A Novel Method for Improving the Operation Autonomy of SIMD Processing Elements

Manuel Lois Anido[1], Alexander Paar[2] and Nader Bagherzadeh[3]

[1]*Federal University of Rio de Janeiro, NCE*
mlois@nce.ufrj.br
[2]*Universität Karlsruhe, Fakultät für Informatik,*
AlexPaar@ieee.org
[3]*University of California, Irvine, CA 92697*
nader@ece.uci.edu

Abstract

A novel method for improving the operation autonomy of the processing elements (PE) of SIMD-like machines is presented and discussed in this paper. The paper shows that it is feasible to avoid most branches and it is also possible to emulate nested if-then-else sentences on the processing elements by combining guarded instructions and pseudo branches. This prevents unnecessary intervention by the array control unit in many data-dependant computations, particularly those with short branch sections. The paper also shows that the simplicity of the method allows it to be implemented both in fine-grain and coarse-grain SIMD/ASIMD architectures because it does not require significant additional silicon area. Finally, it is shown that pseudo branches can be used to control the power saving of those processing elements that have instructions nullified.

1. Introduction

Advancements in ULSI technology have allowed some recent microprocessors to incorporate SIMD instructions in their execution units and it has been shown that significant improvements can be achieved. This has been driven by the need to accelerate multimedia and digital signal processing (DSP) applications because these applications are becoming increasingly important for computer systems as a dominant computing workload.

The Sun UltraSPARC processor enhanced with the "Visual Instruction Set" (VIS)[1], the "MultiMedia eXtentions"(MMX) and Streaming-SIMD Extensions(SSE) from Intel [2], the 3Dnow! Extension from AMD[3], and AltiVec technology from Motorola [4] are examples of SIMD signal processing instruction set extensions on general-purpose processors.

Dr. M. Anido acknowledges the financial support by CNPq. This work was supported by DARPA (DoD) under contract F-33615-97-C-1126 and the National Science Foundation under grant CCR-0083080.

There are also special-purpose multimedia processors (ASICs and DSPs) such as the Trimedia VLIW processor from Philips[5] and nVidia's nForce/geForce GPUs used in Microsoft's Xbox [6]. These processors usually have hardware assists in the form of peripherals for one or more of the multimedia decoding functions. A third category of architectures also used for DSP and multimedia applications are reconfigurable SIMD systems-on-chip, which can be advantageous in many applications, when compared with general purpose processors, ASICs or DSPs. Examples of such chips are MorphoTech's MorphoSys SIMD system-on-chip[7] ClearSpeed's FUZION architecture[8]. The market for such special-purpose multimedia processors is primarily in low-cost embedded applications such as set-top boxes, wireless terminals, digital TVs, and standalone entertainment devices like DVD players.

A recent paper [9] evaluates SIMD, VLIW and Superscalar Architectures for signal processing and multimedia applications. The paper shows that SIMD techniques provide a significant speedup for signal processing and multimedia applications. The observed speedups over a 3-way superscalar out of order execution machine ranges from 1.0 to 5.5. The paper also points out that out-of-order execution and branch prediction are observed to be extremely important for media applications.

2. Related Work on Previous SIMD Systems

An analysis and taxonomy of the different types of PE autonomy in SIMD architectures can be found in [19]. Other analysis of issues in the design of high performance SIMD architectures can also be found in [17,18]. Operation autonomy is one of the types of processor autonomy and is defined as the capability of locally selecting one of a few operations for execution, based on the local data.

The Massively Parallel Processor (MPP) [12] had a register (G register), which contained the internal status of the PE, and *masked* instructions. The content of the G register could

Proceedings of the 15 th Symposium on Integrated Circuits and Systems Design (SBCCI'02)
0-7695-1807-9/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

be changed, depending on the result of an operation and the control unit could specify for which values of the G register the instruction could be executed. *Masked* instructions would be executed by the PEs only if they had the G register equal to 1. Basically, it permitted to inhibit some PEs, depending on some previous result, which could be specified in several different ways. This was not conditional execution, which was introduced later by the Blitzen machine[13].

The Blitzen[13] architecture had two control registers targeted to provide more operation autonomy to each PE, namely the G ("mask") register and the K ("complement") register. The G register had a similar function as in the MPP machine and allowed to turn a PE off during a cycle[13], while the K (complement) register caused certain logical operations to be complemented. Some supported complement instructions were: Add/Sub instruction, load/don'load value to register, etc. These instructions were used to support some data-dependant operations in simple cases. It was a very limited form of PE operation autonomy because branch regions with several instructions in the *then()* and *else()* clauses or regions with complex nested branches could not be emulated. Additionally, preparing the mask in memory before loading it in parallel into the G registers can take a considerable number of cycles because the control unit has to write different data into different memory addresses corresponding to the memory bits of the processing elements (considering, for example, a 1024-bit mask).

Regarding operation autonomy, the MasPar MP-1[14] and MP-2[15,16] machines (also considered as ASIMD machines) incorporated some of the features of the Blitzen architecture, with functions similar to the ones provided by the G and K registers described above. Additionally, there is an OR tree that transmits the status of all PE internal buses (1 + 4 bits) to the control unit. One of the uses of this OR tree is the analysis of the result of some conditions such as Negative or Carry flags. If, for example, none of the carries in the PEs was set, then the OR tree result is zero and the control unit knows that all PEs go the same path in the global if-then-else clause and there is no need to issue instructions for one of the paths. This OR tree helps the control unit to take a fast decision, but it does not increase the operation autonomy of each PE.

In MP-1/MP-2, the control unit can activate subsets of PEs (changing active sets) that will enable or disable execution of subsequent instructions based on the outcome of the branch instruction. To accomplish this, the control unit writes into several memory positions that correspond to a 2D memory plane of the PE memory array. This operation takes many cycles. Once the mask is prepared in the memory plane, the control unit issues a *forall* type instruction that makes all PE's load the memory bit into their G register, enabling the activation/deactivation of individual PE's. Each PE evaluates the parallel condition locally, and sets its activity bit if the test passes. Next, the control unit broadcasts the instructions for the *then()* clause. The process is repeated for the *else()* clause. Thus, the execution time is the *sum* of the execution paths for each possible targets of the i*f-then-else* statement. Therefore, this approach has a considerable overhead, demanding the intervention of the control unit to prepare the mask in the PE's memory and broadcast the masks of all memory planes.

From the above discussion it can be seen that, to date, the PE's of SIMD/ASIMD machines have had very limited operation autonomy, namely: *execute or don't execute* one instruction (controlled by a local bit) and *execute an instruction or execute its complement* (also controlled by a local bit). When the *if-then-else* construct can not be resolved by one of these two cases, as usually occurs when the *then()* clause and the *else()* clause are more complex, the central processor has to intervene and the entire processing array will be stalled several cycles for each branch instruction.

MorphoSys [7,8] is a system-on-chip which consists of a reconfigurable ASIMD-like array of reconfigurable cells (RC) on the same chip with a general-purpose RISC processor and a high performance streaming memory interface. Its architecture is illustrated by Figure 1.

Besides allowing reconfigurable connections between RCs (PEs), MorphoSys has a characteristic of being able to execute different non-communicating instruction streams in different columns (or rows) under a central control. Such characteristic can be used to compute irregular data-parallel algorithms, facilitating the emulation of MIMD behavior. Mapping several algorithms on the MorphoSys architecture showed that very high performance can be achieved, when compared with other approaches, on various algorithms like

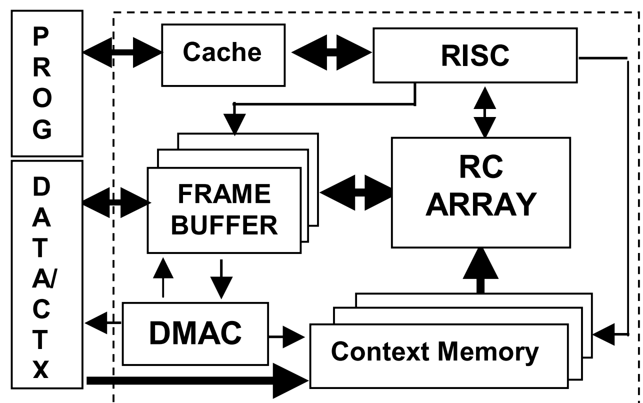


Figure 1 – MorphoSys SIMD System-on-Chip Architecture

IEEE COMPUTER SOCIETY

MPEG encoding, automatic target recognition, ciphering using the IDEA algorithm and several DSP algorithms[7,8]. Aiming at improving the operation autonomy of the reconfigurable cells(PEs), the approach described in this paper was implemented in the simulator of the new version of the MorphoSys reconfigurable SIMD architecture.

3. Using SIMD Guarded Instructions

Guarded execution means selectively executing instructions depending on the value of a condition register. Guarded execution can be performed in two ways: either having guarded instructions (there is a condition field on every guarded instruction) or by having a *Guard* instruction and a *Wakeup* instruction. An instruction which might be turned into a NOP depending on the value of the condition register is called a *Guarded Instruction* [20]. *Guard* and *Wakeup* instructions are used to nullify blocks of instructions. If the condition meets the requirement, then the instruction can be executed; if it does not, then the instruction is dynamically turned into a NOP. Since guarded execution can turn instructions into NOPs, normally there is no need for conditional branches. A more sophisticated form of guarded execution, is termed predicated execution, which can analyze more complex *if()* clauses and encode conditions stored in predicate registers. Predication, as used in recent general-purpose microprocessors, also refers to the fact that two (or more) instruction streams corresponding to the possible paths of *if-then-else* sentences can be dispatched and executed concurrently. Only the path whose predicate evaluates correctly will have its results committed. This is advantageous, when compared to branch prediction, because in branch prediction only one of the paths (*then( )* or *else( )*) is executed and if the prediction was incorrect the pipeline has to be flushed.

Predication has already demonstrated its potential to remove branch instructions from the program code of general-purpose microprocessors. We implemented a simplified form of predication for most of the SIMD instructions of the MorphoSys' architecture, which consists of *guarded execution* based on combinations of the ALU flags (Carry, Zero, Sign and Overflow). In the implementation described in this paper there were not enough bits in the instruction field to implement full predication which would include a field to specify a predicate register – instead, we opted for having more registers in the register file. Additionally, the implementation of a full predication scheme requires a considerable area, which may not be feasible for fine-grain SIMD/ASIMD architectures.

The code illustrated by Figure 2 exemplifies the potential of guarded SIMD instructions in terms of branch reduction, code size reduction and the associated operation autonomy that can be provided for the processing elements of SIMD architectures that incorporate this technique. The prefix of the SIMD instructions specifies the guard condition (GT = Greater than, MI = Minus, etc). The example of Figure 2 shows that by using guarded SIMD instructions it is possible to support simple parallel if-then-else constructs in the processing elements of SIMD architectures because all branches were removed. Guarded execution decreases the number of branch instructions in the code substantially resulting in a decrease in the number of control dependencies. Hence, the Instruction Level Parallelism is

```
HLL Pseudo Code                    RISC Assembly               SIMD Guarded Instructions
                                   Branch Delay =1
int ClipCode( x1,y1,z1,x2,y2,z2)
{                                  OPCODE   dst, sc1, sc2      Guard_OPCODE dst, sc1, sc2
   - - - -                         ======================      ==========================
10  c1 = 0;                        10  sub   r7, zero, z1      10     xor   c1, c1, c1
20  if (x1 < -z1){                 20  sub   r9, x1,  r7       20     sub   r7,  zero, z1
30     c1 = #Kl;                   30  brage 80                30     sub   r9,  x1,  r7
40  }                              40  xor   c1,  c1, c1       40  MI_moveq c1,  #Kl
50  else{                          50  moveq c1,  #Kl          50     sub   r9,  x1,  z1
60        if (x1 > z1){            60  bra   120               60  GT_moveq c1,  #Kr
70          c1 = #Kr;              70  sub   r9,  y1, r7       70     sub   r9,  y1, r7
80        }                        80  sub   r9,  x1, z1       80  MI_moveq r9,  #Kb
90      }                          90  brale 120               90  MI_add   c1,  r9, c1
100 if (y1 < -z1){                 100 sub   r9,  y1, r7       100    sub   r9,  y1, z1
110    temp = #Kb;                 110 moveq c1,  #Kr          110 GT_moveq r9,  #Kt
120    c1 = temp + c1;             120 brage 180               120 GT_add   c1,  r9, c1
130 }                              130 sub   r9,  y1, z1       130 ---- continues ----
140 else{                          140 moveq r9, #Kb
150       if (y1 > z1) {           150 add   c1,  r9, c1
160         temp = #Kt;            160 bra   210
170         c1 = temp + c1;        170 nop
180       }                        180 brale 210
190     }                          190 nop
200 /* Similar for x2,y2,z2 */     200 moveq r9, #Kt
210 ---- continues ---             210 add   c1, r9, c1
220                                220 ----- continues -----
```

Figure 2 – Mapping parallel if-then-else constructs in MorphoSys with no branches

Proceedings of the 15 th Symposium on Integrated Circuits and Systems Design (SBCCI'02)
0-7695-1807-9/02 $17.00 © 2002 **IEEE**

IEEE COMPUTER SOCIETY

increased. In addition to having guarded instructions to minimize the number of branches, there are also a few special instructions that help substituting branches, such as AddSub(flag) and IncDec(flag) instructions.

The objective of the analysis of Figure 2 is to show that *if-then-else* clauses (which translate into data-dependant branches) can be mapped in SIMD processing elements, once the proper hardware support is provided. The code reduction size by using guarded instructions was expected because this has already been demonstrated for general-purpose computers. Instructions that cannot be guarded because of limitations in the number of bits in the instruction code or when a complex *if-then-else* construct can not be implemented using guarded instructions, a *pseudo branch* instruction is employed. Pseudo branches are described in the next section and it will be shown that they can support complex nested *if-then-else* constructs. In the RISC encoding of Figure 2, some instructions are out of order to make use of the branch delay.

4. Supporting Nested If-Then-Else Sentences with *Pseudo Branches*

Although guarded instructions are essential to minimize the number of branches, they are not able to solve all possible situations. Additionally, there are instructions that do not have sufficient bits in the instruction word to specify the guard condition. In these cases, previous microprocessors have used *Guard* and *Wakeup* instructions to support *if-then-else* constructs. However, no previous scheme has supported *nested* if-then-else sentences. If PE's in a SIMD array had a program counter (allowing them to perform branches) there would be no problem. However, as it is not possible to have true branches, a new scheme to solve this problem had to be devised. We introduce *pseudo Branches* in SIMD machines, which are another form of a *Guard* instruction, but with the novel feature of adding a destination tag field in the instruction to support *nested if-then-else* constructs.

The main reason to emulate real branches in PE's is to be able to conditionally (or unconditionally) nullify blocks of code starting at the "branch" instruction and ending at the *target* position, independently of the instructions inside that block of code. This is how real branch instructions work and by doing so complex nested *if-then-else* constructs can be emulated. In normal microprocessors, the *target* address is loaded into the program counter (PC), before branching. One possible solution to specify a target instruction in a stream of SIMD instructions is to include an extra tag field in the instruction word. However, this increases the instruction word width, which is not feasible normally. To tackle these problems, the following innovations were included in MorphoSys' architecture and is illustrated by Figure 5.

- A *pseudo branch* instruction was created and it incorporates a destination (target) tag, to emulate branch behavior. The tag is included by the programmer as in normal assembly languages. It is not an address nor a relative position to the instruction. The assembler checks for duplicated tags and is also responsible for the automatic generation of a corresponding 5-bit code to each different tag.

- A tag field was inserted in the NOP instruction. Therefore, all *pseudo branch* targets must be NOP instructions. By using this approach, there was no need to increase the instruction word width.

- A 1-bit register (*Sleep/Awake* flag) was added to the

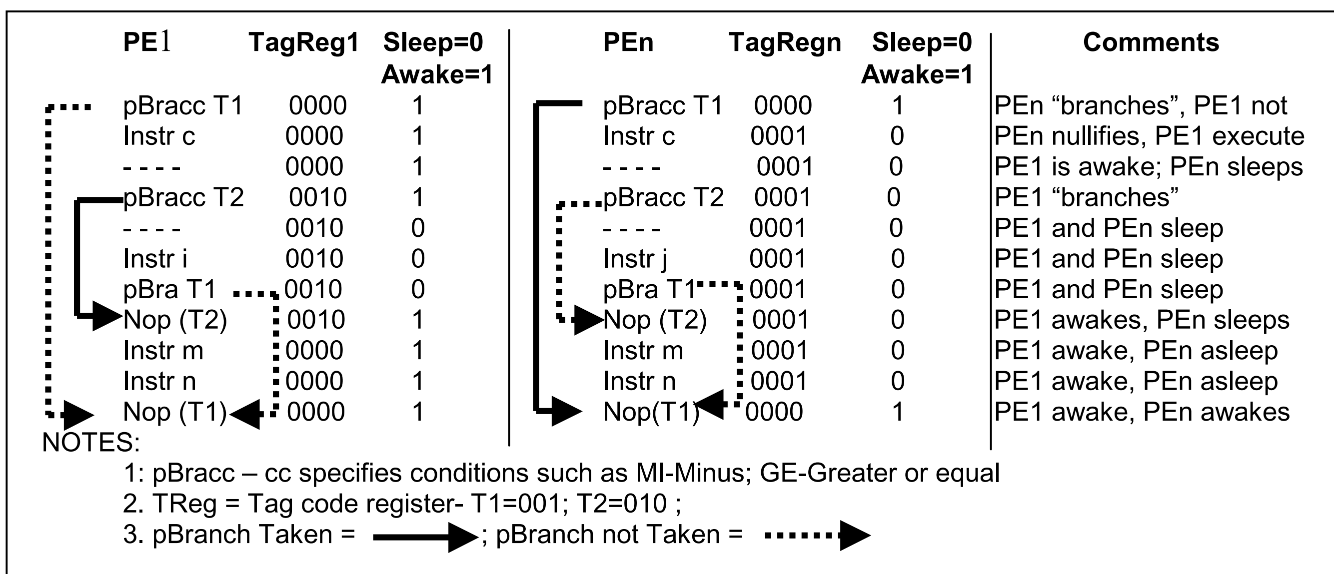


Figure 3 – Example of a data dependant program in two PE's using *pseudo branches* and sleep/awake control

0-7695-1807-9/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

architecture. The operation of this flag is as follows: normally this flag is in the awake (=1) state, which means that the PE is able to execute instructions. When a *pseudo branch* evaluates true (pbranch taken), then the branch tag field is copied to a *Tag Register* (TReg) and the Sleep/Awake flag is set to the Sleep (=0) state, which nullifies (inserts NOPs) all instructions until it is returned to the awake mode. This flag is returned to the awake (=1) state when the content of the tag register and the target tag existing in the NOP instruction are the same.

- A small (5-bit) register termed *Tag Register* (TReg) was added to the architecture. It copies the branch tag field when a *pseudo branch* is taken. Its usage is described above. The register size was limited to 5 bits because it was considered that having up to 31 different tags to support the encoding of nested *if-then-else* constructs was large enough (code 0 is reserved).

In [7,8] is described the internal architecture of a MorphoSys reconfigurable cell (or PE). The context register stores the SIMD instruction, which controls all other internal blocks. The two input multiplexers provide the support for the communication with neighbors and express lanes, as described in [7,8]. As illustrated by figure 5, the *pseudo branch* logic analyzes fields of the instruction word and generates the appropriate signals to control the sleep/awake flag.

The Sleep/Awake flag is also used for power saving in each PE. When in the sleep mode, all combinational blocks of the PE have their power disabled, except for the instruction decoder. The register file and the PE's internal RAM are kept powered because the programmer requires the use of pre-existing results. Having a NOP instruction being executed while the PE is being powered up again is essential because some time is needed between the wake up command and starting executing useful instructions again.

6. Evaluation and Implementation

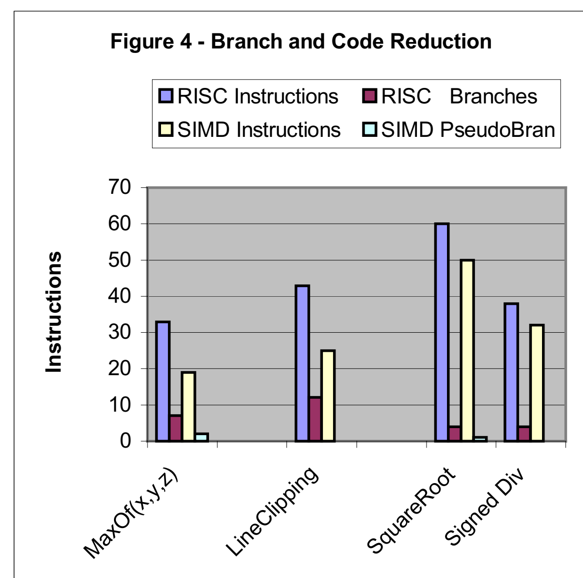
To analyze and evaluate the feasibility of implementing irregular data parallel applications, such as computer image generation (CIG), in the MorphoSys reconfigurable architecture, functions used in ray tracing and in projective approaches were mapped to the architecture and analyzed. We chose well-known functions with several nested *if-then-else* constructs, such as *MaxOf(x,y,z)* and *LineClipping(),* and two arithmetic functions – signed 32-bit by 16-bit division and 32-bit square root (which have less branches than the other two functions). In a parallel ray tracing algorithm, these functions are employed to allow several rays and objects to be processed concurrently in the MorphoSys architecture. In the projective method, several polygon edges can be clipped concurrently in each processing element.

The most important result is that it was possible to map, efficiently, these data-dependant functions using SIMD instruction streams without the intervention of the master processor. Figure 4 also shows that, compared with a general RISC with no guarded instructions, there was a significant reduction in code size (42%) and in the number of branches (90%). A comparison with a RISC microprocessor such as ARM[20], which has guarded instructions, would probably provide a similar result.

The capability of performing concurrent data-dependant tasks inside PEs increases instruction level parallelism and consequently increases efficiency. This is particularly important for the MorphoSys architecture, because it is possible to have up to eight different SIMD instruction streams running concurrently, which increases its capability of dealing with irregular data parallel computations.

Although the evaluation presented is not a complete benchmark, it demonstrates that, given the proper hardware support, it is possible to increase the operation autonomy of SIMD processing elements, allowing better support for irregular data parallel applications.

The implementation of the pseudo branch scheme is simple and does not require significant additional silicon area (Figure 5). Therefore, it can be used in fine-grain and coarse-grain SIMD architectures. The usual ALU flags are combined to generate normal guarded execution (or pseudo branch) conditions. The condition code field of the pseudo branch instruction (or the condition code field of a guarded instruction) selects the proper condition to be tested. In the case of a pseudo branch, if the test condition holds true, a



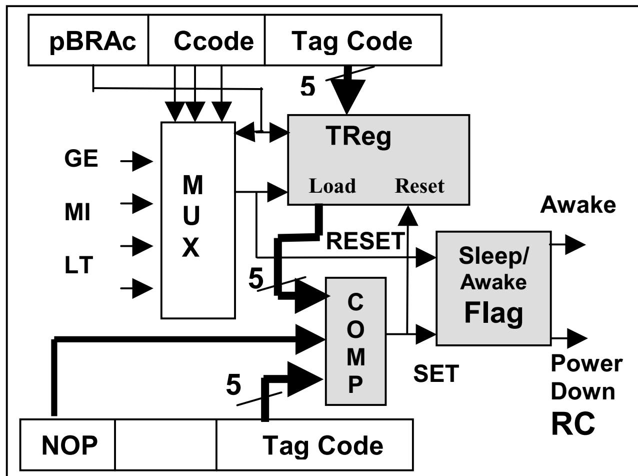


Figure 5 - Hardware Support for *Pseudo Branches*

reset signal commands the Sleep/Awake flag to go to the Sleep state, which also makes the PE to power down. This is illustrated by Figure 5. The Sleep/Awake flag goes to the awake state when a NOP instruction is being executed and the value of the tag register is equal to the tag code of the NOP instruction. A set signal commands the Sleep/Awake flag to go to 1, which powers up the processing element. The *set* signal also resets register TReg, indicating that the target tag was reached.

6. Conclusions

This paper presented and discussed a novel method for improving the operation autonomy of the processing elements of SIMD-like machines. To our knowledge, no previous SIMD/ASIMD architecture has used guarded and predicated execution schemes, such as described in this paper, nor any previous *Guard* instruction has used a tag field to support nested *if-then-else* constructs. The main contribution was to show that it is possible to, efficiently, map some data-dependant functions using SIMD instruction streams without the intervention of the master processor.

By combining guarded instructions with *pseudo branches* it was possible to achieve higher operation autonomy and higher instruction level parallelism than in previous SIMD/ASIMD architectures, providing better support for the execution of non-communicating instruction streams.

The paper also showed that the simplicity of the method allows it to be implemented in fine-grain and coarse-grain SIMD/ASIMD architectures because it does not require significant additional silicon area.. Finally, it was shown that the hardware support needed to implement *pseudo branches* can also be used for the power saving of those processing elements that have to nullify some of the instructions of their instruction stream.

7. References

[1] L. Kohn et al. "The Visual Instruction Set (VIS) in UltraSPARC", COMPCON Digest of Papers, Mar. 1995.
[2] B. Patwardhan, "Introduction to the Streaming SIMD Extensions in the Pentium III – Parts I, II and III", Dr. Dobb's Journal, January, 2002.
[3] M. Fomithchev, "AMD 3Dnow", Dr. Dobb's Journal, Vol. 25, No. 8, pp.40-42, August, 2000.
[4] Motorola, "AltiVec Technology", http://www.mot.com/SPS/PowerPC/AltiVec/index.html
[5] Philips, "Trimedia VLIW processos", http://www.semiconductors.philips.com/markets/industrial/trimedia/
[6] Nvidia, "Nvidia GeForce4 Graphics Processors Family", http://www.nvidia.com/
[7] H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, E. M. C. Filho, - "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications", IEEE Transactions on Computers, pp. 465-481, Vol. 49, No. 5, May, 2000.
[8] M.-H. Lee, H. Singh, G. Lu, F.J. Kurdahi, N. Bagherzadeh, E.M. Chaves Filho, and V.C. Alves, "Design and Implementation of the MorphoSys Reconfigurable Computing Processor", Jourrnal of VLSI Signal Processing , vol. 24, pp. 147-164, Mach 2000.
[9] D. Talla, L. K. John, V. Lapinskii, and B. L. Evans, "Evaluating Signal Processing and Multimedia Applications on SIMD, VLIW and Superscalar Architectures", Proc. IEEE Int. Conf. On Computer Design, pp. 163-172, September 2000.
[10] Thinking Machines Corporation, Connection Machine CM-5 Technical Summary. Technical report, Thinking Machines Corporation, Cambridge, MA, October, 1991.
[12] Batcher, K.E. – "Design of a Massively Parallel Processor", IEEE Transactions on Computers, pp. 837-840, September, 1980.
[13] Blevins, D.W., Davis, E.W., Heaton, R.A. and Reif, J.H.- "BLITZEN: A Highly Integrated Massively Parallel Machine", J. Parallel and Distributed Computing, Vol.8, No.2, pp. 150-160, Feb., 1990.
[14] Blank, T. – "The MasPar MP-1 Architecture", Proc. of the 35th IEEE Computer Society International Conference (COMPCON 90), San Francisco, California, pp. 20-24, February, 1990.
[15] Kim, W. and Tuck, R. – "MasPar MP-2 PE Chip : A totally Cool Hot Chip", Proc. IEEE 1993 Hot Chips Symposium, March, 1993.
[16] Nickolls, J. and Reush, J. – "Autonomous SIMD Flexibility in MP-1 and MP-2", SPAA '95 – 5th Annual ACM Symp. On Parallel Algorithms and Architectures, pp. 98-99, Velen, Germany, June, 1993.
[17] J. D. Allen, V. Garg and D. Schimmel, "Analysis of Control Parallelism in SIMD Instruction Streams," Proc. of the 5th IEEE Symposium on Parallel and Distributed Processors, pp. 383-390, 1993.
[18] Allen, J.D. and Schimmel, D.E. – "Issues in the Design of High Performance SIMD Architecrtures", IEEE Trans. On Parallel and Distributed System, Vol. 7, No. 8, pp. 818-829, Aug. 1996.
[19] P. J. Narayanan, "Processor Autonomy on SIMD Architectures", ICS-7, pp. 127-136, Tokyo, 1993.
[20] Advanced RISC Machines – ARM610 RISC processor, January, 1993.

0-7695-1807-9/02 $17.00 © 2002 **IEEE**
IEEE COMPUTER SOCIETY