computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

THE EFFECT OF DATA-REUSE TRANSFORMATIONS ON MULTIMEDIA APPLICATIONS FOR APPLICATION SPECIFIC PROCESSORS

N. Vassiliadis, A. Chormoviti, N. Kavvadias, S. Nikolaidis

Section of Electronics and Computers, Department of Physics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece, E-mail: nivas@skiathos.physics.auth.gr

Abstract: *Multimedia applications are characterized by a high number of data transfers and storage operations. Appropriate transformations can be applied at the algorithmic level to improve crucial implementation characteristics. In this paper, the effect of data-reuse transformations on power consumption and performance of multimedia applications, realized on an Application Specific Instruction set Processor (ASIP), is examined. An ASIP for multimedia applications designed based on a complete methodology is used to evaluate this effect. Results prove the efficiency of the ASIP solution and indicate benefits from the use of the data-reuse transformations in terms of energy consumption and performance. Also, preliminary results from the exploitation of instruction buffering technique to reduce the energy consumption of the ASIP are presented.*

Keywords: *ASIP, Data-Reuse Transformations, Multimedia Applications.*

1. INTRODUCTION

The popularity of multimedia systems used for computing and exchanging information is rapidly increasing. With the emergence of portable multimedia applications (mobile phones, laptop computers, video cameras, etc) the power consumption has been promoted to a major design consideration due to the requirements for long battery life, large integration scale and the related cooling and reliability issues [1], [2]. Consequently, there is great need for power optimization strategies, especially in higher design levels, where the most significant savings are achieved.

A number of code transformations can be applied to any algorithm aiming at a memory hierarchy where copies of data from larger memories that exhibit high data-reuse are stored to additional layers of smaller memories. In this way, exploiting the temporal locality of data memory references [1], the greater part of the accesses is performed on smaller memories. Since accesses to smaller levels of the memory hierarchy are less power costly, significant power savings can be obtained [3], [4].

Different hardware architectures can be used for the implementation of an application. The use of application specific integrated circuits (ASICs) leads to high performance, small area and power consumption. However they completely lack flexibility since only a specific algorithm can be implemented on the system. A flexible solution, with a negligible Time-to-Market (TTM), is the use of an existing General Purpose Processor (GPP). Such a solution is rather unlikely to be viable, due to the fact that conventional RISC/DSP approaches pose limitations in tuning the architecture towards narrow application domains or they may be prohibitively expensive in respect to energy consumption [5]. Thus, the embedded systems industry shows an increasing interest in ASIPs. ASIPs are processors tailored to the needs of the target application [6], providing the right balance between flexibility, performance, and power consumption.

In this paper a methodology for the implementation of an ASIP from a hardware-software perspective is followed. Based on this methodology an ASIP for multimedia applications is designed. The effect of data-reuse transformations on an application, which is executed on this ASIP, is examined. The Two Dimensional Three-Step Motion Estimation video coding algorithm is used as benchmark. A comparative study indicates that known benefits from data-reuse transformations in power and performance of GPPs are valid for an ASIP approach.

In addition, preliminary results from the use of the instruction buffering technique [14] to decrease energy consumption are presented. Typically applications executed in embedded processors consist of small fragments of code that are heavily executed. Such, fragments can be stored and accessed from a small local storage structure rather than from the Instruction Memory. Since, accesses to the Instruction Memory of an embedded processor are a major source of energy consumption [15] significant energy savings can be obtained.

In section 2, a brief description of the data-reuse methodology and the used benchmark is given. The followed ASIP design flow is presented in section 3. Results are discussed and analyzed in section 4 while we conclude in section 5.

2. DATA-REUSE TRANSFORMATIONS

In data-dominated applications such as multimedia algorithms, significant power savings can be achieved by developing a custom memory organization that exploits the temporal locality in memory accesses [1]. According to the proposed methodology, data sets that are often being accessed in a short period of time are identified and placed into smaller memories leading to a new memory hierarchy. Hence, power savings can be obtained by accessing heavily used data from smaller foreground memories instead of large background memories. Such an optimization requires architectural modifications that consist of adding layers of smaller memories to which frequently used data can be copied. Consequently, there is a trade off here; on the one hand, power consumption is decreased because data is now read mostly from smaller memories, while on the other hand, power consumption is increased because extra memory transfers are introduced.

An exploration of all architectural alternatives is required for finding the optimum solution. This data-reuse exploration is performed by applying a number of code transformations to the original code, which are determined by the group of data sets that are being used in the algorithm. These transformations are extracted according to the methodology described in [3], [4].

As a benchmark, for the application of data-reuse transformations, the popular motion estimation, two dimensional Three-Step Search (TSS) algorithm is used. Motion estimation algorithms are used in MPEG video compression systems [7], [8] to remove the temporal redundancy in video sequences which is determined by the similarities amongst consecutive pictures. Instead of transmitting the whole picture, only the displacements of pixel blocks (motion vectors) between neighboring pictures (frames) and the difference values for these blocks have to be encoded. The calculation of the motion vector is performed by means of the matching criterion, a cost function to be minimized [8].

TSS consists basically of four nested loops, one for each block in the frame, one for each step, one for each candidate block and one for every pixel in the block. The number of the corresponding transformation, produced by the application of the data reuse transformations on the TSS, and the size of the introduced memory, given parametrically, annotate each rectangle in Fig.1.

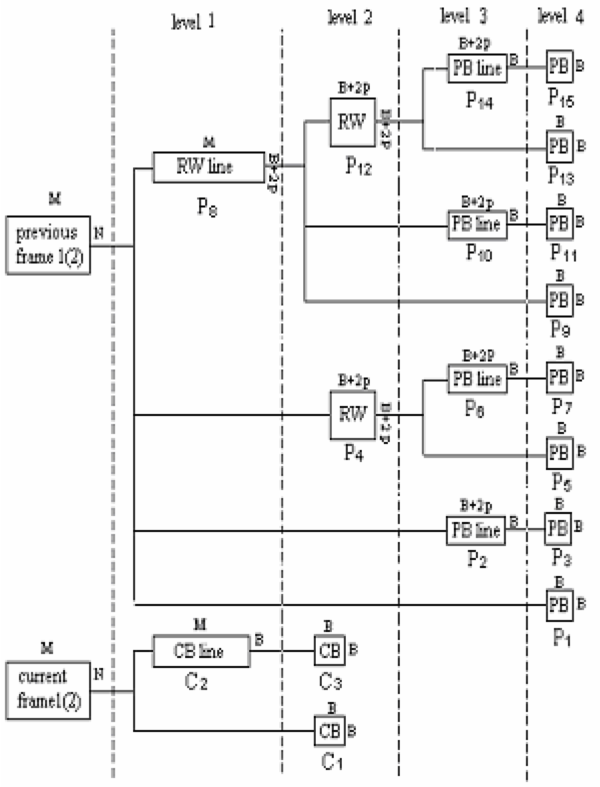


Fig.1 – Custom Memory Hierarchy

3. ASIP DESIGN FLOW

An important issue in ASIP design is the identification of the tradeoffs involved in instruction set and micro-architecture design, which requires efficient architecture design space exploration. In this paper an existing processor architecture is used to initialize the design flow. The goal of the flow is to extend the existing instruction set of the selected processor by identifying and incorporate new instructions from which the target application can benefit in terms of performance and power consumption. At the same time these newly defined instruction should not introduce significant hardware overhead and not increase the critical path of the processor.

The different steps of the followed design flow are presented below. The TSS algorithm and its data reuse transformations are used as representative benchmark. In addition, since the main objective of this work is to study the effect of data reuse transformations on ASIPs, the selection of such an algorithm, suitable for the applications of such transformations, is straightforward.

3.1 ESTABLISHMENT OF AN ARCHITECTURE TEMPLATE

In our work, the architecture model assumed is a 32-bit single-issue machine with a five-stage pipeline, with separate instruction and data buses (Harvard architecture), consistent to the RISC (Reduced Instruction Set Computer) paradigm. The control model is data stationary and execution is performed in-order. Thus, a MIPS-like processor architecture [9] was selected to initialize the design flow.

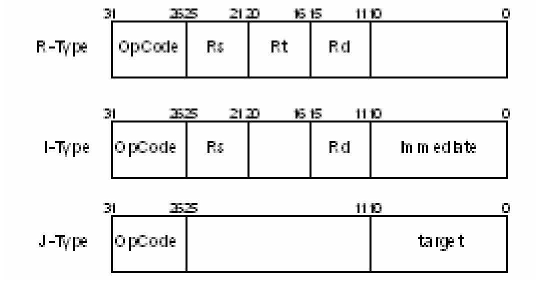


Fig. 2 – Basic Instruction Formats

The initial 32-bit Instructions of the processor can be grouped in three basic formats, illustrated in Fig.2. R-type instructions use two source registers and one destination register. I-type instructions feature one source register, one destination register and an immediate field containing a constant value. Load and Store instructions and conditional branches, for example, belong to this category. J-type format is typically used by jump instructions and contains only a target field.

3.2 FRONT-END COMPILATION

At first, the application described in a high level language, in particular ANSI C, must be compiled for the target architecture. The GNU-GCC [10] for embedded architectures, configured as a cross-compiler for the MIPS architecture, is used for this reason. The TSS algorithms with the different data reuse transformations are compiled and machine code for each transformation is generated in this stage.

3.3 DYNAMIC PROFILING

The produced machine code is dynamically profiled with the GNU [10] tools (gcc, binutils, gdb) configured for the MIPS processor. Profiling information at the levels of C and assembly code, generated by the execution of the application on the target machine is collected. In this way heavily executed portions of the code can be identified. New candidate instructions from which the application can benefit in terms of performance (at first) can be revealed. Dynamic profiling was performed on all, modified by the data reuse transformations, versions of the application code.

Average profiling values from all the different transformations, at the level of the C code, indicate that the control flow of the application, namely the loop and case statements ("for", "while", "loop", "if") in the C code consists 24% of the total execution cycles. The bigger portion of the execution time is consumed on the addressing equations and on access to the different memory layers. That is the 62% of the total execution cycles. Only 14% of the execution time is consumed on pure computational micro-operations.

In addition profiling information on the assembly code indicates that there are patterns of instructions that appear with high frequency on the above identified heavily executed portions of the code. In particular the loop statements are executed with two instructions overhead, one instruction for the iteration of the loop and one for the conditional branch of the program. Clearly a newly defined complex instruction that can control the flow of the program with one cycle overhead can boost the performance of the processor. Also various implementations of the branch instruction could have the same results.

As it was observed, a large portion of the execution time is consumed in the addressing of and access on the different memory layers. Load and Store instructions with opcodes refereeing directly to a specific memory layer should be added. Also an arithmetic operation, in particular an addition, always exists before a memory access due to the addressing calculation. Combining the addition with Load Store instructions, resulting in the support of the appropriate addressing modes, significant speed up of the processor's performance can be achieved. It must be pointed out that the requirement for such addressing modes imposes the use of different pipeline stages for the execution and memory access stages.

Furthermore, profiling information can be used to deploy instruction buffering technique. Identified heavily executed portions of the code are candidates for storage in a local storage structure rather than the main Instruction Memory. The trade-off between the storage size and energy savings will determine the instructions that will finally be included in this local memory.

3.4 INSTRUCTION SET EXTENSIONS

The previously identified instructions, from which the processor can benefit, must be incorporated in the existing instruction set.

Additional hardware resources and changes on the processor architecture necessary for the execution of the new instructions must be taken into account.

A new type of instruction is added to support an alternative implementation of the conditional branch instruction. Fig.3, illustrates the new B-Type format of those instructions. The difference is that the I-Type instruction format presented in Fig.1 can support a conditional branch in which registers, Rs and Rd, are compared and if the branch is taken the program branches to the address indicated by the immediate value. The B-Type format can support comparison between an immediate constant value and the Rs register and branch to the address indicated by the target field, in the case that the branch must be taken. The B-Type format does not require any additional computational hardware components. Slight changes on the Instruction Decoding Unit, the Hazard detection Unit and the overall control logic must be made, to support the new format. These changes slightly increase the area requirements with no penalty to the critical path (performance).

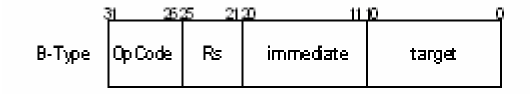


Fig. 3 – The B-Type Instruction Format

A second extension to the instruction set aims to the reduction of the loop overhead from two instructions to one. For this reason the branch instructions of the I-Type and B-Type are combined with an increment operation. The Rs register is incremented, the result is compared with the Rd register, for the I-Type format, or the immediate value for the B-Type format, and based on the result a branch is performed or not. The incremented value of the Rs register is written back in the same register. An increment unit must be included in the Arithmetic/Logical Unit to support the increment/branch MOP. The delay of the new instruction is not in the critical path length therefore no reduction on the performance is occurred. Also minor changes in the control logic of the processor must be performed.

As have been observed the largest portion of the execution cycles is consumed on the addressing equations and memory access. Clearly new defined addressing modes must be incorporated. Firstly separate Load and Store operations for each memory layer are included in the instruction set. Therefore, overhead due to the partition of the memory addresses is removed. Secondly the new Load/Store operations are combined with an addition operation, to produce a new addressing mode, since such MOPs are always executed sequentially. The new complex instruction has a format identical with the R-Type. The Rs and Rt are added, result is used as a memory address in which the content of the Rd register will be stored, or a memory address, the contents of which are going to be loaded in the Rd register. In addition a complex add+Store instruction with immediate value of the data to be stored is included in the instruction set. The SI-Type instruction format is illustrated in Fig.4. The Rs register is added to the Rt register. The produced result is used as the address of the memory layer, provided by the opcode, in which the immediate value is going to be stored. In order for the new instructions to be supported by the architecture, appropriate decoding of the opcodes must be added to give direct access to the desired memory layer. Then, the pipeline stages of the Execution and Memory must be controlled for the new addressing mode. In order for the new defined instructions to be supported by the processor architecture, additional decode logic and control signals, that add a slight hardware overhead, must be included. Also, no increase on the critical path has occurred due to the incorporation of these instructions, resulting in no degradation on performance. The instruction set extensions are summarized in Table 1.

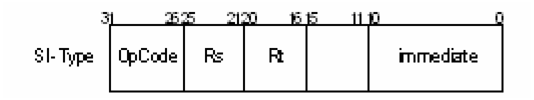


Fig. 4 – The SI-Type Instruction Format

Table 1. Instruction Set Extensions

Description/Format	Instruction Format Type	Additional Hardware Requirements	Penalty
Branch Rs, Immed, Target	B-Type	Control Logic	Area
Inc+Branch Rs, Immed, Target	B-Type	Control Logic + Incrementer Unit	Area+Delay
Inc+Branch Rs, Rd, Target	I-Type	Control Logic + Incrementer Unit	Area+Delay
Add+SW_L# Rs, Rt, Rd	R-Type	Control Logic	Area
Add+LW_L# Rs, Rt, Rd	R-Type	Control Logic	Area
Add+SW_L# Rs, Rt, Immed	SI-Type	Control Logic	Area
L# is the desired level of the custom memory hierarchy			

3.5 CODE RE-GENERATION

Code Re-Generation is performed by taking into account the new defined instructions. First the original code is parsed and the MOPs are reordered in order to construct the previously identified instruction patterns. The patterns are then substituted by the new defined instructions. Finally, due to the fact that there is no flush unit in the processor architecture, MOPs are reordered to keep the pipeline as full as possibly [11]. Fig.5 illustrates an example.

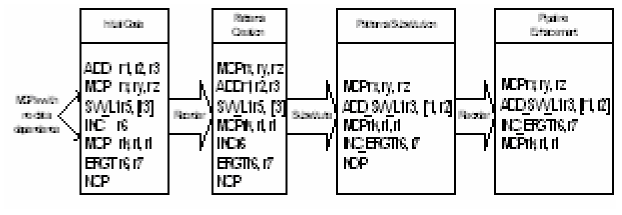


Fig. 5 – Code Re-Generation

3.6 CYCLE ACCURATE SIMULATION

For the evaluation of the candidate new defined instructions a cycle accurate simulation model, using SystemC, is constructed. The application code with the instruction set extensions is executed on the simulator and execution cycles for each transformation are derived. In addition information about the execution of particular instructions and access to crucial hardware components like memories, are collected.

3.7 HARDWARE MODEL

A model in a hardware description language (VHDL) is designed. All the necessary micro-architectural and hardware modifications in order for the new defined instructions to be supported by the designed processor are incorporated. The design is synthesized on a popular standard cell technology and information for the performance (critical path), power consumptions and the area requirement of the processor are collected. Information from the previous executed steps is collected. Results are evaluated and they are presented in the next section.

4. EXPERIMENTAL RESULTS

The different versions of the TSS code, resulting by the application of the data-reuse transformations were compiled for the ASIP core. Cycle accurate simulations were performed using the SystemC simulator designed for this reason. Execution cycles and accesses to different memory layers were collected from the simulations.

The TSS was executed for a picture of MxN=144x176 pixels. The block size B was set to 16 while the search window size [-p,p] was set to [-7,7].

4.1 PERFORMANCE RESULTS

Total execution cycles for each transformation are presented in Fig.6. Fig.6 actually indicates that since data-reuse transformations simplify the addressing calculation, it can be used not only for energy savings but also to boost performance. Specifically, the P4 transformation, which provides the highest performance, achieves performance gain of 55% compared to the original TSS.

In order to evaluate the designed ASIP efficiency towards a GPP with architectural similarities, the application codes were executed on an ARM9TDMI core [12]. For the most efficient P4 transformation the ASIP is capable to deliver 54% performance gain compared with the ARM9TDMI core.

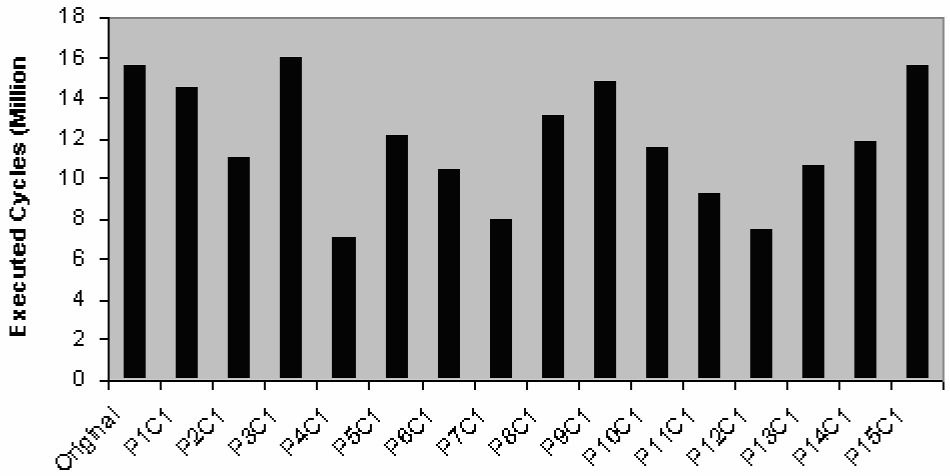


Fig. 6 – Executed Cycles for ASIP

4.2 ENERGY RESULTS

Data-intensive applications are dominated by power consumption due to data and instruction memories accesses [2]. Based on this assumption, energy consumption estimations for the different transformations on the ASIP are presented in Fig.7.

For the calculation of the energy consumptions, accesses to the instruction memory and the used data memory layers for each transformation were obtained from the cycle accurate simulations. SRAM memories with appropriate size were used for each layer of the data memory. For the Instruction memory a 1KByte SRAM was used. Power consumption for each memory was obtained from the Embedded Memory Generator of [13].

Results indicate that energy savings can be obtained through data-reuse transformations based on the reduced number of accesses on the instruction memory. P4 is also the best transformation, in terms of energy savings achieving 61% energy savings compared to the original TSS.

4.3 ENERGY RESULTS USING INSTRUCTION BUFFERING

Fig.7 indicates that energy consumption is dominated by the energy consumption due to accesses on the instruction memory. Therefore, Instruction Buffering can be used to dramatically reduce the Instruction Memory energy consumption and consequently the overall energy consumption.

As already mentioned, TSS code consists primary of nested loops where the inner loop is the most heavily executed one. The code of the inner loop can be moved from the instruction memory to a local storage structure. This structure was modelled as a register file with negligible energy consumption. For all transformations the size of the instruction register file was smaller than the 25% of the operand register file of the ASIP core. Based on these assumptions, energy consumption estimations for the different transformations on the ASIP with the instruction buffering are presented in Fig.8. Results indicate that an average reduction of 33% in energy consumption can be obtained with instruction buffering. Moreover, for the P4 transformation a dramatically 68% reduction, compared to the case with no instruction buffering, is feasible.

5. CONCLUSIONS

In this paper, the effect of data-reuse transformations on multimedia applications implemented on an ASIP platform has been presented. An ASIP for multimedia applications has been designed for this reason. The popular, on video compression systems, TSS algorithm and its modifications imposed by data-reuse transformations were used as benchmarks. Performance and energy consumption of these benchmarks were estimated, on the ASIP.

Results indicate that the ASIP can benefit in terms of performance and energy consumption by selecting the appropriate custom data memory hierarchy. In addition preliminary results on instruction buffering indicate that significant energy reduction is feasible.

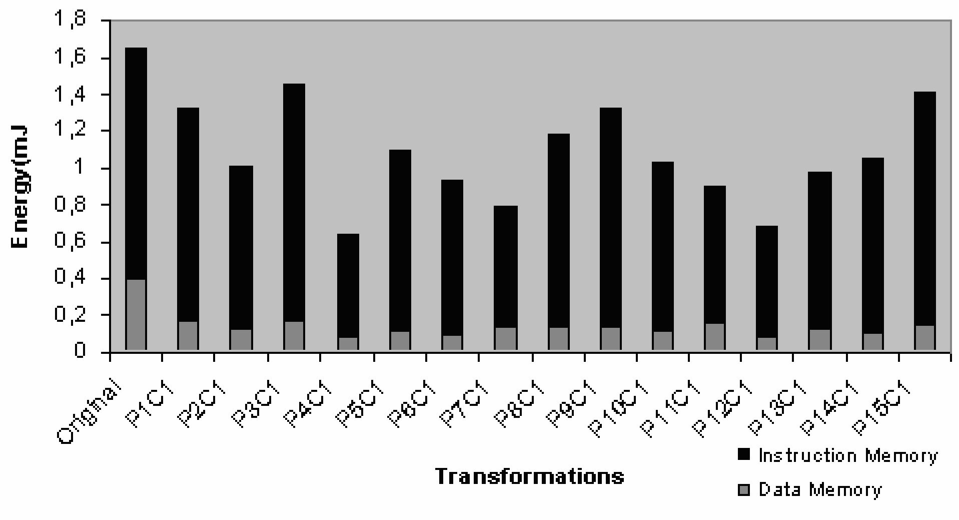


Fig. 7 – ASIP Energy Consumption

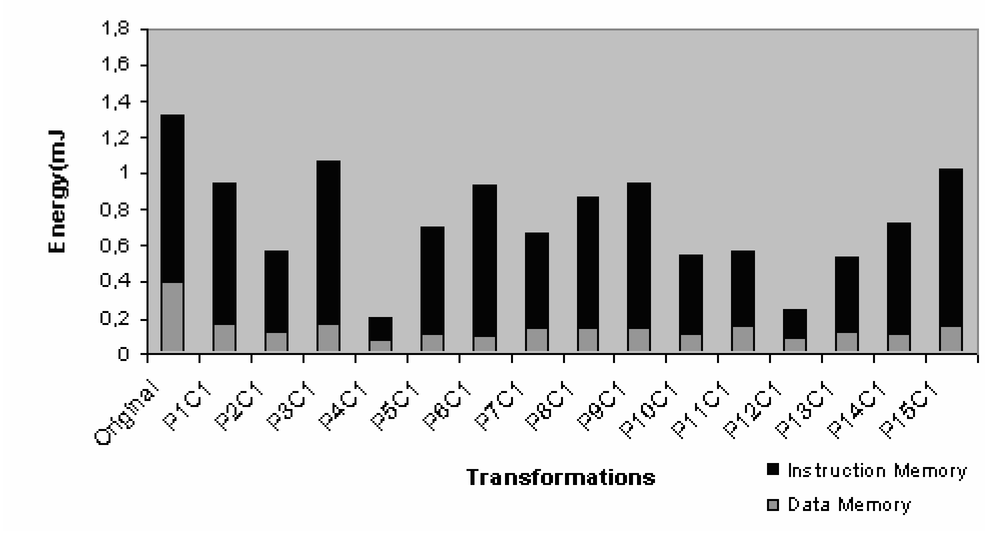


Fig. 8 – ASIP Energy Consumption with Instruction Buffering

6. REFERENCES

[1] F. Catthoor, S. Wuytack et al: *Custom Memory Management Methodology*, Kluwer Academic Publishers, Boston, 1998.

[2] A. Chandrakasan and R. Brodersen: *Low Power Digital CMOS Design*, Kluwer Academic Publishers, Boston, 1995.

[3] S. Wuytack, J-P. Diguet, F. Catthoor, H. De Man : Formalized Methodology for Data Reuse Exploration for Low-Power Hierarchical Mappings, *special issue of IEEE Transactions on VLSI Systems on low power electronics and Design*, Vol. 6, No. 4, pp. 529-537, December 1998.

[4] S. Kougia, A. Chatzigeorgiou, N. Zervas, S. Nikolaidis : Analytical Exploration of Power Efficient Data-reuse Transformations on Multimedia Applications. *International Conference on Acoustics, Speech and Signal Processing*, Utah, May 2001

[5] M.F. Jacome and G. de Veciana, : Design Challenges for New Application-Specific Processors, *IEEE Design and Test of Computers*, Vol. 17, No. 2, pp. 40-50, 2000.

[6] MK. Jain, M. Balakrishnan, A. Kumar : ASIP Design Methodologies : Survey and Issues. *Proceedings of 14th International Conference on VLSI Design*, pp. 76-81, January 2001

[7] International Organization of Standardization, Working Group on Coding of Moving Pictures and Audio, MPEG-4 Video Verification Model Version 18.0, Pisa, January 2001.

[8] Vasudev Bhaskaran and Konstantinos Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures, Second Edition*, Kluwer Academic Publishers, Boston, 1999.

[9] J.Hennessy and D.Patterson,: *Computer Architecture: A quantitative approach*, Morgan Kaufmann,1991.

[10] GNU Website : http://www.gnu.org

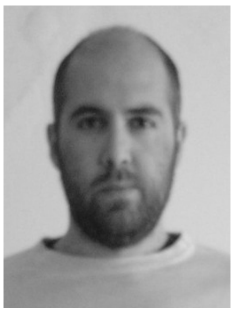
[11] I.-J. Huang and A. Despain, : Synthesis of application specific instruction sets, *IEEE Trans. on CAD*, pp. 663-675, 1995.

[12] ARM Ltd. ARM9 Datasheet, 2003

[13] Dolphin Website: http://www.dolphin.fr/flip/ragtime

[14] Raminder S. Bajwa , Mitsuru Hiraki , Hirotsugu Kojima , Douglas J. Gorny , Kenichi Nitta , Avadhani Shridhar , Koichi Seki , Katsuro Sasaki, Instruction buffering to reduce power in processors for signal processing, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, p.417-424, Dec. 1997

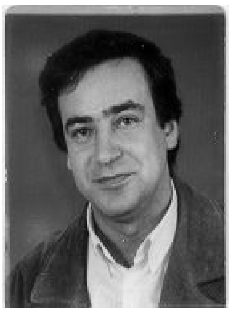
[15] L. Benini et al., A Power Modelling and Estimation Framework for VLIW-based Embedded Systems, *in Int. Workshop on Power and Timing Modelling, Optimization and Simulation, (PATMOS)*, pp. 2.3.1-2.3.10, 2001.



__Nikolaos Vassiliadis__ received the B.Sc. degree in Physics and the M.Sc. in electronics engineering from the Aristotle University of Thessaloniki, Greece in 2001 and 2004, respectively. Currently he is in pursuit of his Ph.D. degree in reconfigurable computing at the same university. His current research interests include reconfigurable computing, computer architecture and hardware/architecture description languages.

Alexandra Chormoviti *received the B.Sc. degree in Physics and the M.Sc. in electronics engineering from the Aristotle University of Thessaloniki, Greece in 2001 and 2004, respectively.*

Nikolaos Kavvadias *received the B.Sc. degree in physics and M.Sc. in electronics engineering from the Aristotle University of Thessaloniki, Greece in 1999 and 2002, respectively. Currently he is in pursuit of his Ph.D. degree in computer engineering at the same university. His current research interests include hardware and architecture description languages, application-specific processor design methodologies, and energy consumption modeling for embedded processors.*



Spiridon Nikolaidis *received the Diploma and PhD degrees in electrical engineering from Patras University, Greece, in 1988 and 1994, respectively. He is now an assistant professor in the Department of Physics of the Aristotle University of Thessaloniki, Greece. His research interests include CMOS gate propagation delay and power consumption modelling, power consumption modelling of processors and design of application specific and reconfigurable processors.*