Journal of VLSI Signal Processing 44, 153–171, 2006

DOI: 10.1007/s11265-006-7513-6

Behavioral-Level Performance and Power Exploration of Data-Intensive Applications Mapped on Programmable Processors*

NIKOLAOS KROUPIS
Dept. of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece

NIKOLAOS ZERVAS
ALMA Technologies, 2 Marathonos Av., Pikermi, Attika, GR 19009, Greece

MINAS DASYGENIS AND KONSTANTINOS TATAS
Dept. of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece

ANTONIOS ARGYRIOU
School of Electrical & Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA

DIMITRIOS SOUDRIS AND ANTONIOS THANAILAKIS
Dept. of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece

Published online: 27 May 2006

Abstract. The continuous increase of the computational power of programmable processors has established them as an attractive design alternative, for implementation of the most computationally intensive applications, like video compression. To enforce this trend, designers implementing applications on programmable platforms have to be provided with reliable and in-depth data and instruction analysis that will allow for the early selection of the most appropriate application for a given set of specifications. To address this need, we introduce a new methodology for early and accurate estimation of the number of instructions required for the execution of an application, together with the number of data memory transfers on a programmable processor. The high-level estimation is achieved by a series of mathematical formulas; these describe not only the arithmetic operations of an application, but also its control and addressing operations, if it is executed on a programmable core. The comparative study, which is done using three popular processors (ARM, MIPS, and Pentium), shows the high efficiency and accuracy of the methodology proposed, in terms of the number of executed (micro-)instructions (i.e. performance) and the number of data memory transfers (i.e. memory power consumption). Using the proposed methodology we estimated an average deviation of 23% in our estimated figures compared with the measurements taken from the real execution on the CPUs.

Keywords: Programmable platforms, estimation methodology, instruction complexity, memory transfers

1. Introduction

Rapid advances in the areas of high-performance programmable and general-purpose processor cores have made them an attractive solution for the realization of data-intensive Digital Signal Processing (DSP)

*This work was supported by the project PENED '99 ED501 funded by GSRT of the Greek Ministry of Development, and the project PRENED '99 KE 874 funded by the Research Committee of the Democritus University of Thrace. This work was partially sponsored by a scholarship from the Public Benefit Foundation of Alexander S. Onassis (Minas Dasygenis).

applications, due to the flexibility they offer, as well as because of the short time-to-market. Even the implementations of complex and computationally-intensive tasks, for instance video compression applications, migrate from custom processors to programmable platforms (e.g. [19] and [24]). To reach a performance- and power-optimized implementation point, the main bottleneck is to manipulate the computational complexity of data-intensive applications.

In data-intensive signal processing applications, for instance multimedia applications, the total memory power consumption is the largest portion of the total energy budget of the whole system [5]. Specifically, power consumption is related to the number of memory transfers, which is the dominant factor of total power cost, and this motivated the researchers to find efficient techniques to reduce it. For that purpose, a data memory hierarchy that exploits the temporal locality of the data, by reusing them, was suggested [3, 5, 16, 17]. If there is a sufficient reuse of the data elements, it can be advantageous to copy the frequently-used data to a smaller memory, so that successive requests of this data element can be read from a smaller memory instead of a larger one. The smaller memories require less power per access and, as a result, the total power consumption can be reduced drastically [21]. In case, the target architecture is a programmable processor, the total memory power consumption consists of two components [17]: i) the energy consumption due to data memory transfer, and ii) the energy consumption due to instruction memory transfers. The array signals dominate mainly the data memory transfers and the memory size for storing them. The number of executed (micro-)instructions determines how many times the instruction memory is accessed, while the code size of instructions determines the instruction memory size. The cost function used for application's exploration of all target processor architectures is evaluated in terms of power, performance and memory size, taking into account both the data and instruction memory hierarchy.

The existing data and instruction transfer and storage exploration methodology provides the optimal solution in terms of power performance and memory size of the physical implementation of the application, assuming a specific core. Figure 1 provides the basic steps of the existing methodology for implementing a data-intensive application on a programmable core [5, 17]. In particular, given the chosen core, the on-chip memory hierarchy as well as the performance and the power consumption (due to data and instruction transfers) can

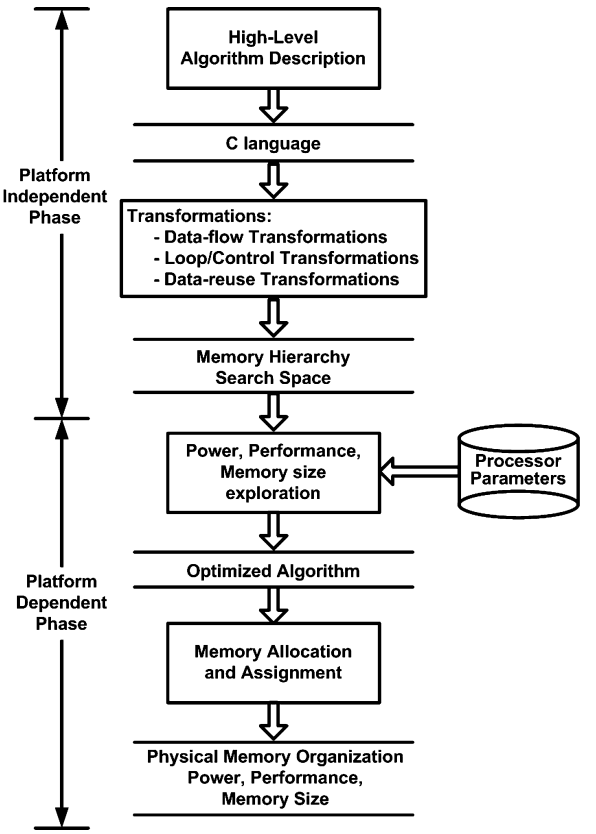

Figure 1. Steps of the existing data and instruction methodology.

be determined. The initial input is a C language description of a data-intensive application. After some preprocessing work on the source code, a first step is to apply a series of high-level transformations for optimizing the source code in terms of power and performance. These transformations are: (i) the data-flow transformations [5], (ii) the loop/control transformations [10, 18] and (iii) the data-reuse transformations [5, 22]. In particular, the data-flow transformations "enable" efficient use of the loop/control transformations. A next step is to map the optimized application to the physical memory, i.e. memory allocation and assignments [5, 18]. The use of data-reuse transformations derives a set of candidate (processor independent) memory hierarchies. Furthermore, appropriate selection of algorithmic transformations affects significantly the application performance and the power consumption [2]. Employing a series of power consumption estimation models and processor tools, we can explore and eventually, determine the power/performance/memory size efficient solution, considering the chosen programmable core.

Although the existing approaches derive optimal solutions, in terms of power and performance, and provide a physical realization, they require significant design time. Therefore, comparative studies among alternative algorithms for the implementation of a specific application will demand an increased time cost. Furthermore, early design phase decisions, for selecting the appropriate algorithm through extensive comparative and exploration studies are not possible. In addition, when a system designer develops a new algorithm for a certain application with given constraints, is not as yet provided with reliable and in-depth analysis data. However, behavioral-level estimations, in terms of critical design parameters, will be very helpful. We fill this gap, by introducing a novel methodology for power and performance estimation during the early design phases.

A theoretical, but still accurate, estimation of the execution speed and power consumption could be proved valuable, since it would heavily prune the time and effort needed for the early exploration phases. To this end, the only path available, today, is the conventional computational complexity analysis, which provides accurate estimates of the arithmetic operations needed for executing an application, for instance, the block-matching criterion evaluation [25]. Although this method can help in the case of custom hardware architectures [6, 19], where control and addressing operations can be executed in parallel with the evaluation of the block-matching criterion, it is not a reliable one in the programmable processor's domain, where in principle all instructions are sequentially executed regardless of their nature (whether they correspond to control, addressing or data processing operations). To face this problem, we introduce an analytical method that allows the estimation of the number of instructions and the number of data memory transfers involved in the execution of a data intensive application. We have derived a set of proven mathematical formulas, which estimate, first, the number of instructions executed by a specific programmable processor and, second, the number of data transfers, with great accuracy, instead of performing a simulation which is, generally speaking, a time consuming procedure. Thus, the exploration procedure requires only application-related information, as well as processor-dependent data, but not simulation of the application on the chosen programmable processor.

The proposed methodology can be "embedded" in well-established design flows, e.g. in DTSE [5], in order to reduce the exploration design time cost. Consider that a designer should choose an power- and performance-optimized algorithm, among a list of candidates, for realizing a certain application, given a specific core. For instance, selection of the appropriate Motion Estimation (ME) algorithm for video encoding. The behavioral level methodology can assist a designer obtaining quite early accurate estimations. Thus, the designer will apply an existing design methodology only to one algorithm (or sometimes to a subset of candidate algorithms). Furthermore, the proposed methodology may be useful in the selection of the appropriate programmable core for a certain application, given specific power and performance specifications. The proposed methodology can be "embedded" before (or after, depending on the required accuracy) the step of transformations of the platform-independent phase [5, 17].

A portable instruction-level profiling tool, named iprof, has been developed in [14]. The portability is achieved by reusing some functionalities of the GNU C/C++ compiler and disassembler. The iprof tool uses basic block counts to individual instructions, such as load/store, and grouping them to arithmetic, control and memory accesses. Finally, the tool reports on the executed instruction statistics of the application. The drawback is that, in order to get the statistics analysis, it is required to compile and execute the algorithm.

A novel Energy-Aware Compilation Framework that can estimate and optimize energy consumption is presented in [8]. Using the algorithmic code, together with architectural and technological parameters, power models, and power/performance constraints, the power consumption and the performance of the algorithm can be estimated. The total power consumption of the system is expressed as a sum of the power consumed in the different components, such as the datapath, caches, clock network, buses, and main memory. This technique provides variables, during the compilation time, which give a statistical report after the execution of the algorithm. This technique is based on low-level parameters, and needs the compilation and execution of the application. In contrast, our proposed methodology is a higher-level estimation, not needing compilation and execution of the algorithm to perform similar estimations.

The comparative study is focused on the performance and power dissipation, i.e. number of executed instructions and number of data memory transfers of well-known DSP benchmark applications. The proposed method uses application-related data (e.g. loop index bounds), as well as platform-dependent

information (e.g. cycles of conditional statements), and produces relatively accurate execution speed and power consumption estimates. The validity and accuracy of the proposed method is illustrated for the cases of three widely-used programmable architectures; namely, the embedded ARM7 [15] combined with an application specific memory hierarchy, the Pentium processor [9] with its fixed memory hierarchy and the 64-bits RISC processor MIPS IV [20]. The first and third implementation platform is considered as typical cases of today's system-on-chip (SoC) for multimedia processing, while the second one is a typical general-purpose platform.

The rest of the paper is organized as follows: We describe the proposed methodology, for estimating the number of instructions and the number of data memory transfers, in Section 2. Also, the algorithmic procedure operations complexity, as well as a series mathematical formulas for a number of benchmark applications, are described. We provide comparisons between the estimation results, provided by the proposed methodology, and the experimental results, obtained by commercial simulators of the programmable processors, in Section 3. Finally, we conclude with Section 4.

2. The Proposed Behavioral Formula-based Methodology

The main purpose of this paper is to estimate the fundamental design parameters of an application during the early design phases without the execution of the application itself. The power consumption of a data-intensive application consuming on a programmable system is strongly-related to the number of executed instructions and the data memory fetches. Consequently, it is sufficient for design exploration purposes to provide accurate estimates of the number (micro-) instructions and data memory transfers. In order to estimate the number of instructions and the number of data memory accesses, we propose a step by step mathematical approach (Fig. 2). Having, as input, the high-level description (e.g. C language) of a data-intensive application, we extract information from the loop structure and loop body. Then, we derive a set of formulas for arithmetic, address and control operations, having as input appropriate data of each programmable platform, and specific values (e.g. values of a loop index) from the application considered. Using these formulas, the last step includes the calculation of the number of executing instructions and the number of data memory accesses. The next sub-section describe, in detail, the new computational complexity scheme for derivation the estimation formulas.

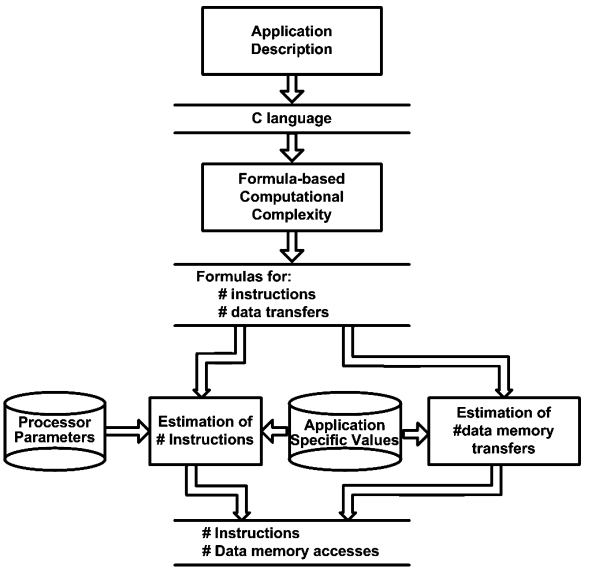

Figure 2. The proposed mathematical based approach.

2.1. *The Proposed Computational Complexity Scheme*

The computational complexity is a crucial characteristic of every application, and it has a direct impact on its execution time. For this reason, it is imperative to estimate this value in the early phases of a design process, especially when the target implementation architecture is a custom hardware. Until now, the term computational complexity referred to the arithmetic operations that occur in the algorithm [25]. For an Application Specific Integrated Circuit (ASIC) realization, this term is correctly used to identify the total complexity of the algorithm, due to the absence of commands of control or addressing operations, and no penalty in conditional statements evaluation or memory addressing. The increased design time and cost required in the case of ASICs lead to the selection of cheaper and faster implementation using programmable cores. The usage of these solutions gives rise to two new factors that have to be evaluated, in order to improve the complexity estimation: (i) the addressing and (ii) the control computational complexity. Consequently, the operations of a data-intensive application running on a programmable core are divided into three categories:

– *Arithmetic operations Complexity* (*ARC*), i.e. the operations employed only for the arithmetic computations imposed by the algorithm (e.g. computations of distance criterion in a motion estimation algorithm).
– *Addressing operations Complexity* (*ADC*), i.e. the operations employed for indexing the array data structures.
– *Control operations Complexity* (*CC*), i.e. the operations employed for implementing the control flow (for-loops, conditional statements, etc).

It should be mentioned, that all these operations are assumed to happen in a single clock cycle, and that all the basic arithmetic (e.g. $+$, $-$, *, etc) and logical (e.g. OR, AND, XOR) operations correspond to the same computational load. This assumption is valid for the programmable processors, which can compute all the above operations in the same number of clock cycles. In any other case, a weight should be assigned to each operation in respect to the number of cycles required for its execution.

The number of instructions executed by a programmable core can be estimated by adding all types of operations. That is:

$$\#instructions = ARC + ADC + CC \tag{1}$$

The number of data memory accesses can be estimated by the number of addressing operations. That is:

$$\#memory\_transfers = ADC \tag{2}$$

Although researchers have tried to measure the computational complexity of data-intensive applications for instance, in the motion estimation (ME) benchmarks, they have considered only arithmetic operations [25]. Experimental results prove that the number of addressing and control operations cannot be neglected, considering a programmable core as an implementation platform. In fact, these two parameters yield almost the same number of operations with arithmetic complexity, in a typical execution context (Fig. 3). This is a significant contribution of the present research work, because the computation and combination of these three parameters can provide a more realistic and accurate complexity estimation of the number of executed instructions, and the number of data transfers.

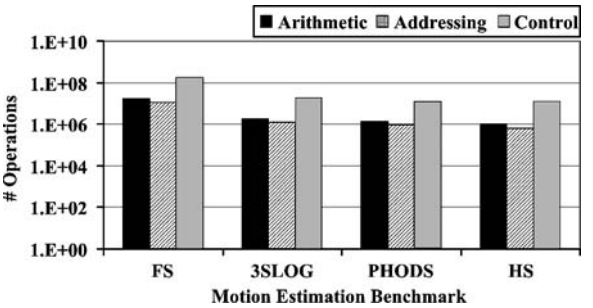

Figure 3. The number of arithmetic, addressing and control operations yield the same number of operations in the motion estimation (ME).

2.2. *Estimation Methodology and Analysis*

In order to estimate the computational complexity of all types of operations required for a specific application, described in a high-level description language (e.g., C), the following steps have to be followed (Fig. 4):

(i) The application has to be analyzed in order to extract the number and type of nested loops, together with the type of statements of each loop body.
(ii) The number of iterations of every loop has to be calculated, which can be used for estimating the number of the total execution instructions and data memory accesses.
(iii) The programmable core is chosen. The proposed methodology can be applied either in general-purpose processors or embedded processors.
(iv) Determination of processor-dependent parameters related to the complexity of the arithmetic operations/statements, the complexity of conditional (control) statements, as well as the data addressing operations.
(v) A loop index, i_k with $k \neq 1$ with lower bound, LB_k, and upper bound, UB_k, and $step_k$ can be any specific value or any algebraic expression (e.g. affine function) with terms which depend on the value of indices $\{i_1, i_2, \ldots i_{k-1}\}$. Particularly, the bound of loop index i_1 can be specific value or an input specification, for instance the frame size of an image processing application.

We have formalized our technique in mathematical formulas for calculating the Arithmetic, Address and Control Complexity. All the types of complexity were obtained by applying our proposed procedure shown, in pseudo-code, in Fig. 4.

/ Procedure for calculating complexity */*

Input */* Application Description */*
for $i_1 = LB_1$ *to* UB_1, $step_1$ *do* */* Loop 1 */*
loop body 1
for $i_2 = LB_2$ *to* UB_2, $step_2$ *do* */* Loop 2 */*
loop body 2
.
.
.
for $i_k = LB_k$ *to* UB_k, $step_k$ *do* */* Loop k*/*
$statement_{(k,1)}$ */* Arithmetic, Address or Control operation */*
$statement_{(k,2)}$
.
.
.
$statement_{(k,j)}$ */* j-th statement of k loop */*
for $i_{k+1} = LB_{k+1}$ *to* UB_k, $step_k$ *do* */* Loop k+1*/*
.
.
.

Output
- *Arithmetic Operation Complexity*
- *Addressing Operation Complexity*
- *Control Operation Complexity*

/ Computational Complexity procedure */*
- *Identify nested loops structure*
- *Determine for k-th loop index:*
  - *Lower bound of loop index,* LB_k
  - *Upper bound of loop index,* UB_k
  - *Step of loop index,* $step_k$
  - *The loops m that includes the statements of k loop*
- *Programmable core selection Determine the type of:*
  - *Arithmetic Operations for each statement of loop body (*ARC_k*)*
  - *Addressing Operations for each statement of loop body (*ADC_k*)*
  - *Control Operations for each statement of loop body (*CC_k*)*

/* Calculation of Total Arithmetic Complexity of application */

$$ARC = \sum_{\forall\, loop\ k} (ARC_k \cdot \prod_{\forall\, loop\ m} \lceil \frac{(UB_m - LB_m + 1)}{step_m} \rceil) \tag{3}$$

/* Calculation of Total Addressing Complexity of application */

$$ADC = \sum_{\forall\, loop\ k} (ADC_k \cdot \prod_{\forall\, loop\ m} \lceil \frac{(UB_m - LB_m + 1)}{step_m} \rceil) \tag{4}$$

/* Calculation of Loop Control complexity of every loop */

$$LC_{total} = LC \sum_{\forall\, loop\ k} (\prod_{\forall\, loop\ m} \lceil \frac{(UB_m - LB_m + 1)}{step_m} \rceil) \tag{5}$$

/* Calculation of Total Control Complexity of application */

$$CC = LC_{total} + \sum_{\forall\, loop\ k} (CC_k \cdot \prod_{\forall\, loop\ m} \lceil \frac{(UB_m - LB_m + 1)}{step_m} \rceil) \tag{6}$$

/* End of Procedure */

Figure 4. The proposed methodology for computational complexity estimation.

Table 1. Breakdown of the full search complexity into ARC, ADC and CC.

Complexity	Full Search (# cycles)
Arithmetic (ARC)	$(2p+1)^2 \cdot N \cdot M \cdot DCC$
Address (ADC)	$2(2p+1)^2 \cdot N \cdot M - (N+M-8B) \cdot (2p+1) \cdot (p+1) + (p+1)^2$
Control (CC)	$N \cdot M(2p+1)^2 \cdot CBC +$ $+\frac{N}{B^2} \cdot [B + 2M(p+1) + M \cdot (2p+1)^2 \cdot (1+B+B^2)] \cdot LC$

Table 2. Breakdown of the three step logarithmic complexity into ARC, ADC and CC.

Complexity	3 Step Logarithmic ($d = 2^{\lfloor \log_2 p \rfloor}$) (# cycles)
Arithmetic (ARC)	$N \cdot M \cdot (8\lceil \log_2 p \rceil + 1) \cdot DCC$
Address (ADC)	$2(8\lceil \log_2 p \rceil + 1) \cdot N \cdot M - \frac{21}{2} d \cdot (N+M) - \frac{77}{2} B \cdot d$
Control (CC)	$N \cdot M \cdot (8\lceil \log_2 p \rceil + 1) \cdot CBC + \frac{N}{B^2} \cdot [B + 2M \cdot (6\lceil \log_2 p \rceil + 1) +$ $+M \cdot B \cdot (8\lceil \log_2 p \rceil + 1) \cdot (1+B)] \cdot LC$

Table 3. Breakdown of the parallel hierarchical one-dimensional search algorithm complexity into ARC, ADC and CC.

Complexity	PHODS ($D = 2^{\lfloor \log_2 p \rfloor}$) (# cycles)
Arithmetic (ARC)	$6M \cdot N \cdot \lceil \log_2 p \rceil \cdot DCC$
Address (ADC)	$12N \cdot M \cdot \lceil \log_2 p \rceil - \frac{7}{2} D \cdot (N+M)$
Control (CC)	$6M \cdot N \cdot \lceil \log_2 p \rceil \cdot CBC + \frac{N}{B^2} \cdot [B + M \cdot (1 + \lceil \log_2 p \rceil) +$ $6M \cdot \lceil \log_2 p \rceil \cdot (1+B+B^2)] \cdot LC$

Table 4. Breakdown of the hierarchical search algorithm complexity into ARC, ADC and CC.

Complexity	Hierarchical Search ($z = \lceil p/2 \rceil$) (# cycles)
Arithmetic (ARC)	$N \cdot M \cdot [\frac{(2z+1)^2}{16} + \frac{45}{4}] \cdot DCC$
Address (ADC)	$\frac{1}{8} N \cdot M \cdot (2z+1)^2 + \frac{45}{2} N \cdot M - \frac{1}{16}(2z+1) \cdot (z+1) \cdot (N+M) +$ $+\frac{5}{2} B \cdot z^2 - \frac{B}{2} + (z+1)^2 - \frac{15}{2}(N+M) + 12B + 8$
Control (CC)	$\frac{NM}{16} \cdot [(2z+1)^2 + 180] \cdot CBC +$ $+\frac{N}{16B^2} \cdot [4B + 2M \cdot (z+1) + M \cdot (1+B+B^2) \cdot (2z+1)^2] \cdot LC +$ $+\frac{N}{B^2} \cdot [\frac{3}{2} B + \frac{65}{2} M + \frac{45}{4} M \cdot B \cdot (1+B)] \cdot LC$

In order to evaluate the proposed methodology, we have selected a number of typical data-intensive benchmark applications from different application domains. More specifically, we have chosen: *i*) four motion estimation (or block matching) benchmarks: Full Search (FS) [25], 3-Step Logarithmic (3SLOG) [11], Parallel Hierarchical One Dimensional Search (PHODS) [25], and Hierarchical Search (HS) [13] , *ii*) an image processing application: the Cavity detector [12], *iii*) a video encoding application: QSDPCM [23], and *iv*) one image compression application: the 1-D wavelet transformation [7]. The derived corresponding analytical formulas of all types of complexity, which are used for the ME benchmarks, are shown in Tables 1–4. In Appendix I, Tables 5–7 provide the appropriate formulas for the remaining three benchmarks. The mathematical formulas shown in Tables 1–7 are derived by employing the procedure shown in Fig. 4. In particular, a series of lemmas given in Appendix II prove, in a detailed manner, the correctness of the derived equations for the FS algorithm. The corresponding formulae of the remaining benchmarks can be proved in a similar way. Using equations (1) and (2), the number of (micro-) instructions and the number of data memory accesses, respectively, can be estimated.

The variables shown in Tables 1–4 have the following designation: M and N are the frame dimensions,

Table 5. Complexity from Cavity Detector algorithm.

Complexity	Cavity Detector
Arithmetic (ARC)	$2NR_{RUNS}[GB + N \cdot M + (N - 2GB)(M - 2GB)(4GB + NB + 1)]$
Address (ADC)	$NR_{RUNS}[15N \cdot M - GB(N + M)(15 + 24GB)$ $+GB(2 + 21GB + 24GB^2) + 3NB(M - 2GB)(N - 2GB)]$
Control (CC)	$NR_{RUNS}[1 - 8GB + 7N + N \cdot M(3LC + 4)$ $+(N - 2GB)(M - 2GB)[LC(4 + 6GB + NB) + NB]]$

NR_{RUNS}: The number of runs
GB: Shift parameter
NB: Block size

Table 6. Complexity from QSDPCM algorithm.

Complexity	QSDPCM
Arithmetic (ARC)	$2[N \cdot M \cdot [\frac{(2z+1)^2}{16} + \frac{45}{4}] \cdot DCC]$ $+1372\frac{N \cdot M}{B^2}$
Address (ADC)	$\frac{1}{8}N \cdot M \cdot (2z + 1)^2 + \frac{45}{2}N \cdot M - \frac{1}{16}(2z + 1) \cdot (z + 1) \cdot (N + M) + \frac{5}{2}B \cdot z^2$ $-\frac{B}{2} + (z + 1)^2 - \frac{15}{2}(N + M) + 12B + 8 + 902\frac{N \cdot M}{B^2}$
Control (CC)	$\frac{NM}{16} \cdot [(2z + 1)^2 + 180] \cdot CBC + \frac{N}{16B^2} \cdot [4B + 2M \cdot (z + 1)$ $+M \cdot (1 + B + B^2) \cdot (2z + 1)^2] \cdot LC$ $+\frac{N}{B^2} \cdot [\frac{3}{2}B + \frac{65}{2}M + \frac{45}{4}M \cdot B \cdot (1 + B)] \cdot LC + \frac{N \cdot M}{B}LC\left(\frac{3}{M} + \frac{742}{B} + \frac{1453}{B \cdot LC}\right)$

z is the Search space parameter

Table 7. Complexity from Wavelet algorithm

Complexity	Wavelet
Arithmetic (ARC)	$15hor_{l1} \cdot wer_{l1} + \frac{l}{2} \cdot ver_{l2}(57hor_{l2} + 15) + 15ver_{l3}(hor_{l3} + 2) + 75ver_{l4} \cdot hor_{l4}$
Address (ADC)	$hor_{l1}(34hor_{l1} + 8) + 46ver_{l2} \cdot hor_{l2} + 8(ver_{l2} + hor_{l2}) + 33hor_{l3} + hor_{l4}(27ver_{l4} + 1) + 8W + 17$
Control (CC)	$LC[22 + 2W + ver_{l1} + 3hor_{l1} + 4hor_{l1} \cdot ver_{l1} + 4ver_{l2} + 2hor_{l2} + hor_{l2} \cdot ver_{l2}$ $12ver_{l3} + 3hor_{l3} + \frac{13}{2}hor_{l3} \cdot ver_{l3} + 13ver_{l4} + 3hor_{l4} + \frac{33}{2}hor_{l4} \cdot ver_{l4}]$

hor_{l1} denotes Horizontal length of level 1
ver_{l1} is Vertical length of level 1
hor_{l2} is Horizontal length of level 2
ver_{l2} is Vertical length of level 2
hor_{l3} is Horizontal length of level 3
ver_{l3} is Vertical length of level 3
hor_{l4} is Horizontal length of level 4
ver_{l5} is Vertical length of level 4, and
W is Width

while p and B represent the search space, and the size of the basic square block, in numbers of pixels, respectively. The Distance Criterion Complexity (DCC) [25] parameter represents the number of the arithmetic operations required by each distance criterion. For instance, the Sum Absolute Difference (SAD) [25] criterion comprises three arithmetic operators: subtraction, absolute value and addition. Loop Complexity (LC) represents the number of operations that contain the loop check condition, which is executed in every loop cycle. Finally, the Check Bound Conditions (CBC) includes the operations needed to check if the data used are contained in the previous frame. Moreover, CBC parameter concerns the block matching algorithms *only*

and depends on the characteristics of these algorithms and the chosen programmable processor. The role of CBC parameter is to check if the search procedure (or block matching procedure) is outside the previous frame, when we compare two successive video frames for calculating the corresponding motion vectors. The DCC, CBC and LC are processor-dependent parameters and have the meaning of the numbers of cycles that the processor allocates, in order to compute each parameter.

For each analyzed DSP benchmark, mathematical formulae are developed that estimate the required numbers of processing cycles and data transfers. The mathematical formulae have an increased significance for the designer, because they allow him/her to have an abstract, but precise enough, estimation of the performance and power consumption of a DSP application, executed on a programmable processor (as we will show in Section 3).

The mathematical formulae of Table 1–7 can be used as a means to measure the execution time (e.g. number of cycles) of every application, without actually executing it, based on the characteristics of a programmable core, which are known by the published datasheets. This information should not be neglected, because it can guide the engineer to the selection, amongst many cores, of a specific core fulfilling the requirements. Furthermore, the number of memory transfers is a relevant metric of the data memory power consumption.

All developed formulae are parameterizable and depend on the specific characteristics of an application, for instance DCC and CBC, which in turn depend on the specific architecture parameters of the programmable core. All these parameters are easily extracted, making the formulas a powerful and fast complexity estimation tool for the implementor, enabling him/her to define with accuracy the processing requirements that his/her design needs, and to explore the efficiency of different architectures quickly.

3. Comparison Studies and Discussion of the Results

For comparison studies, we have chosen the ME benchmarks: FS, 3SLOG, PHODS and HS, and the programmable cores: Pentium, ARM and MIPS. Assuming the SAD block-matching criterion, the corresponding values of the parameters DCC, CBC and LC (Tables 1–4) for each processor are: 3, 1 and 2 for the Pentium [9] core; 3, 11 and 17 for the ARM 7 [15] core, and 5, 16 and 22 for the MIPS IV, respectively. These values are obtained by computing the number of instructions required by the given operation, according to the published technical datasheets. The parameters DCC, CBC and LC can be defined for every processor core, resulting in fast and accurate estimations of the executing instructions. The reason, we chose the set of ME for comparison studies is to prove that a designer can decide at a behavioral level, which algorithm is the most appropriate for realizing a video encoding application, under certain constraints.

Figs. 5, 6 and 7 illustrate the measurements of the computational complexity using the conventional method (i.e. only the arithmetic complexity), that was used until now, and the proposed methodology, as well as the measured results obtained from the actual simulation of the applications, for three different processor architectures. Simulations were carried out using Vtune [9], ARMulator [15] and Simplescalar [1], for Pentium, ARM and MIPS, respectively. In particular, Fig. 5 presents the measurements for the Pentium processor core, together with the estimated values obtained using the proposed and the conventional method. The results show that the proposed method overestimates the number of instructions compared to the conventional method which underestimates them. This can be clarified by the fact that Pentium has the ability to execute more the one (micro-)instructions in one clock cycle. For this reason, the real number of executed cycles in the experimental measurements is lower than that computed by our methodology. The average overestimation is about 35%, while the corresponding values for FS, 3SLOG, PHODS and HS are 35%, 27%, 31% and 48%, respectively. The latter two algorithms exhibit larger overestimation values, due to their inherent structure, which comprises more than one regular nested loop. The average underestimation of the conventional method for Pentium processor core is 56%, which is larger than the proposed method. In other words, the irregular loop structure implies increased number of operations, including the loop check condition. The estimation values of the first two algorithms are more accurate due to their regular nested loop structure, while experimental results shows that the difference in proposed method accuracy, applying or not, global loops transformations is about 3%.

In 6, the measurements concerning the computational complexity of the ARM processor are presented. It can be deduced that the proposed method gives a good approximation of the experimental

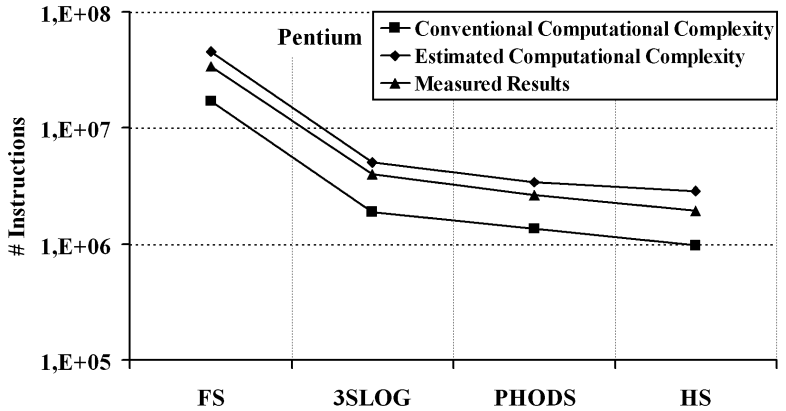

Figure 5. Our estimation technique is more accurate than the conventional technique in the Pentium architecture.

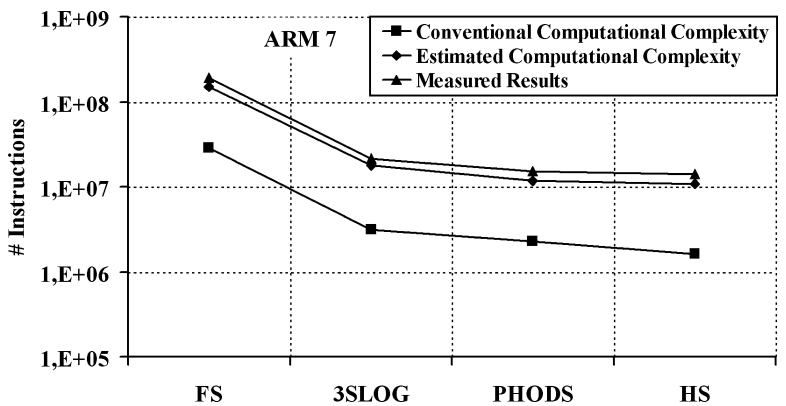

Figure 6. Our estimation technique is more accurate than the conventional technique in the ARM architecture.

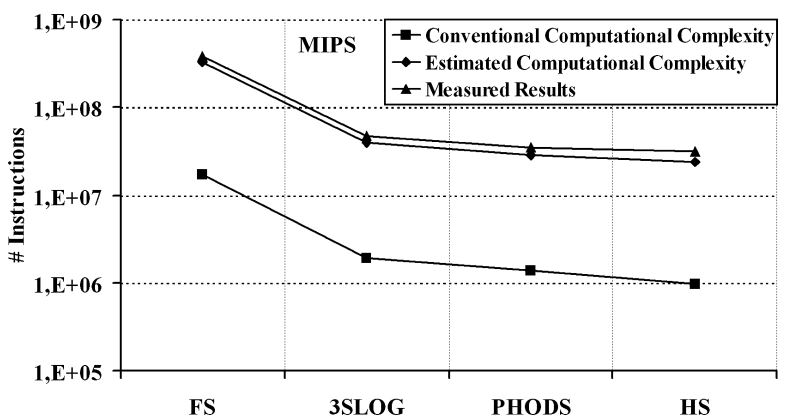

Figure 7. Our estimation technique is more accurate than the conventional technique in the MIPS architecture.

measurements. On the other hand, the conventional method presents a significant deviation. The average underestimation is about 18%, while the corresponding values for FS, 3SLOG, PHODS and HS are 21%, 14%, 21% and 14%, respectively. ARM is a RISC processor, which means that the calculation of DCC, CBC and LC values are very accurate and very close to the reality. The accurate definition of these crucial parameters leads to measurements that are very close to real number of execution instructions. The conventional method provides results with an average underestimation of 83%, which much larger than the corresponding estimation figure of 18% of the proposed approach.

Using the Simplescalar simulator tool we obtained the results for the MIPS processor (Fig. 7). As these results show, we have a good approximation of the experimental measurements, while the conventional method exhibits a significant deviation. The average underestimation of the conventional method is close to 17%, while the individual values for FS, 3SLOG, PHODS and HS are 13%, 14%, 17% and 24%, respectively. Moreover, the average underestimation of the conventional method for MIPS processor core is 96%, which proves that the proposed method gives better results on the estimation of the number of executed instructions.

The previous analysis shows that fast and quite accurate estimations can be achieved by the accurate definition of the crucial parameters DCC, CBC and LC. The accuracy of these parameters has a direct impact on the proposed method. In a RISC architecture, these parameters can easily be defined, due to its inherent simple execution scheme, whereas in CISC architectures this is not so straightforward, and more research and analysis is required. The proposed method could be extended to every algorithm, for every processor core, enabling the designer to have a very good estimation of the number of instructions.

In Fig. 8, the measurements on the number of data memory accesses of ME applications are presented. Since the estimation is based on the fact that memory transfers are dominated by array variables, the actual estimations for "write" and "read" operations of an array element can be immediately extracted by the source code application. In other words, the memory transfers are independent from the chosen programmable core. Fig. 8 shows the number of data memory accesses, as measured by the industrial tool Atomium [4], and the accesses estimated by the proposed methodology. Comparing these two curves, we can deduce the good accuracy of our technique in estimating the number of data memory accesses.

The regular or irregular structure of an application affects its total performance and power consumption. Indeed, it is has been proved that the use of algorithmic transformations affect the total performance and power consumption [5]. In order to prove that both the regular and irregular structure of the same application has no (almost) affect the accuracy of the proposed methodology, we performed the same measurements with the "regular" and the "irregular" (original) version of PHODS and HS applications. Applying Global Loop Transformations (i.e. loop merging) [5] we can produce perfectly nested loops. Considering ARM processor, the comparison results between the proposed methodology and the simulated version of the original application show that there is a small difference

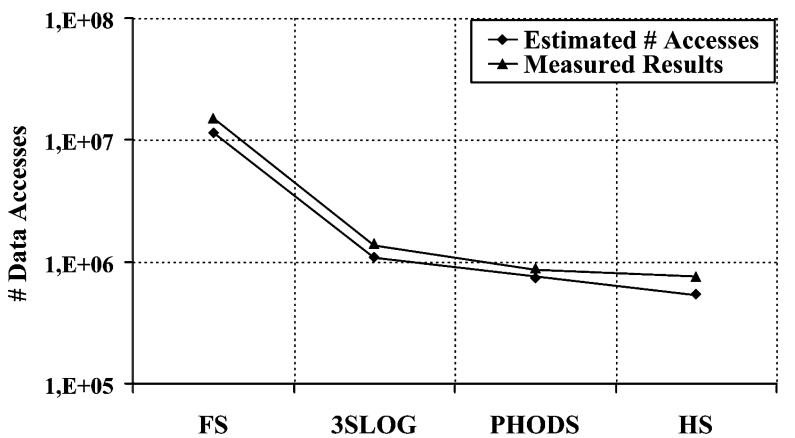

Figure 8. Our estimated number of data accesses follow closely the measured number of accesses.

between the estimations of the proposed methodology and the simulation results of the "transformed" (or regular) version of the application. More specifically, for PHODS application, the corresponding underestimation numbers are 21% and 18%, respectively. Similarly, for HS the corresponding figures are 14% and 12%, respectively. It can be easily concluded that the regular or the irregular structure of the same application does not affect the accuracy of the proposed methodology. However, the structure of an application affects only the absolute numbers of the #instructions and data transfers for each version of the application. Furthermore, the small differences between the above mentioned comparisons stem from the accuracy of the simulation results.

Considering the number of widely-used programmable processors (general-propose and embedded), the effort required to determine core parameters is not so great, and the designer can re-use the same processor-dependent information for new DSP applications. In that sense, a designer can build a library with specific data for each programmable core.

4. Conclusions

The development of a high-level estimation methodology of the number of the executed instructions and the number of the memory accesses, using a novel computational complexity analysis was described. The improved accuracy of the presented architecture, compared to other techniques of exploration of data intensive applications, arises from a novel and detailed computational complexity analysis. Indeed, the new complexity analysis comprises both the mathematical features of the applications considered (e.g. number of arithmetic calculations, type of operators) and the hardware requirements of a programmable platform. Until now, researchers had suggested the use of a number of arithmetic operations as the only way to calculate complexity. In the programmable processor implementation, two new complexity parameters have to be taken into account; addressing and control, which were not taken before into consideration. Experimental results proved the significance of these parameters, and index them as equally important with the parameters of arithmetic operations. The sum of the arithmetic, addressing and control information is the real computational complexity of the application, which is translated to real execution time (processor cycles), required for the implementation on a specific core. Comparison results on the number of executed instructions, among four ME algorithms, proved that the proposed methodology can provide estimation with acceptable level of accuracy. This work revealed the significance of selecting the most suitable algorithm in order to implement, for instance, a multimedia system. Having at his/her disposal a plethora of alternative solutions/implementations of applications in an early design phase, a designer can easily be to the selection of the most suitable application meeting certain design constraints.

APPENDIX I: Computational Complexity Mathematical Formulae

Additional formulae for calculating the arithmetic, the address, and the control operations are provided for three well known applications, namely Cavity Detector [12] (image processing), QSDPCM [23] (video encoding) and 1-D Wavelet [7] (image compression). The corresponding formulae for estimating the instruction complexity and the number of data memory accesses for each application are shown in Table 5, 6 and 7, respectively.

In particular, the algorithm of Cavity detection belongs to the medical image processing domain. It is used to detect tumor cavities in computer tomography pictures of the brain. Compared with the other applications which are block based, this algorithm performs computation on the granularity of a pixel. It consists of a horizontal Gaussian blur, followed by a vertical Gaussian blur. A computation of the edges and a image inversion follow. The application ends with a function that performs root detection. The QSDPCM uses a hierarchical search and implements a full video encoder. It is an interframe compression technique for video images. It involves a motion estimation step, and a quadtree based encoding of the motion compensated frame-to-frame difference signal. Finally, we include the analytical formulas for a 5-level 9/7 tap row/column inverse wavelet transformation algorithm, which widely used in JPEG-2000 or MPEG-4 standards.

APPENDIX II: Mathematical Formulae for Full Search Algorithm

With the term Arithmetic Complexity (ARC), we denote the computational complexity of performing the fundamental arithmetic operations, e.g. addition and

multiplication, while Addressing Complexity (ADC) exists every time there is a memory access to off-chip or on-chip data memory hierarchy. Finally, Control Complexity (CC) is the complexity imposed by the control conditions of the algorithm, for instance, the check bounds condition.

Here, we consider the following Motion Estimation Algorithms: *i*) Full Search (FS) [25], *ii*) 3-Step Logarithmic Search (3SLOG) [11], *iii*) Parallel Hierarchical One-Dimensional Search (PHODS) [25], and *iv*) Hierarchical Search (HS) [13]. All these algorithms are block based, which means that in order to compute the motion vectors between two successive frames, the previous frame and the current frame are divided into a number of blocks, each of which have a fixed number of pixels. The algorithm compares every block from the current frame with the block from previous frames that are placed near the original point. Considering a cost function, the best match between a previous and a current block frame is that with minimum value in the cost function. This results in the motion vectors $MV_{i,x}$, $MV_{i,y}$, of the i-th block and x- and y-axis, respectively.

Since the match is being performed between rectangular regions, the cost function is referred to as a block matching criterion (BMC), where as the search techniques to find the motion vectors, which yield the smallest cost, are referred to as block matching algorithms (BMA).

The simplest method to compute the motion vectors of successive frames is to compute the cost function at each location in the search space (Fig. 9). This is referred to as the Full-Search (FS) algorithm. Thus, for every block of the previous frame, an exhaustive search

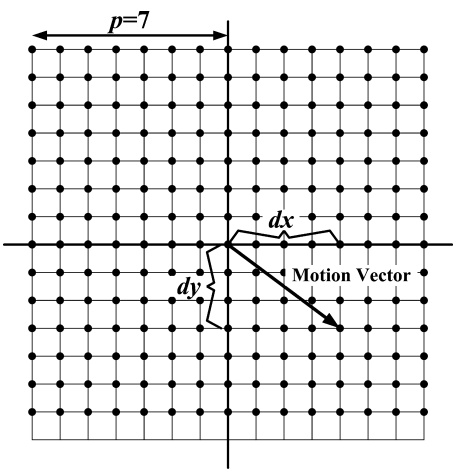

Figure 9. Block matching strategy in full search algorithm.

in all the nearby blocks of the current frame within a radius p, called candidate blocks, is being made, in order to find the current block that yields the smallest cost (or minimizes the cost function of the distance criterion). A pseudo code for the FS is illustrated in Fig. 10. The simplicity of this code can be seen in its C language description, where three sets of regular loops exist: *i*) one set allows to compute the motion vectors for every block of the previous frame, *ii*) one set allows searching within the search space in the current frame, and *iii*) one set allows performing calculations for the distance criterion for every pixel of the two frames. Although FS is a computationally-expensive algorithm, it guarantees finding the blocks that yield the minimum cost value. Due to the high computational complexity, which will be discussed in the following section, alternative search methods are desirable.

The analysis and justification of the mathematical formulas of the three types of complexity of the Table I for an indicative an application, namely FS, will be presented in the following paragraphs. The formulas of the remaining benchmarks, can be proved with similar fashion.

Lemma 1: Let N and M be the frame dimensions, and p and B be the search space parameter and the block size of a frame, respectively. The computational complexity of FS algorithm in terms of arithmetic operations, ARC, is obtained by:

$$ARC = (2p+1)^2 \cdot N \cdot M \cdot DCC \tag{7}$$

where DCC is the arithmetic complexity of a distance criterion.

Proof: The arithmetic operations that exist in the ME algorithms are only due to operations of a distance criterion. The number of block comparisons between each block from current and previous frame are: $(2p+1)^2$, where p the search space parameter, and $(2p+1)$ are the possible search candidate blocks in every direction. Since B is the block size, the total number of the calculations of a distance criterion is: $B \cdot B \cdot (2p+1)^2$. Every frame is divided into $(N/B) \cdot (M/B)$ blocks, which means that if the arithmetic complexity of a distance criterion is DCC, then the total arithmetical computational complexity is:

Using the proposed methodology (Fig. 4) we estimate the ARC of the FS algorithm. From the pseudocode (Fig. 10) we see that the algorithm consists of six nested loops. Every loop is related with an equation

```
for(k=0;k<N/B;k++)       // loop 1, LB1=0, UB1=N/B, step1=1
 for(l=0;l<M/B;l++)      // loop 2, LB2=0, UB2=M/B, step2=1
  {
  for(i=-p;i<p+1;i++)    // loop 3, LB3=-p, UB3=p+1, step3=1
   for(j=-p;j<p+1;j++)   // loop 4, LB4=-p, UB4=p+1, step4=1
    {
    for(m=0;m<B;m++)     // loop 5, LB5=0, UB5=B, step5=1
     for(n=0;n<B;n++)    // loop 6, LB6=0, UB6=B, step6=1
      {
      read_from_current_frame;  // Address operation
      Check_Bound_Condition;    // Control operation
      read_from_previous_frame; // Address operation
      Distance_Criterion();     // Arithmetic operation
      }
    }
 }
```

Figure 10. Pseudo-code for full-search application.

that computes the number of iterations of each loop, as follows:

$$\lceil \frac{(UB_1 - LB_1 + 1)}{step_1} \rceil = N/B \tag{8}$$

$$\lceil \frac{(UB_2 - LB_2 + 1)}{step_2} \rceil = M/B \tag{9}$$

$$\lceil \frac{(UB_3 - LB_3 + 1)}{step_3} \rceil = 2 \cdot p + 1 \tag{10}$$

$$\lceil \frac{(UB_4 - LB_4 + 1)}{step_4} \rceil = 2 \cdot p + 1 \tag{11}$$

$$\lceil \frac{(UB_5 - LB_5 + 1)}{step_5} \rceil = B \tag{12}$$

$$\lceil \frac{(UB_6 - LB_6 + 1)}{step_6} \rceil = B \tag{13}$$

For the pseudo-code (Fig. 10) we see that from the six loops only the last one ARC_6 has arithmetic complexity (all the other loops have arithmetic complexity of 0). The arithmetic complexity of sixth loop is equal to the complexity of DCC (ARC_6 = DCC). Thus, for Eq. 3 of the proposed methodology we have

$$\begin{aligned} ARC &= N/B \cdot M/B \cdot (2p+1) \cdot (2p+1) \cdot N \cdot M \cdot DCC \\ &= (2p+1)^2 \cdot N \cdot M \cdot DCC \end{aligned} \tag{14}$$

$$\begin{aligned} &ARC = [B \cdot B \cdot (2p+1)^2] \cdot [(N/B) \cdot (M/B)] \cdot DCC \Leftrightarrow \\ &ARC = (2p+1)^2 \cdot N \cdot M \cdot DCC \end{aligned} \tag{15}$$

For example, if a designer selects the SAD BMC to employ in a ME algorithm, then DCC equals to 3 operations, one substraction, one absolute value computation and one addition.

Lemma 2: Let N and M be the frame dimensions, and p and B be the search space parameter and the block size of a frame, respectively. The number of execution instructions of FS algorithm in terms of addressing operations, ADR, is given by:

$$ADR = 2(2p+1)^2 \cdot N \cdot M - \\ -(N+M-8B) \cdot (2p+1) \cdot (p+1) + (p+1)^2 \quad (16)$$

Proof: The addressing complexity is the number of accesses to the data memory hierarchy. As the mechanism of FS compares the blocks, for each current frame block located to the frame edge, the ME algorithm should not read data locating outside the frame. However, each block of the current frame, which is located at the boundary of the current frame is compared with a block of the previous frame, some of which contain data locating outside the previous frame. The check bound condition (CBC) checks if the pixel of a block is located or not in the previous frame. In the latter case, we use the zero value for calculations, therefore, there is no access to data memory. The total number of memory accesses to the current frame is:

$$N_{cur\_access} = (2p+1)^2 \cdot N \cdot M \quad (17)$$

Generally speaking, the total number of data from the previous frame required for ME calculations is: $(2p+1)^2 \cdot N \cdot M$. However, the total number of accesses to the data memory is smaller than $(2p+1)^2 \cdot N \cdot M$, because we have data with zero values, i.e. outside the frame. In order to calculate the exact number of memory access of the previous frame, it is sufficient to compute the number of data with zero values. For that purpose the previous frame blocks are categorized into three sets as follows (Fig. 11):

1. The set of internal blocks is defined as:

$$Internal\_blocks = \{block[k][l] \mid k = \\ 1, 2, \ldots, \frac{N}{B} - 2 \bigwedge l = 1, 2, \ldots, \frac{M}{B} - 2\}.$$

2. The set of boundary blocks is defined as:

$$Boundary\_blocks = \{block[k][l] \mid (k = \\ 1, 2, \ldots, \frac{N}{B} - 2 \bigwedge l = 0, \frac{M}{N} - 1), (k = \\ 0, \frac{N}{B} - 1 \bigwedge l = 1, 2, \ldots, \frac{M}{B} - 2)\}$$

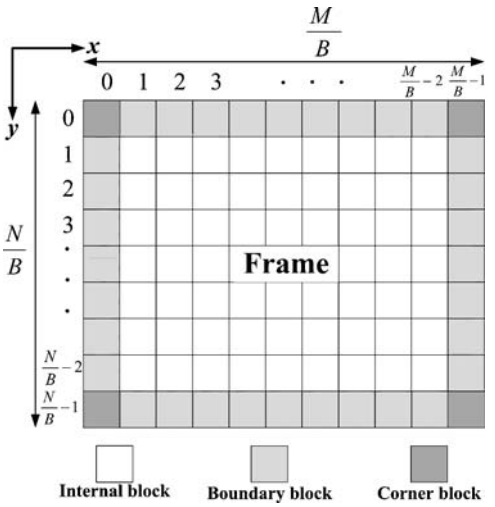

Figure 11. Three kinds of blocks.

3. The set of corner blocks is defined as:

$$Corner\_blocks = \{block[k][l] \mid (k = \\ 0, l = 0), (k = \frac{N}{B} - 1 \bigwedge l = 0), (k = \\ \frac{N}{B} - 1 \bigwedge l = \frac{M}{B} - 1), (k = 0 \bigwedge l = \frac{M}{B} - 1)\}$$

The total number of data, which are used for processing is: $2 \cdot (2p+1)^2 \cdot N \cdot M$. The half of them concern the data of the current frame, while the remaining data of the previous frame. In the following paragraphs, we will compute the total number of data with zero value, which are associated to the three sets of blocks.

1. *Internal blocks*. Having in mind Fig. 11, the internal blocks of the previous frame use always those pixels, which are located in the frame, for every possible position of a previous block. Consequently, the total number of the used data with zero value is:

$$S_{int} = 0 \quad (18)$$

2. *Boundary blocks*. The cardinality, $\| \cdot \|$, of the *Boundary_blocks* set, equals to the total number of boundary blocks. That is: $\| Boundary\_blocks \| = 2(N+M-4B)/B$. It is sufficient to calculate the number of those data with zero values for ME processing of (k, l) block.

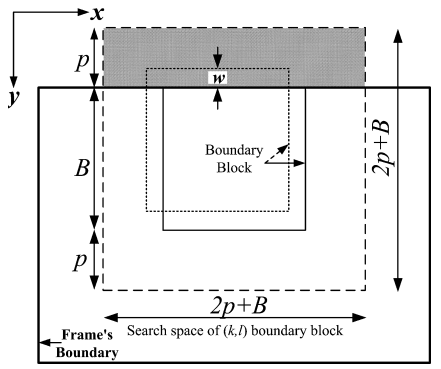

Figure 12. We take into consideration the missed accesses in boundary blocks for better accuracy.

Let w be the external displacement of (k, l) block with respect to the frame's boundary, as it is shown in Fig. 12. The range of w is $1 \leq w \leq p$. For w pixel displacement, there exist $2p + 1$ blocks located on the axis l. Taking into account that $B \cdot B$ is the block size, the associated number of the zero data is: $(2p+1)\cdot w \cdot B$. Hence, for all displacements, w, $1 \leq w \leq p$, the total number of zero data of (k, l) block is given by:

$$S_{bou}^{block} = \sum_{w=1}^{p} (2p+1) \cdot w \cdot B \Leftrightarrow S_{bou}^{block} = \frac{(2p+1) \cdot (p+1) \cdot B}{2} \tag{19}$$

Eventually, the number of zero data of all boundary blocks, i.e. $2 \cdot (N + M - 4B)/B$ is:

$$S_{bou} = (2p+1) \cdot (p+1) \cdot (N + M - 4B) \tag{20}$$

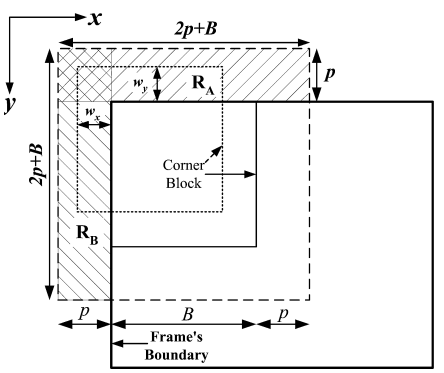

Figure 13. We take into consideration the corner blocks for better accuracy.

3. *Corner blocks*. In order to calculate the number of a zero data of a corner block, we partition the search space area outside the boundary frame into the two regions, R_A, and R_B, as it is illustrated by the shadowed regions of Fig. 13. R_A and R_B are defined by the quadratuple of pairs: $\{(-p, -p), (B + p, -p), (B + p, 0), (-p, 0)\}$ and $\{(-p, -p), (0, -p), (0, B + p), (-p, B + p)\}$, respectively. Taking into account, the proof for boundary blocks, the number of zero data of R_A and R_B are obtained by:

$$S_{R_A}^{block} = S_{R_B}^{block} = \frac{(2p+1) \cdot (p+1)}{2} \cdot B \tag{21}$$

Apparently, the intersection $(R_A \cap R_B)$ of R_A and R_B regions results into a region, which is defined by $\{(-p, -p), (0, -p), (0, 0), (-p, 0)\}$. Therefore, in order to calculate the total number zero data, S_{cor}, it is sufficient to compute the number of zero data of $R_A \cap R_B$. That is:

$$S_{cor}^{block} = S_{R_A}^{block} + S_{R_B}^{block} - S_{R_A \cap R_B}^{block} \tag{22}$$

Let w_k and w_l be the displacements of a corner block for axes k and l with respect to the frame's boundary, respectively. The range of their integer values are: $-p \leq w_k \leq p$ and $-p \leq w_l \leq p$. Then, the total number $S_{R_A \cap R_B}^{block}$ is given by:

$$S_{R_A \cap R_B}^{block} = \sum_{w_k=1}^{p} \sum_{w_l=1}^{p} w_k \cdot w_l \Leftrightarrow \tag{23}$$

$$S_{R_A \cap R_B}^{block} = \frac{(p+1)^2}{4} \tag{24}$$

Substituting eq. (21) and (24) into eq. (22), we obtain that:

$$S_{cor}^{block} = (2p+1) \cdot (p+1) \cdot B - \frac{(p+1)^2}{4} \tag{25}$$

Eventually, for all corners blocks, which are four blocks, the overall number of zero data is:

$$S_{cor} = 4(2p+1) \cdot (p+1) \cdot B - (p+1)^2 \tag{26}$$

Employing the values, S_{int}, S_{bou} and S_{cor}, the total number of zero values is:

$$S = S_{int} + S_{bou} + S_{corner} \Leftrightarrow \tag{27}$$

$$S = (2p+1) \cdot (p+1) \cdot (N + M - 4B) + 4(2p+1) \cdot (p+1) \cdot B - (p+1)^2 \tag{28}$$

Remembering that the total number of memory accesses regarding with the data of previous frame is smaller than $(2p+1)^2 \cdot N \cdot M$, because we have "zero data." Using Eq. (28), the number of memory accesses of previous frame is:

$$N_{pre\_access} = (2p+1)^2 \cdot N \cdot M - S \tag{29}$$

Consequently, the total number of the memory accesses to the current frame, N_{cur_access}, and previous frame, N_{pre_access}, determines the total number of addressing operations, *ADR*. Hence, using Eq. (17) and (29), we obtain that:

$$ADR = N_{cur\_access} + N_{pre\_access} \Leftrightarrow \tag{30}$$

$$\begin{aligned} ADR = {} & 2(2p+1)^2 \cdot N \cdot M \\ & -(N+M-8B) \cdot (2p+1) \\ & \cdot(p+1) + (p+1)^2 \end{aligned} \tag{31}$$

Lemma 3: Let N and M be the frame dimensions, and p and B be the search space parameter and the block size of a frame, respectively. The total number of execution instructions of FS algorithm in terms of control operations, *CC*, is given by:

$$\begin{aligned} CC = {} & N \cdot M \cdot (2p+1)^2 \cdot CBC \\ & + \frac{N}{B^2} \cdot [B + 2M \cdot (p+1) \\ & + M \cdot (2p+1)^2 \cdot (1 + B + B^2)] \cdot LC \end{aligned} \tag{32}$$

where *LC* and *CBC* are the complexity of a single loop condition and a signal of check bound condition, respectively.

Proof: This is the third type of computational complexity in the ME algorithms, which is caused by the control conditions of a algorithm. ME algorithm have two types of control conditions: (a) the check bound condition (CBC) and (b) the loop condition (LC). There is a check bound control of the accesses happening in the previous frame, whether a pixel is outside the frame boundary. Hence, the CBC condition is executed:

$$ExecNumb_{CBC} = N \cdot M \cdot (2p+1)^2 \tag{33}$$

times, because it exists in the innermost loop. That means that the complexity of CBC parameter can be computed by:

$$CC_{CBC} = N \cdot M \cdot (2p+1)^2 \cdot CBC \tag{34}$$

The second type of complexity, LC, occurs in every loop. Thus, the complexity of the most external loop index k (Fig. 10) is:

$$LC_k = LC \cdot N/B \tag{35}$$

Similarly, the complexities for the remaining loop indexes, l, i, j, m, and n are:

$$LC_l = LC \cdot N/B \cdot M/B \tag{36}$$

$$LC_i = LC \cdot N/B \cdot M/B \cdot (2p+1) \tag{37}$$

$$LC_j = LC \cdot N/B \cdot M/B \cdot (2p+1)^2 \tag{38}$$

$$LC_m = LC \cdot N/B \cdot M \cdot (2p+1)^2 \tag{39}$$

$$LC_n = LC \cdot N \cdot M \cdot (2p+1)^2 \tag{40}$$

respectively. Therefore, the total Loop Control Complexity, is the sum of Eq. (35)–(40). Thus:

$$\begin{aligned} LC_{total} = {} & \frac{N}{B^2} \cdot [B + 2M \cdot (p+1) \\ & + M \cdot (2p+1)^2 \cdot (1 + B + B^2)] \cdot LC \end{aligned} \tag{41}$$

Finally, using Eq. (34) and (41), the total number of execution instructions is given by:

$$CC = CC_{CBC} + LC_{total} \Leftrightarrow$$

$$\begin{aligned} CC = {} & N \cdot M \cdot (2p+1)^2 \cdot CBC \\ & + \frac{N}{B^2} \cdot [B + 2M \cdot (p+1) \\ & + M \cdot (2p+1)^2 \cdot (1 + B + B^2)] \cdot LC \end{aligned} \tag{42}$$

References

1. Austin, T., *The SimpleScalar Tool Set, version 3.0*. SimpleScalar LLC.
2. D. Bacon, S. Graham and O. Sharp, "Compiler transformations for high-performance computing." *ACM Comput. Surv.* **26**(4), 1994, pp. 345–420.
3. D. Soudris, N. Zervas, A. Argyriou, M. Dasygenis, K. Tatas, C. Goutis and A. Thanailakis, "Data-Reuse and Parallel Embedded Architectures for Low-Power, Real-Time Multimedia Applications." *in Proc. of 10th Int. Workshop PATMOS 2000 Gottigen, Germany*, 2000, pp. 243–254.

4. F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele and H. De Man, *Global communication and memory optimizing transformations for low power signal processing systems*, VLSI Signal Processing VII. New York: IEEE Press, 1994.
5. F. Catthoor, K. Danckaert, K.K. Kulkarni, E. Brockmeyer, P. G. Kjeldsberg, T. van Achteren and T. Omnes, *Data Access and Storage Management for Embedded Programmable Processors*. Kluwer Academic Publishers, Boston, 2002.
6. F. Catthoor, S. Wuytack, G.E. de Greef, F. Banica, L. Nachtergaele and A. Vandecappelle, *Custom Memory Management Methodology*. Kluwer Academic Publishers, Boston, 1998.
7. G. Lafruit, L. Nachtergaele, B. Vahnhoof and F. Catthoor, "The Local Wavelet Transform: A Memory-Efficient, High-Speed Architecture Optimized to a Region-Oriented Zero-Tree Coder." *Integrated Computer-Aided Engineering* **7**(2), 2000, pp. 89–103.
8. I. Kadayif, M. Kandemir, N. Vijaykrishnan, M. J. Irwin and A. Sivasubramaniam, "EAC: A Compiler Framework for High-Level Energy Estimation and Optimization." *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition*, 2002, pp. 436–442.
9. Intel, *VTune Performance Analyzer*.
10. J. Ramanujam, J. Hong, M. Kandemir and Narayan, "Reducing Memory Requirements of Nested Loops for Embedded Systems." *38th Automation Conference, DAC, Las Vegas, NV, USA*, 2001, pp. 359–364.
11. J. Jain, and A. Jain, "Displacement measurement and its applications in intraframe image coding." *in IEEE Trans. Communications* 29, 1981.
12. K. Danckaert and F. Catthoor and H. De Man, "Platform independent data transfer and storage exploration illustrated on a parallel cavity detection algorithm." *ACM Conference on Parrallel and Distributed Processing Techniques and Applications* III, 1999, pp. 1669–1675.
13. K.M. Nam, J.-S. Kim, Rae-Hong Park and Y. S. Shim, "A fast hierarchical motion vector estimation algorithm using mean pyramid." *IEEE Transactions on Circuits and Systems for Video Technology* **5**(4), 1995, pp. 344–351.
14. P. Kuhn, *Algorithms, Complexity Analysis and VLSI, Architectures for MPEG-4 Motion Estimation*. Kluwer Academic Publishers, 1999.
15. A.R.M. Ltd,*ARM7 data sheet*. ARM Project Manager for Windows, 1994.
16. M. Dasygenis, N. Kroupis, A. Argyriou, K. Tatas, D. Soudris and N. Zervas, "A memory management approach for efficient implementation of multimedia kernels on programmable architectures." *in IEEE Computer Society Annual Workshop on VLSI (WVLSI) Orlando Florida, USA*, 2001, pp. 171–176.
17. M. Dasygenis, N. Kroupis, K. Tatas, A. Argyriou, D. Soudris and A. Thanailakis, "Power and Performance Exploration of Embedded Systems Executing Multimedia Kernels." *IEE Proc.-Comput. Digit. Tech.* **149**(4), 2002, pp. 164–172.
18. P. Panda, F. Catthoor, N. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle and P. G. Kjeldsberg, "Data and memory optimization techniques for embedded systems." *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 6, 2001.
19. P. Pirsch, N. Demassieux and W. Gehrke, "VLSI Architectures for Video Compression: A Survey." *Proceedings of IEEE* **83**(2), 1995, pp. 220–246.
20. C. Price, *MIPS IV Instruction Set, revision 3.1*. Technologies, Inc., Mountain View, CA.
21. J. Rabaey, and M. Pedram *Low Power Design methodologies*. Kluwer Academic Publishers, Boston, 1999.
22. S. Wuytack, J. Diguet, F. Cattohoor and H. De Man, "Formalized methodology for data reuse: exploration for low-power hierarchical memory mappings." *IEEE Transactions on VLSI Systems* **6**(4), 1998, pp. 529–537.
23. P. Strobach, "A New Technique in Scene Adaptive Coding." *Proc. 4th Eur. Signal Processing Conf*, 1998, pp. 1141–1144.
24. T. Araki, M. Toyokura, T. Akiyama, H. Takeno, B. Wilson and K. Aono, "Video DSP architecture for MPEG2 codec." *in Proc. of International Conference On Acoustics, Speech, and Signal Processing* **2**, 1994, pp. 417–420.
25. V. Bhaskaran, K. Kostantinides *Image and Video Compression Standards*. Kluwer Academic Publishers, 1998.

Nikolaos Kroupis was born in Trikala in 1976. He receiver the engineering degree and Ms.C. degree in Department of Electrical and Computer Engineering from Democritous University of Thrace, Greece, in 2000 and 2002, respectively. Since 2002 he has been a Ph.D. student at the Laboratory of Electrical and Electronic Materials Technology. His research interests are in software/hardware co-design of embedded system for signal processing applications.

Nikos D. Zervas received a Diploma in Electrical & Computer Engineering from University of Patras, Greece in 1997. He received the Ph.D. degree in the Department of Electrical and Computer Engineering of the same University in 2004. His research interests are in the area of high-level, power optimization techniques and methodologies for multimedia and telecommunication applications. He has received an award from IEEE Computer Society in the context of Low-Power Design Contest of 2000 IEEE Computer Elements Mesa Workshop. Mr. Zervas is a member of the IEEE, ACM and of the Technical Chamber of Greece.

Minas Dasygenis was born in Thessaloniki in 1976. He received his Diploma in Electrical and Computer Engineering in 1999, from the Democritus University of Thrace, Greece, and for his diploma Thesis he was honored by The Technical Chamber of Greece and Ericsson Hellas. In 2005, he received his PhD Degree from the Democritus University of Thrace. His research interests include low-power VLSI design of arithmetic circuits, residue number system, embedded architectures, DSPs, hardware/ software codesign and IT security. He has published more than 20 papers in international journals and conferences and he has been a principal researcher in three European research projects.

Konstantinos Tatas received his degree in Electrical and Computer Engineering from the Democritus University of Thrace, Greece in 1999. He received his Ph.D. in the VLSI Design and Testing Center in the same University by June 2005. He has been employed as an RTL designer in INTRACOM SA, Greece between 2000 and 2003. His research interests include low-power VLSI design of DSP and multimedia systems, computer arithmetic, IP core design and design for reuse.

Antonios Argyriou received the degree in Electrical and Computer engineering from the Democritous University of Thrace, Greece, in 2001, and the M.S. and Ph.D. degrees in Electrical and Computer engineering from the Georgia Institute of Technology, Atlanta, in 2003 and 2005, respectively. His primary research interests include wireless networks, mobile computing and multimedia communications. He is a member of the IEEE and ACM.

Dimitrios Soudris received his Diploma in Electrical Engineering from the University of Patras, Greece, in 1987. He received the Ph.D. Degree in Electrical Engineering, from the University of Patras in 1992. He is currently working as Ass. Professor in Dept. of Electrical and Computer Engineering, Democritus University of Thrace, Greece. His research interests include low power design, parallel architectures, embedded systems design, and VLSI signal processing. He has published more than 140 papers in international journals and conferences. He was leader and principal investigator in numerous research projects funded from the Greek Government and Industry as well as the European Commission (ESPRIT II-III-IV and 5th and 6th IST). He has served as General Chair and Program Chair for the International Workshop on Power and Timing Modelling, Optimisation, and Simulation (PATMOS). He received an award from INTEL and IBM for the project results of LPGD #25256 (ESPRIT IV). He is a member of the IEEE, the VLSI Systems and Applications Technical Committee of IEEE CAS and the ACM.

Antonios Thanailakis was born in Greece on August 5, 1940. He received B.Sc. degrees in physics and electrical engineering from the University of Thessaloniki, Greece, 1964 and 1968, respectively, and the Msc. and Ph.D. Degrees in electrical engineering and electronics from UMIST, Manchester, U.K. in 1968 and 1971, respectively. He has been a Professor of Microelectronics in Dept. of Electrical and Computer Eng., Democritus Univ. of Thrace, Xanthi, Greece, since 1977. He has been active in electronic device and VLSI system design research since 1968. His current research activities include microelectronic devices and VLSI systems design. He has published a great number of scientific and technical papers, as well as five textbooks. He was leader for carrying out research and development projects funded by Greece, EU, or other organizations on various topics of Microlectronics and VLSI Systems Design (e.g. NATO, ESPRIT, ACTS, STRIDE).