Reiner W. Hartenstein, Jürgen Becker, et al.: A Parallelizing Compilation Method for the Map-oriented Machine; Proceedings of Int. Conf. on Application Specific Array Processors, Strasbourg, France, July 1995

A Parallelizing Compilation Method for the Map-oriented Machine

Reiner W. Hartenstein, Jürgen Becker, Rainer Kress, Helmut Reinig, Karin Schmidt

University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de

Abstract

The paper introduces a novel parallelizing compilation method for the MoM. The MoM (Map-oriented Machine) is an Xputer architecture featuring multiple data sequencers and "soft ALUs". The compiler accepts C-source, which are restructured and partitioned into structural and sequential code providing parallelism at expression and statement level.

1. Introduction

Today we are facing increasingly complex tasks to be performed by computers. Many of these tasks are computation-intensive requiring a huge amount of data throughput and high performance. From empirical studies [9] follows that the major amount in computation time is due to rather simple loop constructs. Since additionally these loop constructs are combined with indexed array data structures, ordinary von Neumann style computers are burdened with mainly addressing computations rather than actual data manipulations. First efforts to reduce addressing overhead and to introduce parallelism have been done by the development of supercomputers [2], [7], and the development of parallelizing compilers [4], [10]. But the exploitation of inherent parallelism is restricted by the hardware structure of the target machines.

A new machine combining the advantages of both structural programming and traditional von Neumann style procedural programming has been introduced by the architectural class of Xputers [1], a data-parallel machine with shared memory. Field-programmable logic is used to offer configurable instructions which allow a fully parallel execution in contrast to e.g. vector and other parallel computers which are mostly working in a pipelined manner.

Thus several requirements arise for the development of a new Xputer compilation method. As input language the imperative language C has been taken. A fine grained parallelism at statement and expression level has been achieved in order to enable the exploitation of the reconfigurable ALU (rALU). A second major issue in Xputer program compilation is the extraction of the program's data and its mapping in a regular way over the Xputer memory space. This data arrangement together with the extracted data dependencies determine the required data sequencing and thus substantially contribute to the efficiency and performance of the program execution [1]. A third issue is the synthesis of the structural code onto the rALU.

Before the parallelizing compilation method is explained, the target hardware is briefly sketched by introducing the Map-oriented Machine 3 (MoM-3).

2. The Map-oriented Machine

The new architectural class of Xputers [1] is especially designed to reduce the von Neumann bottleneck of repetitive decoding and address interpreting. This bottleneck contributes a significant amount to the run time of algorithms out of these areas (90% in image processing, 58% in DSP [1]). Although the actual prototype MoM-3 may serve as stand-alone machine it is currently embedded as a general-purpose co-processor in a VMEbus based workstation. After setup, the MoM-3 runs independently from the host computer until the complete application is processed. Setup in this case means, that the host software has to load the application data into the MoM-3 data memory, to load the parameter sets of the generic address generators, the rALU configuration code and the program for the MoM-3 controller into the control memory and to initiate execution.

The MoM-3 supports up to seven generic address generators (GAGs), each with its own segment of data memory and a rALU subnet on the same board, called computational module (C-module). The rALU subnets of all C-modules are connected to allow propagation of interim results to the next board (figure 1). That way complex operations, which require more resources than a single rALU

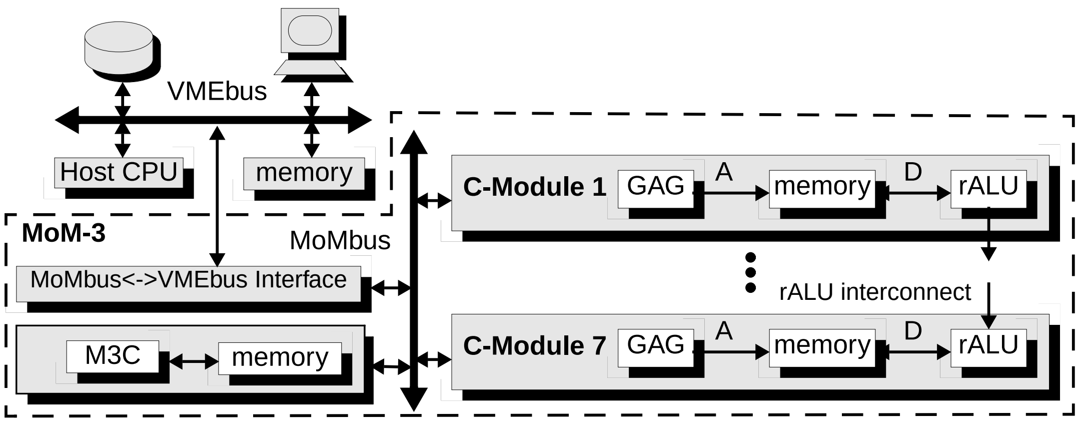

Figure 1. MoM-3 block diagram

subnet can provide, can be done on multiple modules. As long as the data required by a rALU subnet resides in the memory segment on the same C-module, data accesses can be done in parallel to data accesses on the other modules. Otherwise, data is transferred on the common MoMbus, which enforces a sequentialization of non-local data accesses. The rALU is based on field-programmable logic (FPL). In the MoM the reconfigurable datapath architecture (rDPA) is used, an FPL architecture with better throughput and higher area efficiency than FPL available commercially [6]. The (re-)configuration of GAGs and rALU subnets is done by a special MoM-3 controller circuit (M3C). It holds all configuration data for a complete application in its memory and is able to switch between configurations by downloading them to the GAGs or rALU subnets. The MoM-3 operates as a kind of configurable co-processor to a host computer. All I/O operations are done by the host as well as the memory management. The MoM-3 is a merely computational device to accelerate time-critical parts of algorithms. The host has direct access to all memory on the MoM-3, and on the other hand the MoM-3 can access data in the host's memory, though only sequentially due to the single bus.

3. The Compilation Method

A partitioning, restructuring, and mapping method is needed to translate a sequential C program into code which can be executed on an Xputer. This paradigm switch shall be performed without further user interaction. The method itself deals with the fundamental problems similar to those in compiling a program for parallel execution on a multiprocessor system. These problems are: (1) Identify and extract potential parallelism, (2) partition the program into a sequence of execution units according to the granularity of the architecture and the hardware constraints, (3) compute an efficient allocation scheme for the data in the Xputer data map, and (4) synthesize the structural code into the rALU and the remaining procedural code as parameters into the data sequencer hardware.

For Xputer compilation all these problems have to be solved during compile time. First a theory is needed for the program partitioning and restructuring (parallelization). The result of this step is the determination of a partial execution sequence. Secondly the program's data has to be mapped in a regular way onto the 2-dimensionally organized Xputer data map, followed by a computation of the right address accesses (data sequencing) for each variable. Thus far all steps are target-hardware independent. Code generation for the MoM-3 results (1) in a *hardware image file* containing structural information for the configuration of the rALU, and (2) in a *software image file* containing the parameter sets for the data sequencer hardware, especially the generic address generators.

3.1 Determination of a Program Execution Sequence

The result of the parsing of the program is a graphical representation G = (N, E) with node set N and arc set E. The control flow of the program has to be partitioned first, in order to find parallelizable subgraphs. This step is followed by a data partitioning by partial vectorization [10].

Partitioning of the Control-Flow. Given is the program graph G = (N, E). The node set N has to be partitioned, resulting in a number of subgraphs G_k, 1 k n, and an arc set E*, defining a partial execution order. For the structure of the subgraphs an additional criterion has to be formulated, namely that each subgraph has to be convex. The question is now, how the partitioning of a graph into convex subgraphs can be achieved. This is performed by specifying an equivalence relation on the node set N and building the corresponding equivalence classes which represent convex subgraphs. The method for control flow partitioning results in a coarse grained block sequence with a partial execution order [8]. These blocks are still target-hardware independent.

Partitioning of the Data-Flow. The goal of the next compilation step is to maximally parallelize each of the determined blocks in the sequence. Therefore the subgraphs are then maximally vectorized by using the *Allen-Kennedy Vectorization Algorithm* [3]. The result is a new sequence of maximally parallelized blocks containing a partial execution order. Vectorization then generates a maximum degree of parallelism in a statement block of a loop nest for statements having no dependencies or not being part of a recurrence, nor member of a cycle [3]. Here the special Xputer granularity is taken into consideration and the hardware resources are exploited optimally [8]. This is the key for achieving a high performance during Xputer program execution.

3.2 Data Mapping and Data Aligning

The next step in compilation is to decide how the program data (variables, arrays, …) can be mapped onto the two-dimensionally organized Xputer data map $DM = \{DM_x, DM_y\}$ in a regular fashion, and how the data fields of differently mapped data variables can be aligned to a combined data field in order to use only one scan pattern.

Planarization. The two-dimensional data map *DM* is in contrast to the defined arrays which have higher dimensions. This leads to the *mapping problem* resulting in the definition of a data allocation scheme. The target hardware parameters (e.g. seven GAGs are available for the MoM 3) have to be fulfilled. This leads to the *data alignment problem*. Unrolling the dimensions *I* of a variable *A* defined to be d-dimensional, d > 2, and $d \in \mathbf{N}$, means to determine a function *dmap* from the index domain of the data object to the two-dimensional index domain of an Xputer data map DM, by

$$\text{dmap:}\; I_i^A \rightarrow \{DM_x, DM_y\}\text{, with 1 i n.} \qquad (1)$$

The dimensions in the array definition are numbered from the right to the left and are then mapped. Even numbers are mapped onto the x-coordinate (DM_x), odd numbers onto the y-coordinate (DM_y) of the Xputers data map DM. Xputer dimension mapping is a planarization [8].

3.3 Determination of the Data Sequencing

The accessing of the program data variables by their indices is needed for the generation of scan patterns from which the parameter sets for the data sequencer are derived. This results in the determination of an access sequence for each variable according to their indices together with their data fields, which have been mapped into a two-dimensional form. The access sequences then can be used for a computation of corresponding scan patterns and parameter sets. The computation of an access sequence is influenced by the mapping, the alignment, the index expressions, and the according loop limits (upper, lower, step width) [8].

3.4 rALU Synthesis

The datapath synthesis system (DPSS) allows to map statements from a high level language description onto the rDPA. The statements may contain arithmetic or logic expressions, conditions, and loops, that evaluate iterative computations on a small number of input data.

The task of configuring the rDPA is carried out in the following four phases: logic optimization and technology mapping, placement and routing, I/O scheduling, and finally the code generation. Partitioning of the statements onto the different rDPA chips is not necessary since the array of rDPA chips appears as one array of DPUs with transparent chip boundaries [6].

4. Conclusions

The paper has sketched the MoM 3, a prototype of a new data-parallel machine, called Xputer, and proposed a parallelizing compilation method for this machine. The method combines structural and procedural programming, according to the Xputer paradigm of a data sequencer hardware and an FPGA-based reconfigurable ALU. Due to the data sequencing, avoiding repetitive address computation, high performance factors can be achieved [1], [5]. The parallelizing compilation method realizes the paradigm shift from von Neumann paradigm (imposed by the choice of C as input language) to the Xputer computing principles without further user interaction. The method compiles such that the special Xputer fine granularity is achieved together with a high hardware exploitation of the available resources. This fact guarantees high acceleration gains. The implementation of the proposed compilation method is completed. The MoM-3 is currently being built. The address generator chips, the MoM-3 controller chip and the rALU control chip have returned from fabrication and are now under test.

References

[1] A. Ast, R.W. Hartenstein, H. Reinig, K. Schmidt, M. Weber: A General Purpose Xputer Architecture Derived from DSP and Image Processing; in M.A. Bayoumi (ed.): VLSI Design Methodologies for Digital Signal Processing Architectures; Kluwer Academic Publishers, pp. 365-394, 1994.

[2] D.I. Moldovan: Parallel Processing - From Applications to Systems; Morgan Kaufmann Publishers, 1993.

[3] J.R. Allen, K. Kennedy: Automatic Translation of FORTRAN Programs to Vector Form; ACM Transactions on Programming Languages and Systems, Vol. 9, No. 4, pp. 491-542, October 1987.

[4] U. Banerjee: Dependence Analysis for Super-computing; Kluwer, 1988.

[5] Reiner W. Hartenstein, Rainer Kress, Helmut Reinig: A Reconfigurable Accelerator for 32-Bit Arithmetic; International Parallel Processing Symposium, Santa Barbara, USA, April 1995

[6] R.W. Hartenstein, R. Kress, H. Reinig: A Reconfigurable Data-Driven ALU for Xputers; IEEE Workshop on FPGAs for Custom Computing Machines, FCCM´94, Napa, CA., April 1994.

[7] K. Hwang: Advanced Computer Architecture: Parallelism, Scalability, Programmability; McGraw-Hill, 1993.

[8] K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. Thesis, University of Kaiserslautern, 1994

[9] Z. Shen, Z. Li, P.-C. Yew: An Empirical Study of Fortran Programs for Parallelizing Compiler; IEEE Transactions on Parallel and Distributed Systems, Vol.1, No.3, pp. 356-364, July 1990.

[10] H.P. Zima, B. Chapman: Supercompilers for Parallel and Vector Computers; ACM Press Frontier Series, Addison-Wesley, 1990.