MULTIMEDIA ENHANCED GENERAL-PURPOSE PROCESSORS

Stephan Wong, Sorin Cotofana, and Stamatis Vassiliadis

Electrical Engineering Dept., Delft,
Delft University of Technology,
The Netherlands
{Stephan, Sorin, Stamatis}@Cardit.ET.TUDelft.NL

ABSTRACT

This paper proposes a multimedia enhanced general-purpose processor (GPP) architecture. An M$^+$GPP enhanced architecture embodies a (regular) GPP architecture and a multimedia extension. The multimedia extension consists of a number of new multimedia dedicated instructions and (possibly new) hardware units to support their execution. Additionally, reconfigurable hardware units (RHUs) to support the execution of the new multimedia dedicated instructions are discussed. Finally, some implications of applying the M$^+$GPP concept to the superscalar and VLIW GPP architectures are presented.

I. INTRODUCTION

Due to recent developments, e.g., superscalar, ILP, and VLIW, general-purpose processors (GPPs) have become a good candidate for multimedia processing. Even though GPPs can perform multimedia operations, it can be noted, that these processors are not able to reach the performance level achieved by DSPs. This lower peak performance implies that usually GPPs are not able to process multimedia data in real-time and restricts the usage of GPPs in multimedia processing. However, GPPs are very flexible towards changes in multimedia operations and therefore used in multimedia applications in which performance is not a critical requirement. Recently, an attempt was made to bridge the performance gap by so-called *multimedia extended* GPPs which are regular processors with additional instruction sets, e.g., Intel x86 MMX [1], tuned to multimedia operations. While the performance of multimedia extended GPPs can be higher than that of regular GPPs when performing multimedia operations [2], still a sizeable performance gap remains to be bridged between these processors and the multimedia dedicated DSPs.

Recently, reconfigurable hardware co-existing with core processing has been proposed as a good candidate for speeding up processor performance, see for example [3][4][5][6]. Such an approach can be very promising however, as indicated in [3], the organization of such a hybrid processor can be viewed mostly as an open topic.

In this paper, we propose a general framework to improve GPPs multimedia performance that we call multimedia enhanced GPP (M$^+$GPP) architecture[1]. A M$^+$GPP architecture encompasses a (regular) GPP architecture and a multimedia extension. The multimedia extension consists of a number of new multimedia dedicated instructions and (possibly new) hardware reconfigurable units to support their execution. The main contributions in this paper can be summarized as follows:

- The M$^+$GPP concept and its associated trade-offs are introduced and the M$^+$GPP performance implications are discussed.
- Reconfigurable hardware units (RHUs) to support the execution of the new multimedia dedicated instructions are discussed.
- Some implications of applying the M$^+$GPP concept to the superscalar and VLIW GPP architectures are presented.

The paper is organized as follows: Section 2 introduces the M$^+$GPP concept. Section 3 discusses the usage of reconfigurable hardware units (RHUs) to support the multimedia dedicated instructions. Section 4 covers the implications of applying the M$^+$GPP concept to the superscalar and VLIW GPP architectures. Section 5 draws the conclusions of this paper.

II. THE M$^+$GPP CONCEPT

In order to enhance the multimedia performance of GPPs, existing GPP architectures need to be extended with new instructions tailored for a set of commonly

[1]The M$^+$GPP framework also covers the multimedia extended GPP case.

used algorithms found in multimedia applications. Due to the large variety of multimedia applications, not all multimedia applications can be supported by a certain M$^+$GPP. This problem can be alleviated if the new instructions are defined in such a way that one instruction can cover more than one basic operation. This can be achieved in two ways:

- The basic operations pool is partitioned into classes of similar operations and one instruction[2] is associated with each class. One of the input operands of this instruction specifies which basic operation in the class has to be executed.
- Basic operations can be collapsed into one more complex operation for which one instruction should be defined. This method can be applied to groups of basic operations identified to always appear in a certain order within the algorithms.

The hardware support for the new instructions is different for the *simple* and *complex* instructions. The simple instructions are very similar to existing (non-multimedia) instructions thus they can be supported by modifying existing hardware units. The complex instructions cannot be associated with any existing functional unit and thus have to be supported by completely new hardware units. The new hardware units can be implemented using regular chip implementation technologies or Field-Programmable Gate Arrays (FPGAs). The usage of FPGA technology is attractive as the support for all the new (complex) instructions can be implemented in one *reconfigurable* hardware unit (RHU). A potential drawback of this implementation method relates to the fact that due to the usage of FPGAs the implementation cannot always provide an optimal delay, and the reconfiguration of the hardware unit takes time. However, provided by the fact that new unit implementations can be downloaded into the RHU more programmability is added within the M$^+$GPP. In this way, even within the same M$^+$GPP generation, one can extend the architecture with a variety of instructions. As the decoding logic must be also extended to support the new instructions, it is a natural choice to implement (part of) the decoding logic in FPGAs too.

Another important architectural aspect of adding new instructions is the support of multimedia types, because they are usually smaller than the GPP registers width. One method is to re-use the architected registers present in the GPP architecture to store the multimedia data types (such as in the Intel x86 MMX case). This method requires special care to keep track which kind of data type is present in the registers. Another method is to add more (architected) storage capabilities to the M$^+$GPP architecture. In most cases, the last method is more efficient in handling the multimedia data types.

When adding the new instructions to any existing GPP architecture one should be aware of the following implications:

- The decoding hardware is becoming more complex as more instructions are added to the GPP architecture.
- Routing to and from the new hardware units can become more difficult.
- To fully utilize the new instructions and the programmability of the RHU, the multimedia application code needs to be changed manually or retargetable compilers should be used in conjunction with M$^+$GPPs.

Besides these implications, some other issues should be carefully considered when defining an M$^+$GPP architecture. For example, the opcode-space should be able to include all the new instructions. If this is not the case, then solutions to circumvent this problem may cause the performance to be less than the maximum achievable M$^+$GPP performance. Moreover, the memory bandwidth and its interfaces which were not designed for streaming multimedia data may also limit the performance increase.

III. RECONFIGURATION

As indicated in the previous section, FPGA implementation technology is probably the best option to implement the new hardware units. Due to chip area limitations, the capacity of the RHU is limited. Consequently, it cannot provide simultaneous support for the entire set of new instructions and thus unit reconfiguration is needed. Therefore, the most important factor limiting the performance of these units is the reconfiguration time.

Reconfiguration can be accomplished by many methods. One straightforward method is to reconfigure the unit at gate level which translates into long reconfiguration times, because specifying the unit at this level requires large configuration files which must be downloaded into the RHUs. A method to reduce the size of the configuration files is to only specify the interconnections between standard logic units present in the RHU. This method provides fast reconfiguration times, but flexibility of the unit is very much limited. To provide fast reconfiguration and yet maintain the

[2] We note, that in some circumstances more than one instruction may be needed to support the execution of the operations in one class.

flexibility of the FPGA approach, we propose a *coarse-grain reconfiguration* method. In this method, the notion of basic blocks is introduced. The basic blocks perform (basic) operations which can be combined to perform more complex operations. Now, to switch support for different complex instructions, reconfiguration of the RHU can be performed by reconfiguring the basic blocks and/or their interconnections.

Under these circumstances, three different reconfiguration scenarios can be imagined as follows (see Figure 1:

- Both the basic blocks and their interconnections can be reconfigured (Scenario 1).
- Some basic blocks are hardwired, some can be reconfigured. The same holds for the interconnections (Scenario 2).
- All the basic blocks are hardwired and just the interconnections are reconfigurable (Scenario 3).

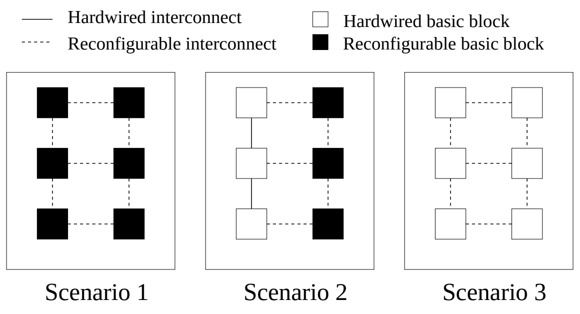

Figure 1: *Implementation Scenarios*

The highest flexibility is provided by scenario 1, but it also require the longest reconfiguration time. Scenario 3 is the least flexible, but provides the fastest reconfiguration time. Scenario 2 is situated between these two scenarios providing reasonable reconfiguration time and yet maintaining reasonable flexibility level.

As a final remark, we want to mention that additional logic is needed as the RHU needs to be reconfigured before instructions are issued to it.

IV. SUPERSCALAR & VLIW M$^+$GPPs

In this section, we attempt to analyze the implications of applying the M$^+$GPP concept to generic superscalar and VLIW architectures. For both cases, we assume that the basic operations were already identified and from these operations we can define the new multimedia instructions to be added to the existing instruction sets.

The superscalar M$^+$GPP architecture is depicted in Figure 2. The hardware support for new simple instructions can be provided by modifying some of the existing functional units. Support for the new complex instructions is provided by a reconfigurable hardware unit (RHU) which is appended as a regular functional unit. The RHU is using the coarse-grain reconfiguration method as discussed in Section 3. The RHU functions like the regular functional units, i.e., instructions are decoded and then issued to the RHU.

Adding the discussed support for the new instructions has the following implications:

- The decoding and issue logic needs to be changed in order to be able to decode the new instructions and issue them to the functional units or RHU.
- Due to the addition of the RHU, more read/write ports should be added to the register file.
- Possibly, the RHU should be directly connected to the main memory to better handle multimedia data.

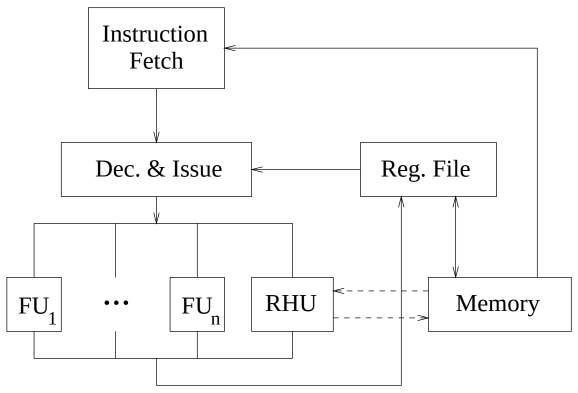

Figure 2: *Superscalar Multimedia Enhanced Architecture*

In an attempt to asses the performance implications of our framework, we carried out some simulation experiments for an out-of-order superscalar M$^+$GPP which includes multimedia instructions to support the DCT algorithms and the Huffman coding found in the encoding schemes of the JPEG [7] and MPEG [8] standards. Preliminary results suggests that the total number of execution cycles is decreased by a value between 12.40% and 23.72%. Moreover, the number of executed instruction is reduced by between 13.67% and 23.61% and the executed branches by between 9.83% and 15.98%.

In the VLIW case, we provide the same type of hardware support for the new instructions. Unlike the superscalar GPP which utilizes (complex) hardware scheduling techniques to issue instructions, the VLIW processors rely on compiler techniques to issue independent instructions to their own functional units. This is achieved by arranging the independent instructions into instruction slots within the (very long) instruction word. The instructions in separate instruction slots are then issued to separate functional units. Adding support for new multimedia instructions in VLIW architectures can be accomplished using one of two main methods. The first method encompasses an additional instruction slot which can contain only multimedia instructions. These instruction are then issued to an RHU. The main disadvantage of this method is that the entire VLIW architecture needs to be modified, because the length of the VLIW has changed.

The second method is to use existing instruction slots to accommodate the new multimedia instructions. In this case, after decoding an instruction in one of these instruction slots, the instruction is issued to either (modified) functional units or the RHU. Using this method, no new instruction slots are required, therefore no major architectural changes are needed. However, when the opcode of the new instructions does not permit easy decoding, the decoding logic complexity of this instruction slot is increased. This might penalize the performance of the whole VLIW processor.

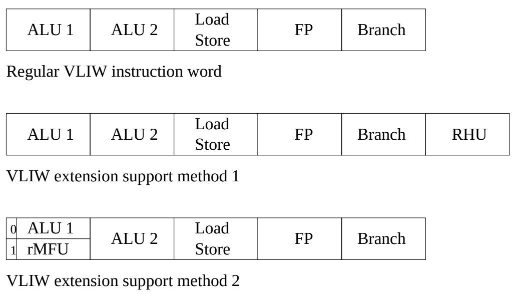

Figure 3: *VLIW Multimedia Enhanced Instruction*

An example of how the VLIW instruction format should be changed to support new multimedia instructions in both methods is depicted in Figure 3. In the figure, we assume for the second method that the first bit of the opcodes specify the target functional unit for the instruction.

Both methods may run into problems when the opcode-space cannot accommodate all the new multimedia instructions. This problem can be solved either by changing the architecture or by utilizing two slots for the encoding of one new multimedia instruction. In this ways, great parts of the opcode-space can be reused by sacrificing one instruction slot in the VLIW when multimedia instructions need to be executed.

V. CONCLUSIONS

In this paper, we discussed a general framework to improve GPPs multimedia performance, the multimedia enhanced GPP (M^+GPP) architecture. First, the M^+GPP concept and its associated trade-offs were introduced and the M^+GPP performance implications were discussed. Subsequently, reconfigurable hardware units (RHUs) to support the execution of the new multimedia dedicated instructions were discussed. Finally, some implications of applying the M^+GPP concept to the superscalar and VLIW GPP architectures were presented.

REFERENCES

[1] A. Peleg and U. Weiser, "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, vol. 16, pp. 42–50, August 1996.

[2] R. B. Lee, "Accelerating Multimedia with Enhanced Microprocessors," *IEEE Micro*, vol. 15, pp. 22–32, April 1995.

[3] J. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," in *Proceedings of the IEEE Symposium if Field-Programmable Custom Computing Machines*, pp. 24–33, April 1997.

[4] A. DeHon, "DPGA-coupled microprocessors: Commodity ICs for the early 21st century," in *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 31–39, April 1994.

[5] R. Razdan and M. D. Smith, "A High-Performance Microarchitecture with hardware-programmable Functional Units," in *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pp. 172–180, November 1994.

[6] R. D. Wittig and P. Chow, "OneChip: An FPGA Processor with Reconfigurable Logic," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 126–135, April 1996.

[7] G. K. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM*, vol. 34, pp. 30–44, April 1991.

[8] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, *MPEG Video Compression Standard.* No. ISBN 0-412-08771-5 in Digital Multimedia Standard Series, Chapman and Hall, 1996.