MICROPROCESSOR ARCHITECTURE AND DESIGN FOR GaAs TECHNOLOGY
uvodni referat na MIEL*88 v Zagrebu

Veljko Milutinović

ABSTRACT

GaAs technology has reached the VLSI level of integration. At the same power consumption, it is up to about half order of magnitude faster than Silicon technology, and up to several orders of magnitude more radiation hard. However, it imposes radical changes in the area of computer architecture and computer design. This paper explains several processor design strategies for GaAs technology, and emphasizes the RISC strategy. It discusses the impacts of GaAs technology on the design of CPU resources (adder, register file, etc.), system resources (cache, coprocessing, etc.), and system software resources (code optimization, hardware-to-software migration, etc.). It summarizes the essence of one 32-bit GaAs microprocessor design, and reviews the lessons learned. Finally, it concentrates on the synergism methodology for GaAs micorprocessor design; actually, on its most promising aspect: the catalytic migration.

1. INTRODUCTION

Possible approaches to microprocessor design for GaAs technology include: bit-slice, functional-division, and RISC [1,2,4]. For a number of reasons, the RISC approach seems to be the most promising; however, the design has to be done with a maximal care, and a maximal awareness of the requirements of the GaAs technology (5,6,7). Issues that present special problems include, but are not lomited to: high ratio of off-chip to on-chip delays and storage recess times (high „off-on"), small on-chip transistor count, plus a relatively high sensitivity of gate delays on fan-in and fan-out.

The RCA's design of a 32-bit GaAs RISC microprocessor represents one of the first three efforts in its domain. General architecture and the pipeline structure of this machine are given in the enclosed figure. Further details can be found in (13). Experiments that have preceeded the design are explained in [12,16]. Discussions of other relevant issues can be found in [3,8,9,10,11,14,15].

2. SUMMARY OF THE BOTTLENECKS

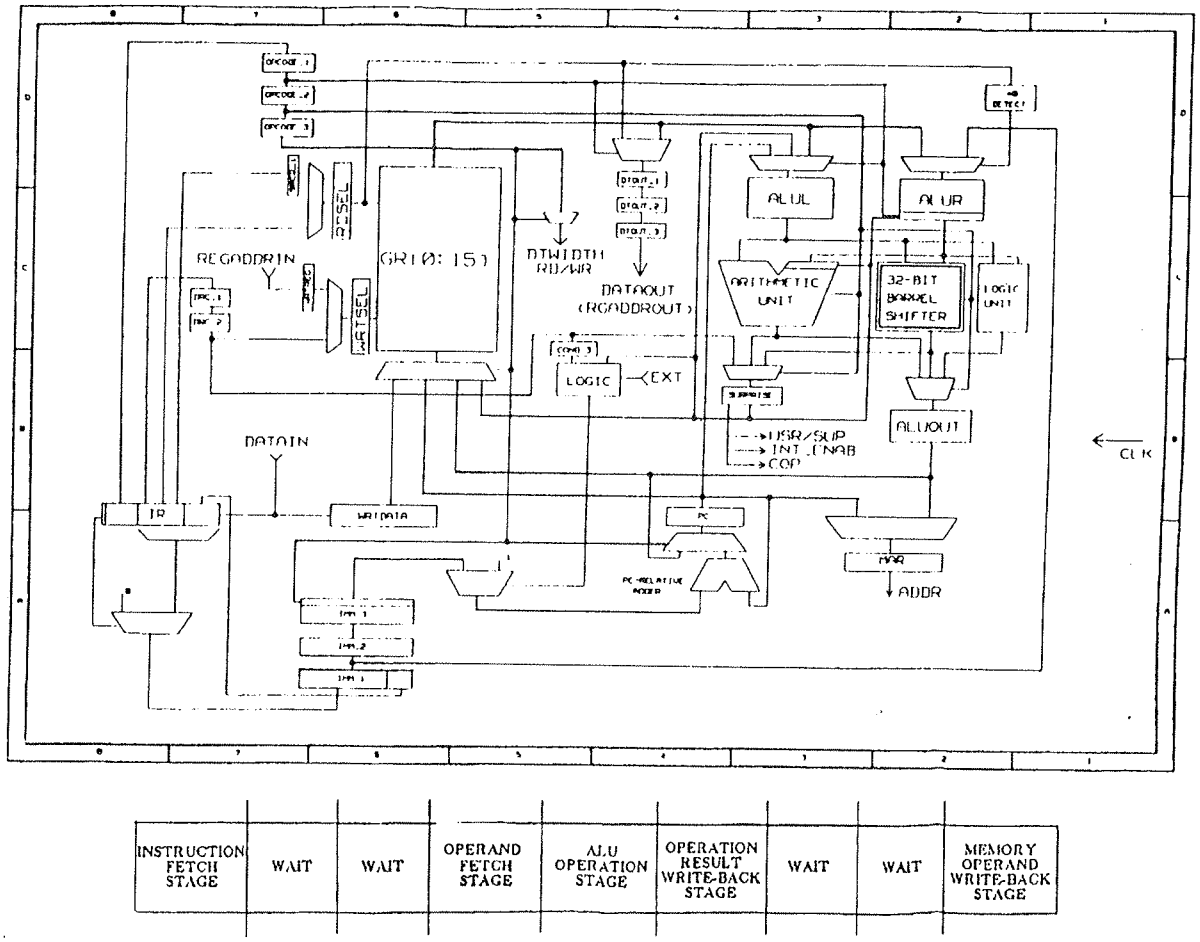
Today it is not a problem to design and implement a 32-bit GaAs microprocessor. The problem is how to design it so that it is close to N times faster (for compiled HLL code) than its Silicon counterpart, where N is the ratio of GaAs/Silicon speeds on the gate level. Benchmarking on the final design has shown that the actual speed (for compiled HLL code) is much below the peak speed of 200 MIPS (for various benchmarks, from about 80 MIPS to about 120 MIPS).

The major bottlenecks are: (a) GaAs technology itself (practical speed and radiation hardness are below the theoretical expectations), (b) Packaging and interconnection technology (this bottleneck seems to be the most difficult to improve), (c) Architecture (architectural constructs are needed that would better address the requirements of the GaAs design environments, and (d) Compiler technology (deep pipelines impose special restrictions not found in Silicon designs).

The purpose of this paper is to concentrate on some new elements of the architecture/compiler synergism methodology that are believed to be able to solve some of the major problems that limit the N-times speed-up.

3. CATALYTIC MIGRATION

Common trend in VLSI processor/computer design for technologies with a relatively large on-chip transistor count, and a relatively low ratio of off-chip to on-chip delays (e.g., CMOS Silicon), is to migrate some of the system software (typically, operating system) into hardware (typically, firmware). The research at Purdue University suggests that the solution for technologies with a relatively small on-chip transistor count, and a relatively large ratio of off-chip to on-chip delays (the case not only with GaAs) is in an opposite type of migration, from hardware (whatever that typically means) to system software (typically, optimizing compiler). The variety of hardware resources that could bi migrated into the optimizing compiler is surprisingly large. We are just not used to thinking in that direction, sometimes we don't dare.



Slika 1

The rationales behind the usefulness of this migration are as follows. If the ratio of off-chip to on-chip delays is fairly high, then the penalty (in terms of the number of clock cycles) for going off the processor chip is fairly high, too. The solution is to „invest" as much processor chip area into the resources that have been proven useful in reducing the frequency of going off the processor chip, or „frequency reductors" (e.g., larger register file, or a larger on-chip cache memory*). However, if technology is characterized with a small on-chip transistor count, then the on-chip implementation of „frequency reductors" may not be feasible, since these resources are typically useful only if large enough. One way to make their implementation feasible is to migrate some of the traditional hardware resources into the optimizing compiler, and make extra room for „frequency reductors" to be properly implemented.

The earliest and the most popular examples of the hardware-to- compiler migration are found in the RISC research of late 70's and early 80's [18,19,20]. These examples include delayed branching, software implementation of the pipeline interlock, and instruction scheduling that eliminates the need for internal forwarding. One possible classification and generalization of the problem is given in [7] where the special attention was dedicated to the synergistic effects that the hardware-to-compiler migration can produce. An important inpuls to this research comes from [17]. It is the first to recognize the fact that, in some cases, the synergistic effects can be more easily achieved, or performance gains further improved, if the migration is accompanied by an addition of some simple hardware to assist in the utilization of static compile-time decisions. Dietz and Chi [17] refer to this type of migration as integration (of compile-time information and run-time hardware).

*After a certain point, these don't serve as „frequency reductors" any more.

Paper [17] discusses several migration examples (e.g., migration of complex arithmetic func tions by implementing them out of add/subtract, shift/rotate, and increment/decrement, migration of complex addressing modes by implementing them out of the simplest ones, and migration of deskewing algrorithm, in skewed memory systems). It also discusses several Integration examples (e.g., migration of cache pollution control, squashing branches, and CRegs).

The main purpose of this paper is to introduce and analyze further examples of migration and especially integration. Here, term migration will be substituted by the term direct migration. Term integration will bi substituted by the term catalytic migration. The later is to underline the fact that the newly added hardware should be much less in compexity than the migrated hardware, and should predominantly serve as a catalyst which enables an effective synergetic process to take place.

This paper also discusses the issues of importance for the implementation and evaluation of different migration candidates.

4. RELEVANT ISSUES

Further classification of direct and catalytic migrations is straightforward. Basic elements of a processor/computer system are control, arithmetic, storage, and I/O. Therefore, it is natural that examples of both direct and catalytic migration are classified in the same way.

Any migration is not necessarily useful, for given criteria. Assume that the criterion is the speed of compiled HLL code, and that the VLSI chip area is fixed. After a function is migrated, it may become slower (typical case). The slowdown is highly dependent on the implementation of the migration, i.e. on the utilized compile-time algorithm; the two main components of each migration (run-time procedure and compile-time procedure) should be treated jointly. Evaluation of the slowdown is an important problem. On the other hand, the released VLSI area is „invested" into resources which speed-up the compiled HLL code. The speedup is dependent both on the type of new resource and on the capability of the optimizing compiler to utilize that resource efficiently. Again, the two issues should be treated jointly. Evaluation of the speed-up is another important problem.

Since it is assumed here that the overall VLSI area did not change, the VLSI area for added resources should be equal to the area under the removed resource(s), minus the area under the catalytic resource(s), if any.

One way to look into the migration issue is as follows. The major problems to solve are:

- 1. Invention of a new migration (direct or catalytic).
- 2. Specification of the run-time architecture (and possible modifications of the machine-level instruction set).
- 3. Determination of the saved VLSI area (which is implementation tools dependent).
- 4. Specification of the compile-time procedures (for new migrations), plus improvements for old migrations (not subject of this research).
- 5. Determination of the slowdown due to migration (in general, could be done empirically or analytically).

Due to a variety of problems involved, the best results are accomplished through a joint effort of researchers with the backgrounds in computer design, computer architecture, VLSI design, compiler design, and performance evaluation. The synergism methodology gives the best results through synergistic effects of joint research of an interdisciplinary team.

5. EXAMPLES

This section briefly introduces a number of recently recognized catalytic migrations, and underlines the migration issues of relevance for computer design and computer architecture. Issues of relevance for VLSI design (i.e., area estimation), compiler design (i.e., compile-time algorithms), and performance evaluation (i.e., speed-up estimation) are the subject of the follow-up work. Here is the list of candidates for the catalytic migration:

- 1. Catalytic running of two different programs in the branch delay slot of each other, with a catalyst which controls the expansion of the code „fanout" tree.
- 2. Catalytic migration of a 2-read-port register file into a 1- read-port register file, with a catalyst which faciliates the „bang-bang" operation.
- 3. Catalytic migration of post-branch NOOPs though the use of IGNORE-like instructions.
- 4. Catalytic migration of the PC-stack, with a catalyst which enables the usage of instructions rather than their addresses.
- 5. Catalytic migration of static RAM into dynamic RAM for on-chip cache memories.

* 6. Catalytic migration of the destination control for loading from multidistance memories.
* 7. Catalytic migration of the windowed register file.
* 8. Catalytic migration of the bus sizing.
* 9. Catalytic migration of the remote-PC.
* 10. Catalytic migration of some elements of instruction/data buffers.

More details on each of the above could be found in [21]. The common thread in all above examples is the existence of a catalyst which is in some cases not absolutely necessary, but helps to increase the efficiency. In many cases a new instruction type serves as a catalyst.

6. CONCLUSION

We strongly believe that the future of GaAs microprocessor is in a further exploitation of the catalytic migration design methodology. In our opinion, future success of GaAs microprocessors depends largely on our ability to come up with new and efficient forms of catalytic migration. Of course, efficient implementation (in the architecture and the compiler) is an important prerequisite of the final success.

7. LIST OF RELATED REFERENCES

The text to follow includes references to various sources that have been quoted in this paper, directly or indirectly. These references include both, surveys of general R&D activities in the field, as well as the research results from Purdue University (only those in the area of microprocessor architecture and design for GaAs technology).

Edited Original Book:

(1) V. Milutinović (editor), Microprocessor Design for GaAs Technology, Prentice Hall, 1988. Contributors to this book include, but are not limited to: Naused (Mayo Foundation), Larson (Hughes), Fura (Boeing), Vlahos (TRW), Heemeyer (CDC), Geideman (McDonnell Douglas), Helbig (RCA), and Milutinović (Purdue).

Edited Repring Selection:

(2) V. Milutinović, D.Fura (editors), Tutorial on GaAs Computer Design, IEEE Computer Society Press, 1988.

State-of-the-Art Survey Paper:

(3) V.Vlahos, V. Milutinović, „GaAs Microprocessors and Digital Systems: A Survey of R&D Efforts," IEEE Micro, February 1988.

Journal Papers:

(4) V. Milutinović, D.Fura, W.Helbig, „An Introduction in the GaAs Computer Architecture for VLSI," IEEE Computer, March 1986, pp. 30-42.

Translated into Japanese and republished by NIKKEI ELECTRONICS, Tokyo, Japan, October 1986.

(5) V. Milutinović, „State-of-the-Art Computer Design for GaAs Technology," IEEE Computer, October 1986 (Guest Editor's Introduction), pp.10-15.

(6) V. Milutinović, A.Silbey, K.Keirn, M.Bettinger, D.Fura, W.Helbig, W.Heagerty, R.Ziegert, R.Schellack, W.Curtice, „System Issues in VLSI Computer Architectures for GaAs," IEEE Computer, October 1986, pp. 45-57.

(7) V. Milutinović, D.Fura, W.Helbig, J.Linn, „Architecture/Compiler Synergism in VLSI Computer Systems for GaAs," IEEE Computer, May 1987, pp. 72-93-

(8) K.McNeley, V.Milutinović „Emulation of a CISC with a RISC," IEEE Micro, February 1987, pp. 60-72.

(9) V. Milutinović, N.Lopez-Benitez, K.Hwang, „A Vertical Migration Microprocessor Architecture for GaAs Implementation and Real-Time Applications," IEEE Transactions on Computers, June 1987, pp. 714-727.

(10) V. Milutinović, „Simulation Study of Vertical-Migration Microprocessor Architecture," IEEE Transactions on Software Engineering, December 1987, pp. 1265-1277.

(11) V. Milutinović, „A Simulation Study of GaAs-Oriented Suboptimal Detection Procedures," IEEE Transactions on Communications, May 1988.

(12) V. Milutinović, M.Bettinger, W.Helbig, „Multiplier/Shifter Design Trade-offs in a 32-bit Microprocessor," IEEE Transactions on Computers, October 1988.

(13) W.Helbig, V. Milutinović, „The RCA's DCFL E/D-MESFET GaAs 32-bit Experimental RISC Machine," IEEE Transactions on Computers, December 1988.

Conference Papers:

(14) J. Fortes, V.Milutinović, R.Dick, W.Helbig, W.Moyers, „A High-Level GaAs Systolic Array," Proceedings of the ACM/IEEE International Workshop on High-Level Computer Architecture, Honolulu, HI, January 1986.

(15) B. Preuničić, S.Lakhani, V.Milutinović, „Modelling and Analysis of Stochastic Propagation Delays in GaAs Adders," Proceedings of the ACM/IEEE Hawaii International Conference on System Sciences, Kona, HI, January 1988.

(16) V.Milutinović, M.Bettinger, W.Helbig, „Adder Design Analysis for GaAs Technology," IEEE Tutorial on GaAs Computer Design, Washington, D.C., January 1988.

Other References:

(17) Dietz, H., Chi, C.-H., „A Compiler-Writer's View of GaAs Computer System Design," Proceedings of the HICSS-21, Kona, Hawaii, January 1988, pp. 256-265.

(18) Radin, G., „The 801 Minicomputer," IBM Journal of Research and Development, Vol. 27, No. 3, May 1983, pp. 237-246.

(19) Paterson, D.A., „Reduced Instruction Set Computers, Communications of the ACM, Vol 28, No. 1, January 1985, pp. 8-21.

(20) Hennessy, J., VLSI Processor Architecture, IEEE Transactions on Computers, Vol. 34, No. 12, December 1985, pp. 66-77-

Internal Report:

(21) Milutinović, V., „The 1988 Sponsored Research Progress Report," Purdue University Internal Report, 1988.

Veljko Milutinović
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907
U.S.A.