On Transparently Exploiting Data-level Parallelism on Soft-processors

Bojan Mihajlović Željko Žilić Warren Gross
Integrated Microsystems Laboratory, McGill University
3480 University Street
Montreal, Quebec, Canada H3A 2A7
{bojan.mihajlovic, zeljko.zilic, warren.gross}@mcgill.ca

ABSTRACT

This paper considers the acceleration of computationally intensive algorithms on FPGAs. When speedups up to 16x are required over the performance of software on a soft-processor, the data-level parallelism of algorithms can be exploited using SIMD extensions to an ISA. We evaluate how a toolchain can be created to suit a parameterizable soft-processor, in order to automate the process of extracting parallelism. Using our technique, developer time and effort can be saved over the traditional method of offloading tasks to specialized hardware units.

1. INTRODUCTION

Field programmable gate arrays (FPGAs) can be used to accelerate computationally intensive algorithms. For low-volume production, an FPGA can yield a speedup for a fraction of the price of an application-specific integrated circuit (ASIC), with an upgrade path that may not be available using a microprocessor or digital signal processor (DSP). Future application changes can be seen as important in the domain of DSP applications, which see fast development cycles along with products shipping based upon unfinished standards (such as IEEE 802.11n). Traditional product obsolescence places demands on energy, time, and money when they are to be replaced, while planning for future changes can make devices resistant or immune to obsolescence.

When system flexibility is important, implementation on an FPGA removes many impediments to future system changes, fixes, and re-purposing. Traditional methods of accelerating algorithms on an FPGA involve the offloading of computationally intensive software inner loops to hardware function units. This method can be time and effort intensive when massive speedups are not required (over 16x, in this case). In recent years the emergence of a plethora of C-to-hardware languages [2] has offered an alternative to the traditional acceleration approach. However, the majority of these languages suffer from drawbacks, including that they are proprietary high-level languages, programmers are not proficient in their use, there are few support libraries, and that design control must be ceded to their automated tools. In addition, a small application change requires a rerun of synthesis, place, and route of the entire system.

We suggest exploiting the data-level parallelism of software executing on a soft-processor instead. If some single-instruction multiple-data (SIMD) instructions can be used in lieu of single-instruction single-data (SISD), a soft-processor can achieve acceptable speedup without the overhead of custom hardware development. Prior works have made the case for using both vector and SIMD soft-processors [5, 1], having focused on their hardware architectures. Here we consider the software toolchain needed to exploit data-level parallelism (DLP) automatically. If a compiler can recognize DLP and generate SIMD instructions from regular high-level code, software developers need not be concerned with the underlying hardware. Without such a compiler, hardware design effort may be replaced by software design effort, as developers are forced both to learn the instruction set and to program in low-level languages or constructs.

Since one of the benefits of soft-processors is that their parameters can be changed to suit a system requirement, such as performance or area footprint, we are also concerned with toolchain support for the multiple configurations of a parameterizable soft-processor.

2. SIMD APPROACH

The use of a soft-processor instruction set architecture (ISA) with SIMD instructions can benefit applications where DLP is unexploited. Such applications include those used in DSP which are composed of purely SISD instructions.

In selecting an instruction set for a SIMD unit, the soft-processor application suite must be examined for common operations. The SIMD extensions of modern ISAs can be used as a model, such as VMX or SSE. These allow at most a 16x speedup, depending on the units of data being processed and the amount of data realignment needed between SIMD registers and memory. A small set of instructions can be created to target area footprint, or a larger set to ensure future flexibility within a family of algorithms or application area.

Modern compilers allow DLP to be exploited without any application code changes, generating SIMD instructions with a technique known as auto-vectorization. We explore the feasibility of applying this to a mixed SISD/SIMD soft-processor.

3. FEASIBILITY OF IMPLEMENTATION

3.1 Toolchain

We use the GNU toolchain, due to its flexibility and ubiquity, to evaluate exploiting DLP automatically. A functional GNU toolchain can be created with the GNU Binutils suite of assembler tools, the GNU Debugger (GDB), and the GNU Compiler Collection (GCC). Binutils includes an assembler and linker, among other tools. The suite also contains CGEN (CPU Tools Generator), which can generate

Binutils for a new target using a single-file CPU definition written in a programming language similar to Lisp.

3.1.1 Assembly and Debug Tools

A custom soft-processor target would typically use the definition file to describe the instruction format, register files, and instruction functionality. The use of CGEN also presents new flexibility advantages in soft-processors. Conditional compilation of the definition file can be used to support multiple hardware configurations. This would include a variable-length register file and any subset of instructions, perhaps suited to a target application or area goal.

While the current version of CGEN supplied with Binutils does not offer native support for the wide datatypes needed to operate on SIMD registers, we find that its source code can be modified to support at least 128-bit datatypes. The VMX set of SIMD extensions use this same register width and allow operations to be simultaneously performed on (4) 32-bit, (8) 16-bit, or (16) 8-bit operands packed into a single SIMD register. The datatypes are used directly in the Lisp-like language to describe instruction functionality to GDB. We also find that GDB and its simulator can similarly be modified to support the wider logical and memory operations.

3.1.2 Compiler

An important compiler consideration is the ease with which it adapts to changing soft-processor configurations. In GCC, the modular nature of its *machine description* file allows a large instruction set to be described as subsets of instructions. As an example, subsets can be divided in terms of the vectors they operate on (32-bit, 16-bit, or 8-bit), or into application-specific instruction sets. Passing command-line switches to the compiler allows instruction subsets to be enabled at application compile-time. This allows the compiler to adapt to any number of instruction set configurations. However, adapting to a variable-length register file would require a compiler rebuild, since register allocator rules are located in a *target machine description* C-language header file. Multiple configurations can be supported with the use of conditional compilation.

Recent versions of the GNU Compiler Collection (GCC) recognize exploitable data parallelism and are able to map operations to SIMD instructions. The compiler features an auto-vectorizer [3] which can generate SIMD instructions from regular high-level code. Recent improvements to GCC include added hooks to the *operational tables* from user-defined patterns, allowing more direct mapping to the operations commonly performed with SIMD instructions. Unfortunately, the hooks offer limited support for auto-vectorizing custom instructions. These instructions would need to be implemented using manual efforts, such as the GCC functions known as *builtin*s, which can be mapped directly to instructions. While auto-vectorizers are generally not as efficient as manual efforts to exploit DLP, they come at minimal cost to the programmer and are likely to become more efficient in the future. This is demonstrated by recent work that has enabled vectorizing outer-loops of functions [3].

3.2 Area and Complexity

The overhead of SIMD instructions on a SISD processor can be small, since much of the hardware remains the same. A register bank can be added to support the increased data widths, but in some implementations the processor's original registers are reused under SIMD aliases. Additional control logic should be added to the load/store unit in order to support SIMD opcodes and registers, but the original instruction width can be retained. Instruction and data caches can be reused, although data cache should be reorganized into lines equal to the SIMD register width. Finally, SIMD function units must also be added, whose overhead will vary.

3.3 Memory Bandwidth

A SIMD function unit that is consuming more data per cycle than its SISD counterpart will need more memory bandwidth. Off-chip memory can generally keep up with the demands of soft-processors, since the clock frequencies achieved on FPGAs are many times lower than those achievable on ASICs.

3.4 Data Misalignment

A classical problem with the use of SIMD instructions is the inefficiency of misaligned memory accesses. This occurs when the memory location being read/written is not a multiple of the SIMD register width. This problem can be mitigated with the use of *permutation* instructions and optimization techniques [4]. Modern compilers such as GCC include support for the automated application of these techniques, and while inefficiencies exist in exploiting DLP, considerable speedups are still attainable.

4. CONCLUSION

We have presented an argument for the use of SIMD extensions on soft-processors when performance increases are sought. Exploiting data-level parallelism in software has advantages in FPGA area footprint, design time, and scalability over a traditional acceleration approach. We find that it is viable to create a toolchain that will support the many hardware configurations of a parameterizable soft-processor. Modifications to a standard toolchain would allow SIMD instructions to be supported and generated from regular high-level code. In this way, the cost of both initial application deployment and subsequent changes is kept low due to the absence of custom hardware design effort.

5. REFERENCES

[1] J. Cho, H. Chang, and W. Sung. An FPGA based SIMD processor with a vector memory unit. *Circuits and Systems, IEEE International Symposium on.*, 2006.

[2] E. El-Araby, M. Taher, M. Abouellail, T. El-Ghazawi, and G. Newby. Comparative Analysis of High Level Programming for Reconfigurable Computers: Methodology and Empirical Study. *Programmable Logic, 3rd Southern Conference on.*, 2007.

[3] D. Nuzman and A. Zaks. Outer-Loop Vectorization - Revisited for Short SIMD Architectures. *accepted for publication in Parallel Architectures and Compilation Techniques, International Conference on.*, 2008.

[4] G. Ren, P. Wu, and D. Padua. Optimizing data permutations for SIMD devices. *Programming Language Design and Implementation, Proceeding of the Conference on.*, 2006.

[5] J. Yu, G. Lemieux, and C. Eagleston. Vector Processing as a Soft-core CPU Accelerator. *Field Programmable Gate Arrays, International Symposium on.*, 2008.