Mapping Applications to Coarse-Grain Reconfigurable Architectures

Mohammed Ashraful Alam Tuhin [*†]
ashraful@cs.mun.ca

Theodore S. Norvell [‡†]
theo@engr.mun.ca

October 22, 2006

Abstract

Coarse-grained reconfigurable architectures (CGRAs) are capable of achieving both goals of high performance and flexibility. CGRAs not only improve performance by exploiting the features of repetitive computations, but also can adapt to diverse computations by dynamically changing configurations of an array of its internal processing elements (PEs) and their interconnections. Many CGRAs have been developed recently as programmable coprocessors, minimizing the overhead of the central processor in many computation-intensive applications. They act as co-processors for accelerating computation-intensive portions of embedded system applications. System designers are attracted to CGRAs because they bridge the gap between Application Specific Integrated Circuits (ASICs) and microprocessors by providing the high performance of ASICs with flexibility of reconfiguration of fine-grained FPGAs. Some of the application areas of these architectures are image processing, DSP, encryption, pattern recognition, and other multimedia applications. This paper surveys the methods of compiling applications to coarse-grained reconfigurable architectures.

1 Background

Reconfigurable computing systems represent an intermediate approach between application specific integrated circuits (ASICs) and general-purpose processors. The idea driving reconfigurable computing is to avoid the **von Neumann bottleneck** (the bandwidth limitation between processor and memory) through direct computation mapping into hardware. These systems are also capable of dynamic changing of hardware logic, which it implements. Thus, an application can be partitioned temporarily for execution on the hardware which enables to execute more hardware than the existing gates to fit. Reconfigurable computing refers to systems incorporating some form of hardware programmability — customizing how the hardware is used using a number of physical control points. These control points can then be changed periodically in order to execute different applications using the same hardware.

Over the last one and a half decade, the rapid growth of computer architecture and microprocessor has brought about a radical growth in both circuit densities and speed of VLSI systems. Some computation-intensive applications, previously feasible only on supercomputers, are presently feasible on workstations and PCs. Similarly, the use of reconfigurable architecture has also extended during the past decade or two. The principle attributes of these reconfigurable architectures are the capability of dynamic mapping of a portion of a program to the hardware to exploit the implicit data parallelism in the program. Field Programmable Gate Arrays (FPGAs), which are the most widely used reconfigurable hardware, are more capable of exploiting the inherent parallelism than traditional processors. So naturally for some applications FPGA-based reconfigurable architectures perform much better than processor-based alternatives. However, for applications where the data path is coarse-grained (8 bit or more), the performance and power consumptions on FPGAs are handled inefficiently. Also the compilation time and the reconfiguration time on FPGAs are long. To overcome these disadvantages, many coarse-grained or ALU-based reconfigurable systems have been proposed as an alternative between FPGA-based systems and fixed logic CPUs. Coarse-grained

[*]Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada A1B 3X5.

[†]Supported by NSERC.

[‡]Faculty of Engineering and Applied Science, Memorial University of Newfoundland, St. John's, NL, Canada A1B 3X5.

reconfigurable architectures (CGRAs) provide massive parallelism, high computational capability and they can be configured dynamically, making them the most attractive in the years to come especially in embedded system design.

In the past one and a half decades, different types of CGRAs with different granularity, fabrics, and mapping techniques, and intended for different applications have been developed. They have identical processing elements (PEs), even though wide variation exists in the number and functionality of components and the interconnections between them. These architectures often consists of tens to hundreds of PEs intended to execute word-level operations as opposed to the bit-level ones common in FPGAs. The coarse granularity of CGRAs drastically reduces the power, area, delay, and configuration time compared with FPGAs. As a result, we have seen the emergence of a wide range of CGRAs over recent years.

Mapping applications to CGRA is a combination of assigning time cycles for operations to execute in (scheduling), mapping these operation executions to specific processing elements (allocation), and routing the operands or input data by mapping and scheduling data communications to specific interconnects in the fabric (routing). This type of mapping requires several considerations. The first two are the selection of target architecture and the appropriate programming language. The language is important because with the help of this one has to write the application that has to be mapped to the target architecture.

There has not been much work on mapping applications directly on to coarse-grained reconfigurable architectures. Although there is extensive research and even commercial tools for FPGAs and fine-grained reconfigurable architectures, the techniques developed for them are not directly applicable to CGRAs (a PE contains at least an ALU), because of the substantial differences in PEs and interconnection architectures among others. This paper investigates the works done so far on compilation approach for CGRA.

2 Coarse-grained Reconfigurable Architectures

Hartenstein surveyed the works done so far in the fields of reconfigurable computing [1]. He first briefly outlined the major aspects of various types of reconfigurable architectures and demonstrated possible methods for programming them.

According to Hartenstein mesh-based architectures arrange their processing elements in a rectangular array, featuring horizontal and vertical connections. This structure allows efficient parallelism and a good use of communication resources. However, the advantages of a mesh are traded for the need of an efficient placement and routing step. The quality of this step can have a remarkable impact on the application performance. Due to the relative low number of processing elements, the placement and routing is often much less complex than for e.g. FPGAs.

The arrangement of the processing elements encourages nearest neighbor links between adjacent elements as an obvious communication resource. Typically, longer lines are added with different lengths, which allow connections over several processing elements.

A more uncommon arrangement of processing elements is one or several linear arrays, typically featuring connections between the neighbors. This structure is motivated by the idea of mapping pipelines onto it, with each processing element featuring one pipeline stage. This works well for linear pipelines without forks. If there are forks in the pipeline, which would need a two-dimensional realization, additional routing resources are needed, which are normally provided by longer lines spanning the whole or a part of the array, often being segmented. The linear structure allows a direct mapping of pipelines, with the mentioned problems for forks.

A full crossbar switch allows arbitrary connections between processing elements, making it the most powerful communication network. The routing task is thus a simple operation.

3 Compilation on Coarse-grained Reconfigurable Architectures

3.1 Constraints for mapping

During mapping applications to coarse-grained reconfigurable architectures (CGRAs) the following constraints mentioned by Hannig et al. in their work [2, 3] need to be considered: (a) array of PEs, (b) memory, (c) interconnect structures, (d) I/O ports, (e) synchronization, and (f) reconfiguration mechanisms. These constraints are due to the limited and fixed amount of resources found in programmable devices and reconfigurable logic devices. When mapping applications to CGRAs, the above constraints make mapping quite difficult. The applications must be analyzed to detect the operations it performs. Especially the operations that are executed frequently must be identified.

A multi-mode addressing scheme is used for combining different sizes of memory. Memory can be divided into local memory as registers files within each PEs or into memory banks capable of storing hundreds of thousands of words. During data mapping the alignment and the number of the memory banks play a vital role. If the application is memory intensive rather than computer intensive, the number of memory units should be larger than ALUs. In short, it is necessary to determine the suitable number and types of functional units required for a particular application.

3.2 Mapping Techniques

There have been some efforts on compiling applications (computation intensive part, mainly loops) onto coarse-grain reconfigurable architectures (CGRAs). The various compilation techniques mainly depend on the characteristics of the particular CGRA. Several attempts taken in this regard are briefly described below.

In Garp [4], the reconfigurable hardware is a row of processing elements. The host is capable of configuring and controlling the reconfigurable array using instruction set extensions. Garp's features can be effectively utilized through automatic compilation. The compiler draws heavily from techniques used in compilers for VLIW architectures to identify Instruction Level Parallelism (ILP) in the source program, and then to schedule code partitions for execution on the array of computing elements. The processor can run at a higher clock rate as the Garp array remains separate from the main processor. This allows Garp to have the best performance on sequential code that has little ILP. Garp's array allows the merging of multiple dependent operations into a single module, reducing the critical path. The overlapping iterations on Garp don't compete for function units, thereby making scheduling much simpler.

In CHIMAERA [5], the reconfigurable hardware is a collection of programmable logic blocks organized as interconnected rows. It is a micro-architecture that integrates a reconfigurable functional unit into the pipeline of an aggressive, dynamically scheduled superscalar processor. CHIMAERA tightly couples a processor and a reconfigurable functional unit (RFU). CHIMAERA C compiler automatically generates binaries for RFU execution. It is capable of mapping a sequence of instructions into a single RFU operation. The focus of the compiler is on identifying frequently executed instruction sequences and mapping them to a Reconfigurable Functional Unit Operation (RFUO) that will execute on the reconfigurable hardware.

PipeRench [6] is an interconnection network of configurable logic block and storage elements. The approach is to analyze the application's virtual pipeline, which is mapped onto physical pipe stages to maximize execution throughput. The compiler uses a greedy place-and-route algorithm to map these pipe stages onto the reconfigurable fabric. PipeRench uses a technique called pipeline reconfiguration to improve compilation time, reconfiguration time, and forward compatibility. A reconfigurable system partitions computations between the fabric and the system's other execution units. The fabric does reconfigurable computations whereas the processor does system computations. The system performs reconfigurable computations by configuring the fabric to implement a circuit customized for each particular reconfigurable computation. The compiler embeds computations in a single static configuration rather than an instruction sequence, reducing instruction bandwidth and control overhead. However, their technique is limited to very specific architectures, and thus cannot be applied to other coarse-grained reconfigurable architectures.

The RAW micro-architecture [7] is a set of interconnected tiles, each of which contains its own program and data memories, ALUs, registers, configurable logic and a programmable switch that can support both static and dynamic routing. RAW compiler uses the SUIF compiler infrastructure. The compiler partitions the program into multiple, coarse-grained parallel threads, each of which is then mapped onto a set of tiles. The RAW compiler views the set of N tiles in a RAW machine as a collection of functional units for exploiting ILP. But the compiler can generate unoptimized code for a small set of programs.

The RaPiD architecture [8] is a field programmable architecture that allows pipelined computational structures to be created from a linear array of ALUs, registers and memories. These are interconnected and controlled using a combination of static and dynamic control. RaPiD has a linear datapath that is a different approach compared with 2-dimensional meshes of processing elements (PEs). Its Functional Units (FUS) communicate in nearest-neighbor fashion. This constraint simplifies application mapping but restricts the design space dramatically. The VLIW compiler front-end is used to transform programs written in a high-level language like C or Java to a control/data flow graph that is then scheduled to the configurable data path. The scheduling problem is formulated as a place and route problem that maps data flow graphs from the program control/data flow graph to a computing substrate comprising multiple instances of the data path unrolled in time.

Some research efforts [9, 10] have focused on generic issues and problems in compilation like optimal code partitioning, and optimal scheduling of computation kernels for maximum throughput. While [10] proposes dynamic programming to generate an optimal kernel schedule, [9] proposes an exploration algorithm to produce the optimal linear schedule of kernels in order to minimize reconfiguration overhead and maximize data reuse.

Bondalapati et al. [11] develop algorithmic techniques to map loops in a loop onto reconfigurable architectures. Their technique is granularity-neutral for reconfigurable architectures. They aim to minimize the run-time reconfiguration when the resources (e.g., PEs) in the architecture are less than what is needed to pipeline all the computations in the loop. They use heuristic algorithms to reduce the reconfigurable cost between different pipeline segments. They have similar concerns in that they try to find a better pipeline organization during mapping applications onto the architecture. However, the only architectural feature they consider is the reconfiguration resource.

Bondalapati [12] also proposed data context switching technique to maximize the throughput of DSP applications by mapping nested loops onto reconfigurable architectures. Data Context Switching overcomes the feedback dependencies by switching between different contexts of the outermost loop. It uses embedded local memory available in most reconfigurable architectures. The technique is capable of hiding the delay from the loop-carried dependency when there is an outer loop with no loop-carried dependency, by utilizing the independent data sets of the outer loop. Even though this technique was applied to a coarse-grained reconfigurable architecture, Chameleon [13], as well as an FPGA, it requires an outer loop with no loop-carried dependency to find independent data sets. Moreover, it assumes that each PE has access to an ample local memory to store the data context when the outer loop is executing other iterations, which is not valid for many coarse-grain reconfigurable architectures.

Huang and Malik [14] proposed a design methodology of a dynamically reconfigurable data path architecture, which is used as an accelerating coprocessor. Even though this work targets coarse-grained reconfigurable architecture, their reconfigurable data path is application-specific and reconfiguration is used only to switch between the loops for which the reconfigurable data path is designed.

Nikhil et al. [15, 16] devised an algorithm for automatic mapping applications (loops) to Dynamically Reconfigurable ALU Array (DRAA), a generic reconfigurable architecture template which can represent a wide range of coarse-grained reconfigurable arrays. They placed and routed the operations of a loop body onto the ALU array, to be executed in a loop-pipelined fashion. Their algorithm maximized utilization of the memory bandwidth. They did this in two steps. First they produced line-level placements by combining the operations of a given loop. Then they combined the line placements to create plane level. Their algorithm reduced the global interconnect requirements and gave near optimal mapping for several loops.

ADRES [17] is a power-efficient flexible architecture template that combines a very long instruction word (VLIW) DSP with a coarse-grain array. The array, containing many functional units, accelerates data-flow loops by exploiting high degrees of loop-level parallelism. The VLIW DSP efficiently executes the part of the code which can't achieve so large parallelism. The VLIW and the array are coupled and communicate by a shared VLIW register file.

The DRESC [18] retargetable C compiler targets both the VLIW processor and the array. Application source code can therefore be compiled directly onto the coarse-grained reconfigurable processor. The architecture template allows designers to specify the interconnection, the type and the number of functional units. The architectural flexibility of ADRES, combined with the C design flow, allows a designer to rapidly explore architectural options for an application domain. DRESC [18] framework is a novel modulo scheduling algorithm, which is capable of pipelining a loop onto the partially interconnected array to achieve high parallelism. The task of modulo scheduling is to map the program graph to the architecture graph and try to achieve optimal performance while respecting all dependencies.

KressArray [19] uses simulated annealing to simultaneously solve the placement and routing subproblems. However, it does not support multiple configurations for one loop. In KressArray, the datapath synthesis system (DPSS) maps statements of a high level language description onto the reconfigurable DataPath Architecture (rDPA). Configuring the rDPA is composed of logic optimization and technology mapping, placement and routing, and I/O scheduling.

MorphoSys [20] aims at applications which have inherent data-parallelism, high granularity, and high throughput requirements. In it an advanced processor with multi-threading may be used to enable concurrent processing of application programs by the Reconfigurable Cell (RC) array and the main processor.

4 Future Research Directions

There has not been any work so far for mapping applications solely on coarse-grained reconfigurable architectures. We are working in this direction. The application will be first written using the parallel, object oriented language proposed in [21]. The source code will then be translated into parallel program graph (PPG) [22] (PPG subsumes program dependence graphs (PDGs) [23] and conventional control flow graphs (CFGs) [24]). Sequential consistency will be used as the correctness criteria. Optimizations will be done on the PPG. Then after applying scheduling, allocation and routing on the PPG, the algorithm can be mapped to the target CGRA.

References

[1] R. Hartenstein. A Decade of Reconfigurable Computing: A Visionary Retrospective. In *Design, Automation and Test in Europe*, pages 642–649, Munich, Germany, Mar 2001. IEEE Computer Society.

[2] Frank Hannig, Hritam Dutta, and Jurgen Teich. Mapping of Regular Nested Loop Programs to Coarse-Grained Reconfigurable Arrays Constraints and Methodology. 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 3, 2004.

[3] F. Hannig, H. Dutta, and J. Teich. Regular Mapping for Coarse-grained Reconfigurable Architectures, 2004.

[4] T. Callahan, J. Hauser, and J. Wawrzynek. The Garp architecture and C compiler. *IEEE Computer*, 33(4):62–69, 2000.

[5] Zhi Alex Ye, Andreas Moshovos, Scott Hauck, and Prithviraj Banerjee. CHIMAERA: A High-Performance Architecture with a Tightly-Coupled Reconfigurable Functional Unit. In *ISCA*, pages 225–235, 2000.

[6] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matt Moe, and R. Reed Taylor. PipeRench: A Reconfigurable Architecture and Compiler. *Computer*, 33(4):70–77, 2000.

[7] Elliot Waingold, Michael Taylor, Devabhaktuni Srikrishna, Vivek Sarkar, Walter Lee, Victor Lee, Jang Kim, Matthew Frank, Peter Finch, Rajeev Barua, Jonathan Babb, Saman Amarasinghe, and Anant Agarwal. Baring It All to Software: Raw Machines. *Computer*, 30(9):86–93, 1997.

[8] Carl Ebeling, Darren C. Cronquist, Paul Franklin, Jason Secosky, and Stefan G. Berg. Mapping Applications to the RaPiD Configurable Architecture. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 106–115, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[9] R. Maestre, F. Kurdahi, N. Bagerzadeh, H. Singh, R. Hermida, and M. Fernandez. Kernel Scheduling in Reconfigurable Computing, 1999.

[10] Kiran Bondalapati, George Papavassilopoulos, and Viktor K. Prasanna. Mapping Applications onto Reconfigurable Architectures using Dynamic Programming, 1999.

[11] Kiran Bondalapati and Viktor K. Prasanna. Loop Pipelining and Optimization for Run Time Reconfiguration. *Lecture Notes in Computer Science*, 1800:906–??, 2000.

[12] Kiran Bondalapati. Parallelizing DSP Nested Loops on Reconfigurable Architectures using Data Context Switching. In *Design Automation Conference*, pages 273–276, 2001.

[13] http://www.chameleonsystems.com.

[14] Z. Huang and S. Malik. Exploiting Operation Level Parallelism through Dynamically Reconfigurable Data Paths, 2002.

[15] Jong eun Lee, Kiyoung Choi, and Nikil D. Dutt. Compilation approach for coarse-grained reconfigurable architectures. *IEEE Des. Test*, 20(1):26–33, 2003.

[16] Jong eun Lee, Kiyoung Choi, and Nikil D. Dutt. An algorithm for mapping loops onto coarse-grained reconfigurable architectures. In *LCTES '03: Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, pages 183–188, New York, NY, USA, 2003. ACM Press.

[17] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix. In *FPL*, pages 61–70, 2003.

[18] B. Mei, S. Vernalde, D. Verkest, H. Man, and R. Lauwereins. DRESC: A retargetable compiler for coarse-grained reconfigurable architectures, 2002.

[19] Reiner W. Hartenstein and Rainer Kress. A datapath synthesis system for the reconfigurable datapath architecture. In *ASP-DAC '95: Proceedings of the 1995 conference on Asia Pacific design automation (CD-ROM)*, page 77, New York, NY, USA, 1995. ACM Press.

[20] Hartej Singh, Ming-Hau Lee, Guangming Lu, Nader Bagherzadeh, Fadi J. Kurdahi, and Eliseu M. Chaves Filho. Morphosys: An integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Trans. Comput.*, 49(5):465–481, 2000.

[21] Theodore S. Norvell. Language Design for CGRA project. 2006.

[22] Vivek Sarkar and Barbara Simons. Parallel program graphs and their classification. In *Proceedings of the 6th International Workshop on Languages and Compilers for Parallel Computing*, pages 633–655, London, UK, 1994. Springer-Verlag.

[23] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Trans. Program. Lang. Syst.*, 9(3):319–349, 1987.

[24] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.