Low Power Coarse-Grained Reconfigurable Instruction Set Processor

Francisco Barat Murali Jayapala Tom Vander Aa Geert Deconinck Rudy Lauwereins
Henk Corporaal

***Abstract*— In this paper, we present a novel coarse-grained reconfigurable processor and study its power consumption. Preliminary results show that the presented coarse-grained processor can achieve on average 2.5x the performance of a RISC processor at an 18% overhead in energy consumption.**

I. Introduction

This paper presents CRISP, a coarse-grained reconfigurable instruction set processor designed for multimedia applications that can accelerate multimedia applications in a power efficient manner. The power of this architecture lies in the reconfigurable logic, which is composed of complex blocks such as ALUs or multipliers, that operate on the data sizes typically found in multimedia applications (8 to 32 bits), and is divided in independently enabled slices in order to reduce overall energy consumption and reconfiguration times. Also important is the tight coupling to the main microprocessor (the reconfigurable logic is seen as an extra functional unit) that allows quick control and data communication between the processor and the reconfigurable logic. Results on a set of multimedia applications show that the reconfigurable processor is able to achieve on average 2.5 times the performance of a RISC processor with just an average of 18% energy overhead.

II. A Low Power Reconfigurable Architecture

CRISP (Coarse-grained Reconfigurable Instruction Set Processor) is an instruction set processor composed of a main processor core tightly coupled to some coarse-grained reconfigurable logic. The coarse-grained reconfigurable logic is placed in a reconfigurable functional unit (RFU), and just like any other functional unit, an operation can be issued to it every clock cycle. The RFU reads/ writes data from/ to the main register file. The main processor can be any type of processor, though in this paper we will assume the processor is a simple RISC (Reduced Instruction Set Computer) processor.

Figure 1 presents the overall architecture of the complete processor. The main processor core reads its instructions from the level 1 instruction cache and obtains data via the level 1 data cache. Both caches are connected to a unified level 2 cache, which is in turn connected to an external memory. The reconfigurable fabric, in the center of the figure and directly controlled by the main processor core, contains configuration memory that is loaded via the unified level 2 cache (this allows reuse of configurations loaded from external memory and reduces reconfiguration times). The reconfigurable fabric can directly access the data cache via several data ports. Additionally, the reconfigurable logic communicates with the main processor core via a functional unit interface.

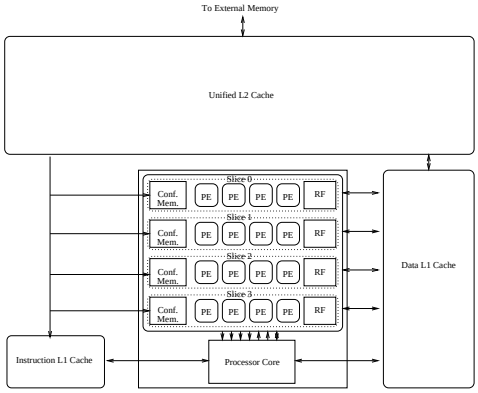

Fig. 1. Example CRISP instance (RFU: Reconfigurable Functional Unit, PE: Processing Element, FU: Functional Unit)

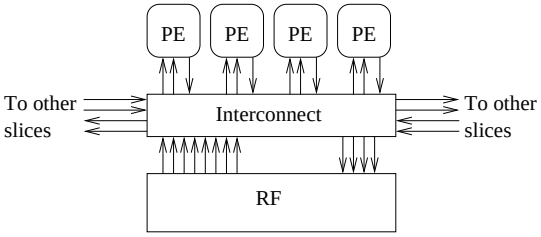

Fig. 2. Internals of a reconfigurable slice

As shown in figure 1, the reconfigurable functional unit is divided in reconfigurable slices, one of which is shown with more detail on figure 2. Each slice contains several coarse-grained processing elements (PEs), a register file, interconnect, and a small configuration memory. Each processing element is either an ALU, shifter, multiplier or memory unit. Such complex processing elements are better suited than the traditional logic blocks based on look up tables (LUTs) for the execution of the operations typically found in multimedia applications, which are word-oriented and not bit-oriented. These complex PEs allow the reconfigurable logic to operate at higher frequencies with lower power consumption when compared to traditional FPGAs.

F. Barat, M. Jayapala, T. Vander Aa and G. Deconinck are with ESAT/K.U.Leuven, Belgium. email: firstname.lastname@esat.kuleuven.ac.be
R. Lauwereins is with Imec, Belgium. email: lauwerei@imec.be
H. Corporaal is with T.U.Eindhoven, The Netherlands. email: h.corporaal@tue.nl

The processing elements inside a slice are connected together through programmable interconnect. This interconnect is a full crossbar that operates on words and has the same complexity as the bypass network typically found in current VLIW (Very Long Instruction Word) microprocessors. This crossbar can connect the output of any processing element to the input of any other processing element. It also connects the processing elements to the register file and to the other slices. In most cases, each processing element writes its output to the register file of the slice, but this behavior can be optionally bypassed, just like in traditional FPGAs, and the result routed to a different processing element. By combining this optional register write and the interconnect crossbar, it is possible to perform spatial computation such that elements in a data flow chain are connected together through the crossbar. The processing element at the end of the chain is connected to the register file.

Each reconfigurable slice also contains a configuration memory. This configuration memory stores the configuration for the slice's datapath components. Since the typical loop requires several configurations to be quickly alternated (as will be discussed in section III), the configuration memory must be multicontexted, (i.e. it must be able to store several configurations). Switching from one context (or configuration) to another takes just one clock cycle and is equivalent to reading from a shallow and wide memory. In the case of a slice with four processing elements, the width of this configuration memory is around 128 bits, much less than the bits required for a slice of a typical FPGA. The number of configurations in the configuration memory typically ranges between 8 and 32 contexts. Ideally, the number of contexts should be kept as small as possible to reduce the energy consumption of the configuration memory.

The reconfigurable functional unit is activated via a special reconfigurable instruction as shown in Figure 3. This reconfigurable instruction contains two main pieces of information. First, it contains a reconfigurable instruction identifier (RID) that specifies which of the many available configurations must be used. This identifier can select among a larger number of configurations than the number of contexts available in the configuration memory. If the required configuration is not currently loaded in the configuration memory, which behaves like a small cache indexed by this RID, the system is halted and the adequate configuration is loaded from the unified level 2 cache.

Aside from the RID, the reconfigurable instruction includes several fields of one bit length that specify which slices are going to be activated. Figure 3 shows these slice enable fields (named ENx in the figure). For those parts of the application that require a small number of processing elements, only the first slice will be activated. For those parts with higher parallelism requirements, more slices will be used. This mechanism, which can be considered as a form of partial reconfiguration, reduces the size of the configuration stream that must be loaded in the case of a configuration miss. Additionally, the slices of the reconfigurable datapath that are not required can be switched off, thus providing an effective form of energy consumption control.

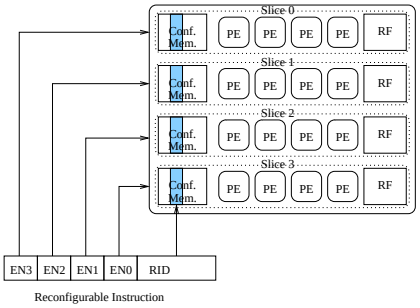

Fig. 3. Fields of a reconfigurable instruction. EN: slice enable bit, RID: reconfigurable instruction identifier

III. Compilation techniques

Code generation for any reconfigurable instruction set processor involves main two steps: synthesis of the different configurations for the reconfigurable array and generation of the code for the main processor (not mapped to the reconfigurable array). In the case of CRISP, with processing elements of complexity similar to standard functional units, existing VLIW techniques have been reused.

On our research compiler (based on Trimaran [1]), code generation for loops is based on software pipelining. In software pipelining, iterations are initiated at regular intervals and execute simultaneously but in different stages of the computation. This allows mapping the available parallelism onto the number of resources of CRISP. With this technique [2], the code generated for a loop will contain as many configurations as the initiation interval of the loop (the number of cycles of the loop kernel). It is therefore important to check that an iteration does not last more than the number of available contexts in the configuration cache. If this was not the case, the generated code would need constant reconfiguration and performance would fall down.

Software pipelining can also be modified to exploit the ability to perform spatial computation by chaining operations [2]. This allows a reduction of the critical path length of inner loops, with the corresponding decrease in execution time. The process of code generation with spatial computation requires a proper model of the timing delay of the processing elements and the interconnect, since the process is similar to the place and route stage in FPGAs.

Additionally, our compiler studies for each loop the required number of slices. Only the necessary number of slices are used, in order to reduce both reconfiguration times and energy consumption.

IV. Results

We evaluated a set of multimedia applications on a simulated processor. We have used Wattch [3] as a starting point for our power calculations. Figure 4 shows the normalized execution time of the set of benchmarks in several configurations. RISC represents the baseline RISC processor and the other entries represent the RISC processor with the number of reconfigurable slices ranging from 1 to 4. From this graph we can see that as the number of slices is increased, the execution time drops until the curve saturates. After this saturation point adding extra

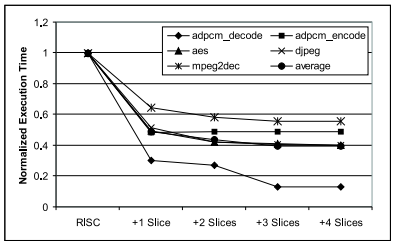

Fig. 4. Normalized execution time versus number of reconfigurable slices

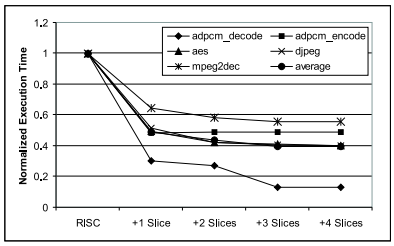

Fig. 5. Normalized execution time versus number of reconfigurable slices

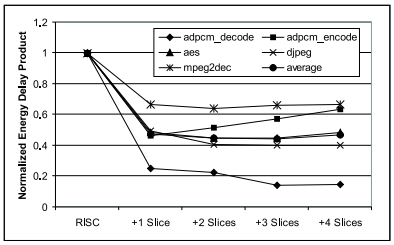

Fig. 6. Energy-delay product versus number of reconfigurable slices

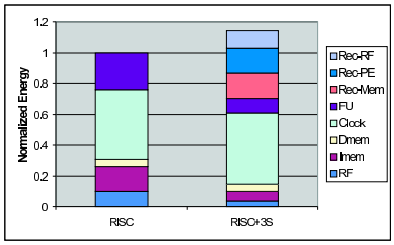

Fig. 7. normalized energy distribution for a RISC processor and a RISC processor with 3 slices for benchmark AES (right)

slices does not improve the performance. The saturation point depends on characteristics of the benchmark. ADPCM encode saturates with just one cluster, while others like ADPCM decode or mpeg2dec profit with 3 or more clusters. From this figure we see that the average performance increase is 2.5 and the maximum is 5.

Figure 4 shows the normalized energy consumption for the benchmark set. In general, the total energy consumption increases as more resources are added to the processor. The fact that the increase in energy consumption is not as fast as one might expect is derived from a better utilization of the processing elements of the processor as more slices are added. After the performance saturation point, the energy consumption is only increased by the extra load in the clock network (better clock gating would reduce this effect). The extra slices that are not using are basically shut off and hence their contribution to the energy consumption is negligible.

Figure 6 shows the normalized energy delay product (calculated as application execution time in cycles times the application energy consumption). It can be observed from there that all reconfigurable processors have a better energy delay product than the baseline RISC. The reason for this is that the reconfigurable processors are able to exploit the parallelism in the application reducing the number of instructions that need to executed.

Figure 7 shows the different components to the energy consumption of the RISC processor and a RISC processor with 3 slices for the benchmark AES. The benchmark AES runs at 2.5 the speed of the RISC processor and consumes only around 15% more energy consumption. We can see that a significant part of the energy consumption of the RISC processor is transfered to the reconfigurable components in the reconfigurable processor. Energy in the instruction memory decreases at an increase in the energy of the configuration memory. Energy from the functional units decreases and is transfered to the processing elements (resulting in almost the same energy consumption). In the case of the register file energy, the energy in all register files (processor core plus reconfigurable logic) is increased due to the usage of more power consuming units in the reconfigurable logic (multiported register files). Finally, the clock power is also increased, and from what can be seen from the figure, represents a significant amount of the energy consumption.

V. Conclusions

This paper has presented a coarse-grained architecture designed for low power multimedia applications. The reconfigurable logic is divided in slices that can be independently activated to reduce the power consumption of the processor. The processor achieves an average 2.5 performance increase over a standard RISC processor with just an 18% energy overhead. The reasons for the low power are the following; coarse-grained datapath elements, small and sliced configuration memory, sliced datapath and energy aware compiler.

Future work will study in more detail the power consumption of the processor. Also, we will study the effects of spatial computation on the performance and power consumption of the processor.

Acknowledgements

This work is in part supported by MESA under MEDEA+.

References

[1] "Trimaran: An infrastructure for instruction level parallelism." http://www.trimaran.org, 1999.

[2] F. Barat, M. Jayapala, P. O. de Beeck, and G. Deconinck, "Software pipelining for coarse-grained reconfigurable instruction set processors," in *Proc. ASP-DAC*, pp. 338–344, Jan. 2002.

[3] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proc. 27th Int'l Symp. Computer Architecture (ISCA 2000)*, pp. 83–94, June 2000.