RESOURCE ALLOCATION ALGORITHM AND OPENMP EXTENSIONS FOR PARALLEL EXECUTION ON A HETEROGENEOUS RECONFIGURABLE PLATFORM

Vlad-Mihai Sima, Elena Moscu Panainte, Koen Bertels

Computer Engineering
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

ABSTRACT

In this paper, we present the compiler extensions, based on OpenMP libraries, needed for supporting parallel execution on the reconfigurable Molen platform. More specifically, we propose an ILP algorithm to map parallel applications on the target platform, assuming that for a section of the application, the designer can select from a set of hardware implementations with different area and speedup features. Based on profile information, the algorithm aims to minimize the total execution time of the running threads, taking into account the limited reconfigurable area. We show that the speedup of our algorithm compared to other related algorithms is up to 1.9x for a real application and the real hardware implementation of the kernels. We also investigate the impact of several factors such as the size of the reconfigurable area and the number of threads on our algorithm and determine the range of parameters for which the algorithm is efficient. [1]

1. INTRODUCTION

Due to the increasing complexity of computer systems and the increasing demands for faster, smaller, low cost and low price devices, the hardware and software designers investigate new strategies to efficiently use the available and usually limited resources. Reconfigurable Computing allows for a wide range of such problems to be solved in a fast, but also economic and efficient manner, as it combines the flexibility of GPP (general purpose processor) with the speedup of the (reconfigurable) hardware.

In this paper, we propose a compile-time allocation algorithm that allows the efficient use of the reconfigurable hardware for parallel OpenMP based applications. One main novelty of the algorithm is that for each kernel, it analyses a set of hardware implementations for this kernel, which in the rest of the paper are denoted as scenarios. One common example of scenarios is the implementation on the reconfigurable hardware of one, two or more iterations of a loop or of the kernel inside the loop. The algorithm aims to select the scenarios that will minimize the total execution time of the applications.

The paper is organized as follows: in Section 2 we briefly present the Molen programming paradigm for reconfigurable architectures and related work. Next, we give a real motivational example and also present the problem overview. A detailed description of the allocation algorithm is presented in Section 4. The results of the algorithm are shown in Section 5, with a comparison to other related algorithms and analyses of the factors that influence the results. In Section 6, we present conclusions and outline new research directions.

2. BACKGROUND AND RELATED WORK

The programming model for a reconfigurable platform must offer an abstraction of the available resources to the programmer, together with a model of interaction between the components. The **MOLEN programming paradigm** [1] is a paradigm that abstracts the hardware and allows the programmer and the compiler to use efficiently the underlying hardware. With only 'one time' architectural extension, the Molen programming paradigm allows for a virtually infinite number of new hardware operations to be executed on the reconfigurable hardware. For the parallel execution, the minimal architectural extension include the following instructions: SET, EXECUTE, BREAK and MOVTX and MOVFX.

OpenMP is set of compiler directives, library functions and environment variables that can be used to specify parallelism in applications developed for shared memory architecture. Because of this OpenMP is the obvious choice of parallelism to be used with Molen.

As far as area allocation for FPGA's is concerned, the **existing approaches** are related mostly to hardware/software partitioning or placement. In [2] the authors propose an optimal placement based on packing classes. Partitioning is

[1]This research is sponsored by: hArtes project (IST-035143) supported by Sixth Framework Programme of the European Community under thematic area Embedded Systems

discussed in [3] from the algorithmic point of view. Two formulations of the problem are given, and it is proven that one is NP-hard while the other has a polynomial solution. A simulated annealing algorithm is described in [4] for architectures that support just serial execution. Dynamic granularity selection is addressed in [5] where an algorithm is given to identify the possible partitions for an application. In [6], two ILP algorithms are given, which minimize the reconfiguration area by deciding if a kernel should always be configured, should run in software or just be reconfigured as needed. All these approaches have in common the fact that they consider a task can be implemented either in software or hardware.

Run-time solutions for allocation are presented in [7] and [8] where an algorithm based on early partial reconfiguration and incremental reconfiguration is presented.

3. PROBLEM OVERVIEW

Assume we have several threads that use OpenMP pragmas, for each thread, several implementations can be available, each with its specific software execution time, hardware execution time and area requirements (an example is given in Table 1 for a real application).

The solution we propose for an efficient FPGA area allocation is to select at compile-time a specific scenario for each of the threads such that the area of all the scenarios is less than the total area available. The configuration can be done just once as now there will be no conflicts.

Problem statement: We call a scenario a set of implementations for a kernel executed inside an application thread. In our context, a kernel can be composed of multiple functions, for which the execution time is an important percent of the total execution time of the thread. The scenario - s_j - will be characterized by: software execution time t_{sw}, hardware execution time t_{hw} and area occupied a. The scenarios are generated based on the available hardware implementations and profiling information. A scenario group represents a set of all available scenarios for a particular kernel K_i. Assuming we have m scenarios for kernel K_i: $SG_{K_i} = \{s_{i,j}, j = \overline{1..m} / s_{i,j} = (t_{sw}, t_{hw}, a)\}$.

Our problem can be formulated as follows: having a set of scenario groups $SGS = \{SG_{K_1}, SG_{K_2}, ...SG_{K_n}\}$ that are in conflict at runtime, and a total area S, determine the particular scenarios selection that will be used $SS = \{s_i / s_i \in SG_i\}$ that minimizes the total execution time of the running threads taking into account the parallel hardware execution on the reconfigurable hardware and the limited size of the reconfigurable area.

4. RECONFIGURABLE HARDWARE ALLOCATOR FOR OPENMP AND MOLEN

4.1. MOLEN extensions for OpenMP

The minimal case of MOLEN (see [1] for further details) that allows parallel execution just for CCUs while the CPU is stalled is not suited for multiple application scenario. For the complete case of MOLEN the execution has to be synchronized for the whole FPGA and GPP, which will imply synchronizing all the running applications. The proposed **instruction set extension** is to use a new instruction that will interrogate if a specific hardware operation has finished.

To efficiently map OpenMP applications to an architecture implementing MOLEN several tools need extensions. For the **operating system** system call will be provided for each of the MOLEN concepts: configuration, execution and parameter transfer. The details of the system calls are beyond the scope of this paper. The system calls can be generated at appropriate places by the MOLEN **compiler** which is based on GCC 4.2.

The **profiler** and **hardware estimator** play an important role as these tools need to provide to the compiler information about all kernels, and the possible implementation, enabling the compiler to take the correct decisions.

The partitioner identifies the kernels of the application that will run in parallel and will compete for reconfigurable area. Using the profiling data, several scenarios will be provided for each of the identified sections. It is the responsibility of the compiler to choose a specific scenario based on the area requirements and total execution time.

4.2. Allocation algorithm

In the rest of this section, we present a compile-time allocation algorithm for threads that compete for the reconfigurable hardware. We assume that the profiler provides the set of kernels that require simultaneously the reconfigurable hardware and we aim to select the optimal scenario for each kernel such that the total execution time of the threads is minimized. However taking into account the actual total execution time for the threads is dependent on the sequence in which the threads require the reconfigurable hardware and we do not have such information at compile time, we want to minimize the superior limit of this total execution time. For a set of scenarios with n elements the execution time is $U = \sum_{k=1}^{n}(t_{sw_k}) + \max_k(t_{hw_k})$.

For the problem defined in Section 3 we transform it in an ILP problem as follows.

0-1 selection In our case, only one scenario from a scenario group must be selected for execution. We adopt the following notations:

- n the total number of scenario groups

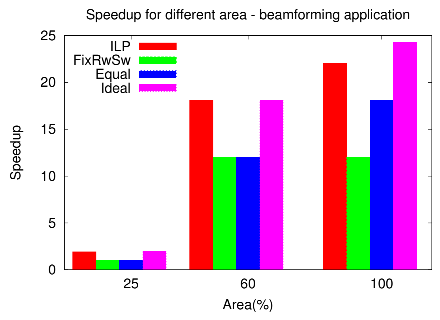

Fig. 1. Speedup compared to software execution

Thread	Scenario	SW Time (ms)	HW Time (ms)	Area (%)
1	$s_{1,1}$	1635000	0	0
	....			
	$s_{1,5}$	87325	44792	81
2	$s_{2,1}$	1570000	0	0
	....			
	$s_{2,6}$	25	13326	84

Table 1. Scenarios for beamforming applications

- m_i be the number of scenarios in scenario group i
- $S_{i,j}$ the j-th scenario from scenario group i
- $x_{i,j}$ a boolean variable such that $x_{i,j} \begin{cases} 0, S_{i,j} \notin SS \\ 1, S_{i,j} \in SS \end{cases}$.
- A the total area available

Finding the result set of scenarios is reduced to finding the values for all $x_{i,j}$.

The objective function is $\min(\sum_{i=1}^{n}(\sum_{j=1}^{m_i}(t_{sw_{i,j}} * x_{i,j})) + \max_{i=1}^{n}(\sum_{j=1}^{m_i}(t_{hw_{i,j}} * x_{i,j})))$ which can be simply expressed as:

$$\min(\sum_{i=1}^{n}\sum_{j=1}^{m_i}(t_{sw_{i,j}} * x_{i,j}) + y) \qquad (1)$$

$$\text{for each i from 1 to n: } \sum_{j=1}^{m_j}(t_{hw_{i,j}} * x_{i,j}) \leq y \qquad (2)$$

The constraints can be represented as a system of **linear pseudo-boolean inequalities**. The one scenario per scenario group constraint will be expressed for each $i = 1..n$ as: $\sum_{j=1}^{m_j} x_{i,j} = 1$.

The selected scenarios must fit together on the available reconfigurable hardware, thus we have the area constraint: $\sum_{i=1}^{n}\sum_{j=1}^{m_i}(a_{i,j} * x_{i,j}) \leq A$.

5. RESULTS

In this section, we present the results of the algorithm for two case studies: a beamforming application and synthetic results.

The **beamforming application** idea is to enhance the capabilities of sensors - in our case microphones - by jointly taking the individual signals of multiple sensors into one computation and thereby modify their spatial directivity. The application is composed of two threads: one that computes the signals for each source and one that adjusts the parameters of the computation based on the movement of the sources in space. For the computation thread the kernel is the FIR filter, executed in parallel for all the sources. The thread that performs the adjustments has a kernel represented by a matrix multiplication.

The **synthetic applications** are generated for different numbers of conflicting scenario groups. We analyzed when the number of scenario groups ranges from 2 to 7. For each case, a number of 300 problems were randomly generated, with a number of 2 to 8 scenarios per scenario group and a speedup between 2 and 20. The area used for the hardware scenarios was from 5 to 50 and the total area was proportional to the number of scenario groups. The previous parameters were chosen after analyzing the beamforming applications and various hardware kernels.

We target an **architecture** similar to Xilinx Virtex series, and we assume there is enough local memory on the FPGA so that the kernels can be executed independently without bus conflicts. For the beamforming application, we have implemented and tested each kernel individually using the DWARV tool ([9]). We used Xilinx Virtex II Pro running at 300 Mhz and the hardware designs clocked at 100 Mhz. The results for the whole execution were estimated from the profilling information available.

We compare our algorithm which is denoted in the rest of the section as *ILP* to a simple allocation solution when the total available area is equally divided between running threads - referred to as the *Equal* case and to an adapted version of one of the algorithms proposed in [6] reffered to as *FixRwSW*. The *FixRwSw* algorithm relies just on choosing from a hardware or a software version of the kernel. For all the algorithms we consider the average execution time of all possible schedules - as the threads can execute in different order, and can't be scheduled from the compile time as they may depend on inputs. We also compute the ideal case - marked *Ideal* in the graphs. The ideal case represents the minimum time that can be obtained for the best allocation and execution sequence.

The **results for the beamforming application** are summarized in Figure 1. Our algorithm performs better than both *Equal* and *FixRwSW* and is close to the ideal case. The

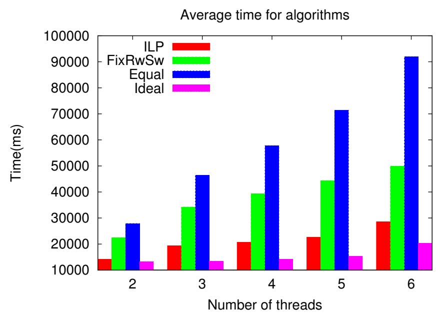

Fig. 2. Execution time for synthetic applications for different number of threads

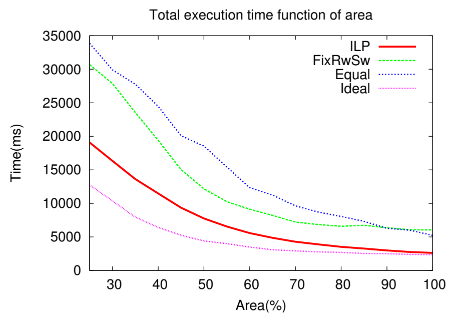

Fig. 3. Execution time vs area for synthetic applications

difference from the ideal case is due to the fact that the algorithm optimises the execution time for any order, while the ideal case considers just the best possible order.

The behavior of the algorithms when the number of threads is increased is presented in Figure 2. For all the cases the *ILP* algorithm obtains a smaller execution time for the averages of schedules.

We also investigate the impact on performance of the size of the reconfigurable hardware. For a set of 4 tasks, the results are presented in Figure 3. We notice that our algorithm converges faster than *Equal* and *ILP* to the ideal case. As *FixRwSW* doesn't consider parallelism we can see it will not improve the execution time after a certain area. Also *ILP* becomes just 1.2 times worse than *Ideal*.

From the above results we can conclude that our algorithm can be applied effectively when the available area is comparable to the area used by a scenario group in case all the iterations are implemented in parallel. However our algorithm is not suitable for small kernels which can be dynamically configured at run-time. In order to dynamically change the selected scenarios run-time algorithm must be addressed.

6. CONCLUSIONS

In this paper, we propose an allocation algorithms that selects between multiple implementations of the kernels taking into account the hardware parallel execution and the size of the available reconfigurable area. We compared the *ILP* algorithm to a simple allocation algorithm which equally divides the available area to the number of threads and estimated that for a real application and hardware implementation our algorithm has a double speedup. Our algorithm is close (up to 33%) to the ideal case (ideal sequence of threads). Finally we determine that our algorithm performs well for those cases where the size of the reconfigurable area is not large enough to fit all the largest scenarios inside. When validating these algorithms using the beamforming application, we demonstrated that the estimated speedup obtained when applying the *ILP* algorithm, is close to the ideal case.

In our future work, we will extend the ILP model with additional constraints like available memory and memory bandwidth. Additionally we will investigate the runtime algorithms for allocation and scheduling in the context of the runtime support.

7. REFERENCES

[1] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The molen polymorphic processor," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1363–1375, 2004.

[2] S. Fekete, E. Köhler, and J. Teich, "Optimal fpga module placement with temporal precedence constraints," in *DATE '01: Proceedings of the conference on Design, automation and test in Europe*. Piscataway, NJ, USA: IEEE Press, 2001, pp. 658–667.

[3] P. Arató, Z. Ádám Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 1, pp. 136–156, 2005.

[4] S. Banerjee and N. Dutt, "Efficient search space exploration for hw-sw partitioning," in *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2004, pp. 122–127.

[5] J. Henkel and R. Ernst, "An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 9, no. 2, pp. 273–290, 2001.

[6] E. M. Panainte, K. Bertels, and S. Vassiliadis, "Compiler-driven fpga-area allocation for reconfigurable computing," in *DATE '06: Proceedings of the conference on Design, automation and test in Europe*. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2006, pp. 369–374.

[7] D. A. K. . G. B. W. Martyn A. George, Mathew J. Pink, "Efficient allocation of fpga area to multiple users in an operating system for reconfigurable computing," in *In Proceedings of ERSA*, 2002, pp. 238–242.

[8] B. Jeong, S. Yoo, S. Lee, and K. Choi, "Hardware-software cosynthesis for run-time incrementally reconfigurable fpgas," in *ASP-DAC '00: Proceedings of the 2000 conference on Asia South Pacific design automation*. New York, NY, USA: ACM, 2000, pp. 169–174.

[9] Y. D. Yankova, G. Kuzmanov, K. Bertels, G. N. Gaydadjiev, J. Lu, and S. Vassiliadis, "Dwarv: Delftworkbench automated reconfigurable vhdl generator," in *In Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL07)*, August 2007, pp. 697–701.