

In 11th International Conference on Field Programmable Logic and Applications (FPL'01), Belfast, Northern Ireland, UK, August 27-29, 2001, G. Brebner, and R. Woods (Eds.), LNCS (Lecture Notes on Computer Science) 2147, Springer Verlag, pp. 523-533.

Compilation Increasing the Scheduling Scope for Multi-Memory-FPGA-based Custom Computing Machines*

João M P Cardoso¹ and Horácio C Neto²

INESC-ID, Lisboa

¹Faculty of Sciences and Technology/University of Algarve
Campus de Gambelas, 8000 117 – Faro, Portugal, email: jmpc@acm.org

²IST, Lisboa, email: hcn@inesc.pt

Abstract. This paper presents new achievements on the automatic mapping of abstract algorithms, written in imperative software programming languages, to custom computing machines. The reconfigurable hardware element of the target architecture consists of one field-programmable gate array coupled with one or more memories. The compilation flow exposes operation- and functional-level parallelism, and speculative execution. Such expositions are efficiently represented by an hierarchical model. In order to take full advantage of such representation, the scheduling scope is significantly improved by merging basic blocks at loop boundaries and by considering the parallel execution of exposed concurrent loops. The paper describes the scheduling technique, shows a study on the impact of the merge operation, and reveals the improvements achieved when the exposed parallelism is fully satisfied.

1. Introduction

One of the most challenging issues for reconfigurable computing systems is the development of mapping methods able to reduce the growing gap between design capacity and technology [1]. New field-programmable gate arrays (FPGAs) with millions of logic gates are becoming available, but high-level methods still lack.

The goal of our work is to provide a compilation framework to automatically map abstract algorithms written in software programming languages, such as Java, to a reconfigurable hardware (*reconfigware*) system [2, 3]. We believe, as other authors do [4], that efforts on mapping from high-level programs are fundamental for the success of reconfigurable computing systems, because typical users do not have the design expertise required today for developing the *reconfigware* part. Furthermore, software languages are more apt than hardware specific languages to describe applications with huge complexity [5] and their use permits the migration to *reconfigware* of already developed algorithms - mostly programmed in C/C++ and/or Java.

This article addresses some hardware compilation issues of our approach and it particularly presents a technique to enlarge the scheduling scope when generating the control unit model. The approach resorts to compilation techniques to expose inherent

* Research partially supported by the Portuguese PRAXIS XXI Program under the scope of the AXEL Project PRAXIS/2/2.1/TIT/1643/95.

In *11th International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast, Northern Ireland, UK, August 27-29, 2001, G. Brebner, and R. Woods (Eds.), LNCS (Lecture Notes on Computer Science) 2147, **Springer Verlag**, pp. 523-533.

parallelism in an abstract algorithm represented in an imperative software programming language. Such parallelism is embodied in intermediate formats. Multiple flows of control and loop-hierarchies are exposed and exhibited in those formats as well. An appropriate choice of the representation is essential for achieving high-performance results (inefficient representations may significantly hamper the results achieved by the scheduler). The goal is to take full advantage of using a specific architecture that can implement multiple-flows of control, high levels of ILP (instruction level parallelism) and functional-level parallelism. Specific hardware models permit that speculative execution implementations do not need recover, unless when dealing with operations with side-effects (memory stores), and so create a natural mode for speculative execution.

One of the mostly used representations is the traditional control-data flow graph (CDFG) [6]. It explicitly represents the control-flow structure of the input algorithm and groups operations in basic blocks (BBs). The low number of instructions in a BB in control-flow intensive applications [7] limits the performance achieved by using the CDFG. The hyperblock [8] is one of the graph representations, which prioritizes regions of the control-flow graph (CFG) to be considered as a whole. The hyperblock has been used in [9] for, mainly other goals, increasing the number of operations considered at the same time by a fine-grain scheduler. We use a hierarchical program dependence graph (HPDG), which is a representation based on the hierarchical task graph (HTG) [10] and on the Program Dependence Graph (PDG) [11]. It is constructed from the data-dependence (DDG) and merge-dependence (MDG) graphs [2, 3] and so BBs of the initial CFG emerge reordered only based on the dependences exposed. Loops are represented in hierarchical levels. Research in high-level synthesis has considered the scheduling of two concurrent loops whose bodies share functional units [12]. The approach does not use an arbitration scheme (maybe because it is unviable when sharing many functional units). It forms all the possibilities that could occur when scheduling two loops with different schedule lengths and furnishes control units to coordinate the parallel execution of such loops. Although with a different purpose, the HTG has been already used by *state-of-the-art* high-level synthesis approaches targeting ASICs [13]. This approach uses the HTG for moving operations beyond basic blocks but does not enable functional parallelism.

This paper is structured in the following sections. Section 2 describes briefly the GALADRIEL and NENYA compilers. Section 3 describes the intermediate representations and explains the scheduling by regions. Experimental results are shown in Section 4. Section 5 resumes the related work. Finally, in section 6, conclusions, future and ongoing work are enumerated.

2. Summary of the GALADRIEL & NENYA Compilers

GALADRIEL is a front-end compiler that receives the Java™ bytecodes of a given method and exposes various levels of parallelism. Such parallelism is represented in graph models (constructed using extensions to standard compilation techniques [14]) that are input to the NENYA *reconfigware* compiler [2]. NENYA is customized to reconfigurable computing systems whose architecture consists of one FPGA connected to one or more memories (RAMs). The compilation steps performed by NENYA are illustrated in Fig. 1.

In 11th *International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast, Northern Ireland, UK, August 27-29, 2001, G. Brebner, and R. Woods (Eds.), LNCS (Lecture Notes on Computer Science) 2147, **Springer Verlag**, pp. 523-533.

NENYA already integrates a number of dataflow optimizations suitable to improve the overall results: bit-width reductions, bit-constant-patterns propagation, tree-height reduction, operator strength reduction on multiplications by constants, etc. It considers the use of speculative execution for all operations (except for memory stores). The arrays in the input algorithm are mapped to memories connected to the FPGA. The base address assignment scheme used, for each array, permits to save area and reduce the delay of the address generation mechanism. Resource sharing is only considered when a bus is accessed by multiple sources (such as units interfacing to memories). When FPGA resources are insufficient to accommodate the architecture, temporal partitioning is performed [15] and multiple *reconfigware* instances are generated.

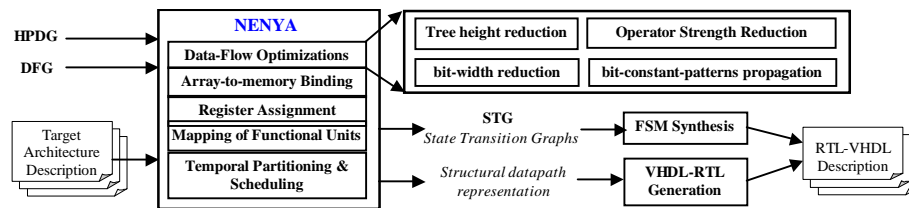


Fig. 1. Steps of the compilation flow performed by NENYA.

NENYA generates a behavioral VHDL-RTL description of each state transition graph (STG) and a structural VHDL-RTL description of each datapath. A logic synthesis tool is currently used for synthesizing the control unit circuit from the STG representations. Each datapath is entirely produced by using circuit generators. The use of circuit generators permits to have much more accurate area/performance estimations than those that are usually available by using logic synthesis (difficult to predict), and provides better results in some arithmetic operators with almost close-to-zero-compilation overhead.

3. Region-Based Scheduling

The representation models used are an HPDG and a global dataflow graph (DFG). The HPDG explicitly exhibits the reordering of BBs and loops based on its dependencies, previously exposed from the initial sequential program. Loop bodies are represented in hierarchical levels. Loop blocks that appear side-by-side in the same level of the HPDG can be simultaneously executed (see Loop.1 and Loop.4 in Fig.2b). Such loop blocks contain BBs or/and another loop blocks. At the bottom level, the HPDG contains only BBs. A global DFG represents the original code at the operation level and so exhibiting operation-level parallelism. The DFG uses multiplexer nodes in merge-dependence points, selection logic on the select signals of the multiplexers, and predicated logic on store operations existent in conditional paths, among other operations. The control unit manages execution of such store operations based on predicate's evaluation.

Each block in the HPDG is linked to the related set of the nodes on the DFG. Each DFG node is annotated with the execution delay of the component in the library that

In 11th *International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast, Northern Ireland, UK, August 27-29, 2001, G. Brebner, and R. Woods (Eds.), LNCS (Lecture Notes on Computer Science) 2147, **Springer Verlag**, pp. 523-533.

implements the correspondent operation, considering the bit-width of operands and results. From the HPDG and the global DFG the scheduling phase outputs STGs responsible for the orchestration of the datapaths (see the block diagram in Fig. 2c).

Although the scheduler does not consider resource sharing, a temporal partitioning scheme is done, whenever needed, at the top level of the HPDG [3].

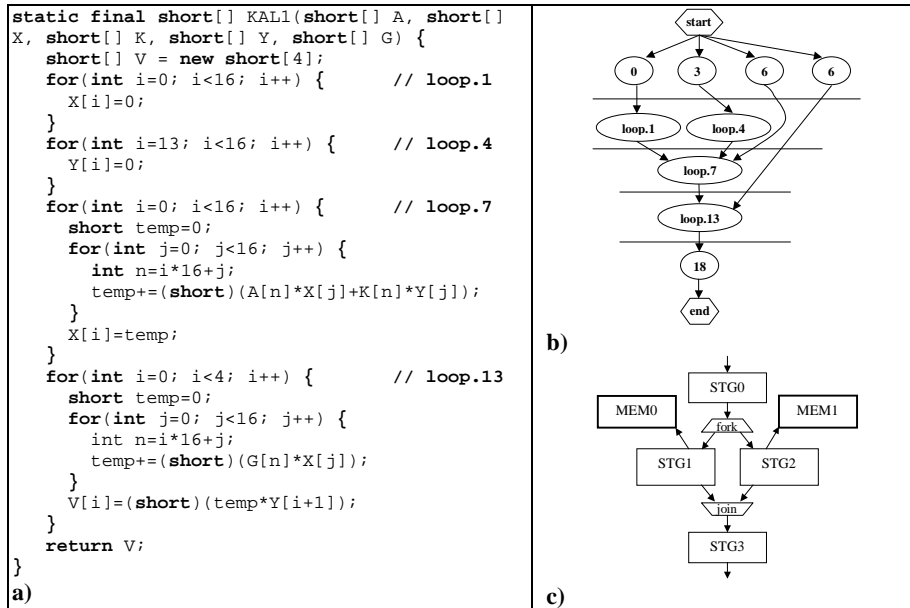


Fig. 2. Kalman example: a) Java code; b) Top level of the HPDG; c) Block diagram of STGs.

Usual scheduling schemes consider, at each time, only operations in the same BB of the CDFG. Therefore, they produce a schedule that, although might consider the parallel execution of operations inside BBs, serializes the states according to the original sequence of BBs in the CDFG/CFG. Such schemes are unable to take advantage of all the inherent parallelism exposed.

As has been already explained, in our approach operation-, BB-, and functional-level parallelism are exposed and explicitly exhibited in the HPDG. A scheduler should consider, whenever possible, the image of a whole group of BBs (herein, designated by region) in the global DFG to generate datapath solutions with high-performance. The capacity to deal with regions is an important scheme to improve scheduling results. Fig.3 shows a simple example illustrating the effect on considering a region during scheduling. Considering three clock cycles for the latency of each multiplier and one clock cycle for the other operations, the region reduces the schedule length by one clock step. Region based scheduling may have even more impact when two or more BBs and/or loop blocks in a region can be executed in parallel.

Regions are created hierarchically among loop boundaries (i.e., when a loop node is found in the HPDG all the region up to that point is scheduled) using an ASAP (as soon as possible) scheme. In Fig.2b the existent loops define the boundaries of the regions. The scheduler uses, for each region on the HPDG, the related section on the

In 11th *International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast, Northern Ireland, UK, August 27-29, 2001, G. Brebner, and R. Woods (Eds.), LNCS (Lecture Notes on Computer Science) 2147, **Springer Verlag**, pp. 523-533.

global DFG. The core of the scheduler, when scheduling each region, is based on the static-list scheduling algorithm [6]. The role of the scheduler is to generate STGs that model a control unit for orchestrating memory accesses and for executing loops (see Fig.2c). Currently, the scheduling scheme does not consider the parallel execution of two or more concurrent loops when they have accesses to the same single-port memory (an arbitration scheme needs to be addressed). When two or more loops can be simultaneously executed, independent STGs are created and a fork-join scheme is used to activate and synchronize them.

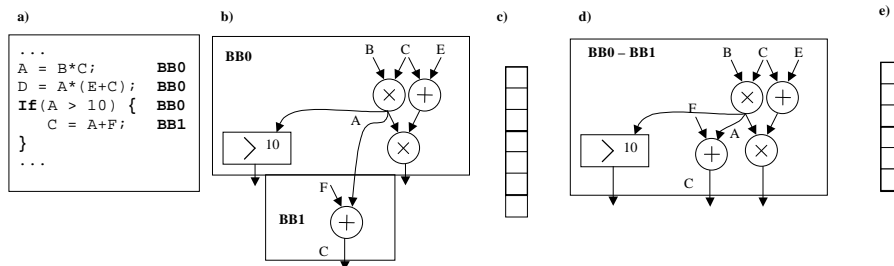


Fig. 3. Source code; b) BBs and each DFG; c) Scheduling at the BB level (7 steps); d) Creation of a region by merging BBs, and the global DFG; e) Scheduling the region (6 steps).

4. Experimental Results

In order to show the effectiveness of the approach, a number of test examples was considered and the results are reported here. All the results report estimations obtained by NENYA, from a description of each operation existent in the source code considering the implementation on the XC6000 family of FPGAs [15] (not considering interconnection costs).

The examples considered have low to medium complexity (maximum of 6 arrays and 6 loops) and all of them were automatically compiled from Java without any modification of the original code (except in the 5 examples where loops have been manually unrolled).

SQRT is a sqrt function (input of 12-bits: output of 6-bits), USQRT is another sqrt function (32: 16-bits). CRC-32 is the **Cyclic Redundancy Check** of 32-bits using the polynomial of the Ethernet protocol. QRS refers to the loop body of the benchmark from [16]. HAMMING is a hamming decoder (input of 7-bits and output of 4-bits). MASKBITS is an image processing algorithm that masks an input image (512×512) with a mask of equal dimension. CCW and INTERSECT are two geometric functions [17]. BPIC is a binary pattern image encoder [18]. FPBI is a low pass image filter with a window of 3×3 pixels. IDEA is a submodule of the **International Data Encryption Algorithm** (accepts 8 input bytes, a 52 element sub-key of 16-bit words and returns 8-bytes). SOBEL is the sobel operator over an image and performs horizontal and vertical convolutions. KAL1 refers to the Kalman filter [13]. DCT is the Discrete Cosine Transform and is performed on blocks of 8×8 elements (*short* type). DCT1, BPIC1-3, and FPBI1-2 are versions of the above examples with some

In 11th *International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast, Northern Ireland, UK, August 27-29, 2001, G. Brebner, and R. Woods (Eds.), LNCS (Lecture Notes on Computer Science) 2147, **Springer Verlag**, pp. 523-533.

loops unrolled. We group the examples in two sets: the 1st set has more *if-then-else* structures [SQRT to INTERSECT] and the 2nd set is more general [BPIC to KAL1].

Results represent the estimated schedule length (esl) of the longest path. Fig.4 shows the effect on the esl using the HPDG versus the CDFG approaches.

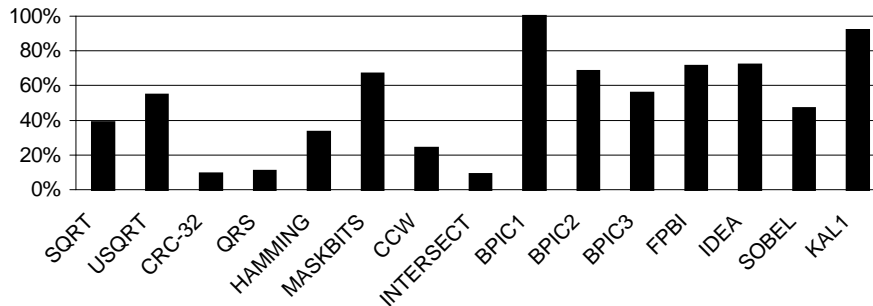


Fig. 4. Schedule lengths estimated from the HPDG normalized to the CDFG.

Considering the estimated scheduling lengths with the first set of examples:

- For three examples (CRC-32, QRS and INTERSECT) the esl with the HPDG is less than 10% of the esl obtained with the use of the CDFG. For all the set the esl with the HPDG is less than 70% of the CDFG.
- As far as the increase on the number of operations executed in parallel during a state of the control unit is concerned: for USQRT an increase from 12 to 54 is obtained, and for INTERSECT from 5 to 65. For the other examples the increase was lower but never insignificant.

Considering the estimated scheduling lengths with the second set of examples:

- For BPIC1 there is no improvement, for KAL1 the esl for HPDG is about 90% of the CDFG. For all the other examples the esl with the HPDG is less than 70% of the CDFG.
- The most significant increases on the number of operations executed in a state of the control unit are obtained for BPIC2: from 37 to 53, BPIC3: from 120 to 134, and SOBEL: from 10 to 16.

The results summarized above show important performance gains when using the HPDG with the scheduling scope increase. Next, the memory access bottleneck is quantified in some examples. We compare the esl when considering only one single-port external memory attached to the FPGA with the esl when considering no constraints about the number of ports of such memory. Fig.5 shows the resultant decrease in performance when considering only a single-port. When unrolling is used (e.g., BPIC1-2, FPBI1-3, and DCT1) the decrease in performance is even more accentuated.

In 11th *International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast, Northern Ireland, UK, August 27-29, 2001, G. Brebner, and R. Woods (Eds.), LNCS (Lecture Notes on Computer Science) 2147, **Springer Verlag**, pp. 523-533.

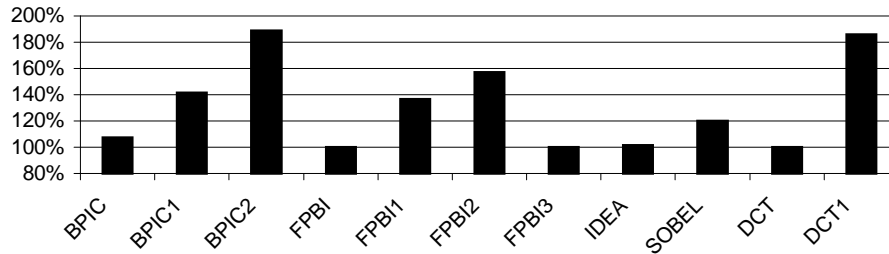
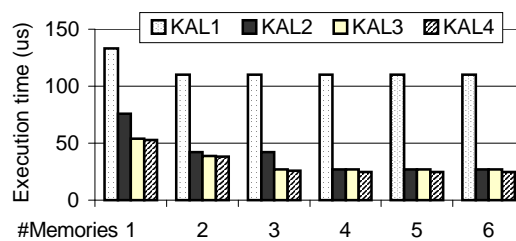


Fig. 5. Increase on the estimated schedule lengths (values larger than 100%) with a single-port memory compared with the use of a multiport memory.

For evaluation of the impact of the number of memories attached to the FPGA the following versions of the Kalman example (the example uses 6 arrays) are considered (see the main characteristics in Fig.6a): original version (KAL1); with the two inner loops unrolled (KAL2); with the first two loops and the two inner loops unrolled (KAL3); with the first two loops, the inner loop of the third loop and the last two loops unrolled (KAL4). Fig. 6b shows the esl for each version when increasing the number of memories attached to the FPGA. The best schedules need 2 memories for KAL1, 3 memories for KAL3 and 4 memories for KAL2 and KAL4. The figure also shows the improvement when considering loop unrolling. It can be seen that, for this example, speed-ups of almost 7 are achieved by the unrolling versions, considering various single-port memories, over the original version (KAL1) with a single-port memory.

Version	KAL1	KAL2	KAL3	KAL4
# Lo Java	30	96	135	239
# JVM Inst.	118	694	770	1,417
# BBs	19	13	7	4
# DFG Nodes	108	438	443	656
# loops	6	4	2	1
# memory accesses	1,191	1,191	853	853

a)



b)

Fig. 6. Four versions of the Kalman example: a) Main characteristics; b) Estimated execution times.

5. Related Work

In this section we resume some research on compiling for reconfigurable computing systems. Work on compilers targeting commercial FPGAs from abstract algorithms written in imperative software programming languages is emphasized since it is closer to the work described in this paper. Work on mapping methods using software

In *11th International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast, Northern Ireland, UK, August 27-29, 2001, G. Brebner, and R. Woods (Eds.), LNCS (Lecture Notes on Computer Science) 2147, **Springer Verlag**, pp. 523-533.

programming languages enabled for describing hardware-like semantics and structures are not considered.

PRISM-I [21], II [22] and the Transmogriifier C [23] compilers are one of the first compilation approaches for FPGA-based reconfigurable computing systems. Such approaches have identified important issues on compiling for reconfigurable computing but simple representation models have been used. Some authors adapted traditional high-level synthesis approaches to architectural synthesis for reconfigurable computing systems [24].

CAMERON [25], CHAMPION [26] and MATCH [27] are recent approaches that resort to a commercial logic synthesis tool for generating the hardware structure to be implemented in FPGA-based systems. Both CAMERON and CHAMPION are tailored for image processing applications. CAMERON starts from a single-assignment C language (SA-C). CHAMPION starts from the Cantata graphical programming environment. MATCH accepts MATLAB™ descriptions and focuses on important aspects of mapping tasks to system resources (FPGAs, microprocessors, DSPs, etc.), but uses rudimentary approaches to compile to FPGAs (uses a direct translation of the abstract syntax tree to behavioral RTL-VHDL).

Important aspects of pipelining in applications with regular loops (constant loop bounds) and array indexing (using affine index access functions) have been studied in [28] and [29] in the context of compilation from C to FPGA-based systems. [30] shows pipeline techniques for loops and recursive programs.

Many research efforts are being tailored to architectures of processing elements grouping a microprocessor with *reconfigware* in the same chip. Most of them research compilation techniques targeting their own architectures. The most remarkable are Garp-C [8] and NAPA-C [27] compilers, just to name a few.

The efforts of high-level synthesis from software programming languages have been mostly tailored to C. However, the low level used in the C language to manipulate memory contents, such as the use of pointers to fine-grain physical locations, can hinder static resolution and incapacitate a significant number of optimizations. Furthermore, the research targeting ASICs differs mainly from the research targeting FPGA-based systems on both the fixed architecture layer of the FPGAs and its programmable facility.

Our work differs from all the above approaches on several points. The one emphasized in this paper, claims the use of intermediate representations that explicitly expose high levels of parallelism and multiple-flows of control, and by increasing the scope of the scheduler, the compiler is able to satisfy the exposed parallelism.

6. Conclusions and Future Work

This paper presents methods for reconfigurable hardware compilation from a high-level imperative software programming language. The mapping methods target systems with one FPGA coupled with one or more memories. The approach uses an intermediate representation model, which allows the fully exploitation of the exposed parallelism from an input abstract algorithm. The representations used explicitly exhibit multiple flows of control, speculative execution, and parallelism at the operation and functional levels. A region-based scheduling scheme is described that tries to satisfy such levels of parallelism.

In *11th International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast, Northern Ireland, UK, August 27-29, 2001, G. Brebner, and R. Woods (Eds.), LNCS (Lecture Notes on Computer Science) 2147, **Springer Verlag**, pp. 523-533.

The results show the importance of the approach as far as performance is concerned. Performance increases are shown to be provided by the new scheduling technique when comparing it to basic block based schedulers.

Future work will address software pipelining, efficient arrays to memory binding algorithms (including distributed on-chip memory banks), and arbitration schemes to allow the parallel execution of loops when the same memory port is shared among concurrent loops. Work on the backend for retargeting other FPGA devices will be also conducted since its importance from a compilation point of view. Another important aspect that needs to be improved is the capability to share large hardware resources (e.g. multipliers) and to consider specific treatments on the presence of mutually-exclusive paths.

7. References

1. J. Becker, R. Hartenstein, M. Herz, U. Nageldinger, "Parallelization in Co-Compilation for Configurable Accelerators," In *Proc. of the Asia South Pacific Design Automation Conference (ASP-DAC'98)*, Yokohama, Japan, February 10-13.
2. João M. P. Cardoso, and Horácio C. Neto, "Macro-Based Hardware Compilation of Java™ Bytecodes into a Dynamic Reconfigurable Computing System," In *Proc. of the IEEE 7th Symposium on Field-Programmable Custom Computing Machines (FCCM'99)*, Napa Valley, CA, USA, April 21 - 23, 1999, pp. 2-11.
3. João M. P. Cardoso, *Compilation of Java™ Algorithms for Reconfigurable Computing Systems with Exploitation of Operation-Level Parallelism*, Ph.D. thesis (in portuguese), IST (*Instituto Superior Técnico*), Lisbon, Portugal, October 2000.
4. Scott Hauck, "The Future of Reconfigurable Systems," *Keynote Address, 5th Canadian Conference on Field Programmable Devices*, Montreal, June 1998.
5. J. Babb et al, "Parallelizing Applications into Silicon," In *Proc. of the 7th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'99)*, Napa Valley, CA, USA, April 21-23, 1999, pp. 70-80.
6. D. Gajski, et al., *High-Level Synthesis, Introduction to Chip and System Design*, Kluwer Academic Publishers, 1992.
7. J. A. Fisher, and B. R. Rau, "Instruction-Level Parallel Processing," In H. C. Torng and S. Vassiliadis, editors, *Instruction-Level Parallel Processors*, IEEE Computer Society Press, 1995, pp. 41-49.
8. Scott A. Mahlke, David C. Lin, William Y. Chen, Richard E. Hank, Roger A. Bringmann, "Effective Compiler Support for Predicated Execution Using the Hyperblock," In *Proc. of the 25th International Symposium on Microarchitecture*, Dec. 1992, pp. 45-54.
9. Timothy J. Callahan, John Hauser, and John Wawrzyniek, "The Garp Architecture and C Compiler," *IEEE Computer*, Vol.33, No. 4, April 2000, pp. 62-69.
10. M. Girkar, and C. D. Polychronopoulos, "Automatic Extraction of Functional Parallelism from ordinary Programs," In *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 2, March 1992, pp. 166-178.
11. J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The Program Dependence Graph and its uses in optimization," *ACM Transactions on Programming Languages and Systems*, vol. 9, no. 3, July 1987, pp. 319-349.
12. G. Lakshminarayana, K. S. Khouri, and N. K. Jha, "Wavesched: A Novel Scheduling Technique for Control-Flow Intensive Designs," In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 5, May 1999, pp. 505-523.

In 11th *International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast, Northern Ireland, UK, August 27-29, 2001, G. Brebner, and R. Woods (Eds.), LNCS (Lecture Notes on Computer Science) 2147, **Springer Verlag**, pp. 523-533.

13. S. Gupta, N. Savoiu, S. Kim, N.D. Dutt, R.K. Gupta, A. Nicolau, "Speculation Techniques for High Level synthesis of Control Intensive Designs," in *Proc. of the 38th Design Automation Conference (DAC'01)*, Las Vegas, Nevada, USA, June 18-22, 2001.
14. S. S. Muchnick, *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1997.
15. João M. P. Cardoso, and Horácio C. Neto, "An Enhanced Static-List Scheduling Algorithm for Temporal Partitioning onto RPU's," In *Proc. of the IFIP X Intl. Conference on Very Large Scale Integration (VLSI'99)*, Lisbon, December 1-3, 1999, pp. 485-496.
16. Cleland. O. Newton, "A Synthesis Process applied to the Kalman Filter Benchmark," In *Proc. of the Workshop on High-Level Synthesis*, DRA Malvern, UK. <ftp://ftp.ics.uci.edu/pub/hlsynth/HLSynth92/kalman>
17. Xilinx Inc., *XC6000 Field Programmable Gate Arrays*, v.1.10, April 24, 1997.
18. ____, Benchmarks repository, Workshop on High-Level Synthesis, 1995, <ftp://ftp.ics.uci.edu/pub/hlsynth/HLSynth95>
19. R. Sedgewick, *Algorithms in C++*, Addison-Wesley, Publishing Company, Inc., 1992.
20. Morse Rodriguez, "Evaluating Video Codecs," In *IEEE Multimedia*, Fall 1994, pp. 25-33.
21. P. Athanas, H. Silverman, "Processor Reconfiguration through Instruction-Set Metamorphosis: Architecture and Compiler," *IEEE Computer*, vol. 26, n. 3, March 1993.
22. L. Agarwal, et al., "An Asynchronous Approach to Efficient Execution of Programs on Adaptive Architectures Utilizing FPGAs," In *Proc. of the 2nd IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'94)*, Napa Valley, CA, USA, April 1994.
23. D. Galloway, "The Transmogripher C Hardware Description Language and Compiler for FPGAs," In *Proc. of the 3rd IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'95)*, Napa Valley, CA, USA, April 1995.
24. Ouais, et al., "An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures," In *Proc. of the Reconfigurable Architectures Workshop (RAW'98)*, Orlando, Florida, USA, March 30, 1998.
25. J. Hammes, R. Rinker, W. Böhm, W. Najjar, B. Draper, R. Beveridge, "Cameron: High Level Language Compilation for Reconfigurable Systems," In *Proc. of the Int. Conference on Parallel Architectures and Compilation Techniques (PACT'99)*, Newport Beach, CA, USA, Oct. 12-16, 1999.
26. Sze-Wei Ong, et al., "Automatic Mapping of Multiple Applications to Multiple Adaptive Computing Systems," In *IEEE 9th Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*, Rohnert Park, California, USA, April 30 – May 2, 2001.
27. P. Banerjee, et al., "A MATLAB Compiler For Distributed, Heterogeneous, Reconfigurable Computing Systems," In *Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, Napa Valley, CA, USA, Apr. 17-19, 2000, pp. 39-48.
28. M. Weinhardt, and W. Luk, "Pipeline Vectorization," In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2. Feb. 2001, pp. 234-233.
29. Pedro Diniz, and Joonseok Park, "Automatic Synthesis of Data Storage and Control Structures for FPGA-based Computing Engines," In *Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, Napa Valley, CA, USA, Apr. 17-19, 2000, pp. 91-100.
30. Tsutomu Maruyama, and Tsutomu Hoshino, "A C to HDL compiler for pipeline processing on FPGAs," In *Proc. of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, Napa Valley, CA, USA, Apr. 17-19, 2000, pp. 91-100.
31. Maya Gokhale, and Janice M. Stone, "Napa C: Compiling for a Hybrid/FPGA Architecture," In *Proc. of the IEEE 6th Symposium on Field-Programmable Custom Computing Machines (FCCM'98)*, Napa Valley, CA, USA, April 1998, pp. 126-135.