA). (1991) 266–275
9. : TI Inc. (2002) http://www.ti.com/.
10. Rau, B.R.: Iterative modulo scheduling. Technical report, Hewlett-Packard Lab: HPL-94-115 (1995)
11. Becker, J., Thomas, A., Vorbach, M., Baumgarte, V.: An industrial/academic configurable system-on-chip project(CSoC): Coarse-grain XPP-/leon-based architecture integration. In: Proc. Design Automation and Test in Europe(DATE). (2003)
12. Kastrup, B.: Automatic Synthesis of Reconfigurable Instruction Set Accelerations. PhD thesis, Eindhoven University of Technology (2001)
13. Razdan, R., Brace, K., Smith, M.D.: Prisc software acceleration techniques. In: Proc. 1994 IEEE Intl. Conf. on Computer Design. (1994)
14. Hauck, S., Fry, T.W., Hosler, M.M., Kao, J.P.: The Chimaera reconfigurable functional unit. In: Proc. the IEEE Symposium on FPGAs for Custom Computing Machines. (1997)