

ρ -VEX: A PRELIMINARY PERFORMANCE AND CONFIGURATION ANALYSIS

Thijs van As, Dr. Stephan Wong (*Supervisor*), Prof. Geoffrey Brown (*Advisor*)
t.vanas@gmail.com

Computer Engineering Laboratory,
Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology, The Netherlands

ABSTRACT

Increasingly more computing power is demanded for multimedia applications. This paper presents a design space exploration and a justification for an embedded VLIW co-processor, targeted at the Molen reconfigurable computing platform: ρ -VEX. Our results show both a significant speedup in the execution time for different application domains and a proposal for a particular implementation of ρ -VEX. The performance speedup is achieved while having the advantage of a fixed footprint area-wise.

1 Introduction

The demand for computing power in consumer electronics is increasing at a very high rate. In the age of the Internet, multimedia and 3D visualizations, people need high-performance computers in order to cope with all the latest applications. We think that solutions based mainly on general-purpose processing power, do not provide the most optimal (resource- and power-efficient) systems. We believe that a combination of general-purpose and application-specific processing power is able to provide more efficient solutions for this increasing demand.

This paper presents the results of a preliminary analysis of the ρ -VEX [12] project, entailing the design and implementation of a VEX-based embedded Very Long Instruction Word (VLIW) co-processor core that is aimed to operate within the Molen reconfigurable computing paradigm [8, 14, 13].

The Molen polymorphic processor provides the possibility of executing an application-specific core in a custom generated hardware unit, which resides inside a reconfigurable fabric. The general-purpose processor within the Molen machine takes care of general-purpose calculations, parallel to the application-specific calculations by the custom unit. Overall application speedups of more than 3 times have been achieved on different state-of-the-art multimedia applications [7].

VLIW processors are efficient machines for calculations that contain a lot of Instruction Level Parallelism (ILP) that can be exposed by a good compiler. Applications in the multimedia domain happen to contain a lot of

ILP, because they typically consist of many independent repetitive calculations.

By means of embedding a VLIW co-processor inside the reconfigurable fabric of a Molen machine, we aim to bridge the gap between the execution time of an application-specific kernel on the general-purpose processor and a custom generated hardware unit. This would result in a compromise between two fields. The first is the execution time of the application-specific kernel, and the second is the on-chip area used for the hardware. One VLIW co-processor is able to perform a large number of different calculations within a fixed area footprint, whilst a custom hardware unit is only able to calculate (probably) one calculation on a fixed area footprint.

The results of our research propose an implementation space and a justification of the existence by means of expected computing performance for such a VLIW co-processor, hereafter called ρ -VEX.

The remainder of this paper is organized as follows. Section 2 presents a general introduction to the VEX VLIW architecture and the Molen reconfigurable computing paradigm. Subsequently, Section 3 discusses significant VEX machine configurations. Section 4 presents the performance analysis done to justify the existence of a ρ -VEX processor. A proposal for the implementation space is presented in Section 5. Finally, Section 6 presents the conclusions of our research.

2 Background

Our work is mainly based on two projects in computer architectural design, namely the VEX VLIW processor, and the Molen reconfigurable processor. This Section first provides a background on the VEX architecture, and then on the Molen reconfigurable computing paradigm.

2.1 The VEX VLIW Architecture

The VEX [2] (VLIW Example) ISA is loosely modeled on the ISA of the HP/ST Lx [1] family of VLIW embedded cores. The VEX ISA offers a scalable technology platform for embedded VLIW processors, that allows variation in

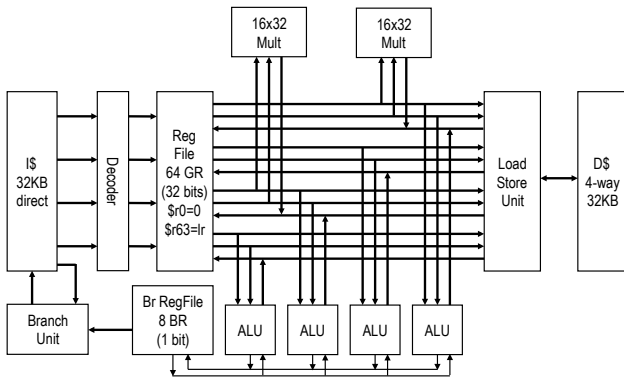


Figure 1. Structure of the default VEX cluster [2]

many aspects, including instruction issue width, organization of functional units, and instruction set. The VEX ISA supports a multi-cluster implementation, where each cluster provides a separate (possibly different) VEX ISA implementation. Each cluster has the ability to issue multiple operations in the same instruction. A VEX multi-cluster processor shares one instruction fetch unit and one memory controller. The customizability of the instruction set enables the definitions of special-purpose instructions in an organized way. Figure 1 presents the structure of a default VEX cluster, with an instruction issue width of 4.

A VEX Toolchain [4] is provided by Hewlett-Packard Laboratories, which offers a VEX C compiler and a VEX simulator. The compiler allows the user/designer to adjust parameters of the VEX processor (like the number of clusters and the issue width) easily. The VEX simulator offers an architecture-level simulator which comes with a set of POSIX-like libraries, a cache simulator and an API. The simulator is able to output many statistical run-time data of simulated applications.

The VEX C compiler is a derivation of the Lx/ST200 C compiler, itself a descendant of the Multiflow C compiler. It uses trace scheduling as its main scheduling method. Trace scheduling implies that operations will be restructured so that large ‘traces’ appear without branches. Profiling of compiled applications is supported via the GNU Profiler *gprof*.

The VEX simulator is a ‘compiled simulator’, contrary to an ‘interpreted simulator’. An interpreted simulator is the straightforward, but slowest, way of simulating an architecture. The interpreter mimics the target processor, and does everything the target processor does on the instructions: fetching, decoding, executing... Because of the interpretation overhead this is a slow solution (but relatively easy to implement). A compiled simulator translates the target executable binary code to a binary executable that can run on the host system. This removes a lot of the interpretation overhead in an interpreted simulator, and is thus a lot faster.

As of the start of our research, there is no hardware implementation of the VEX ISA known to the authors. Nei-

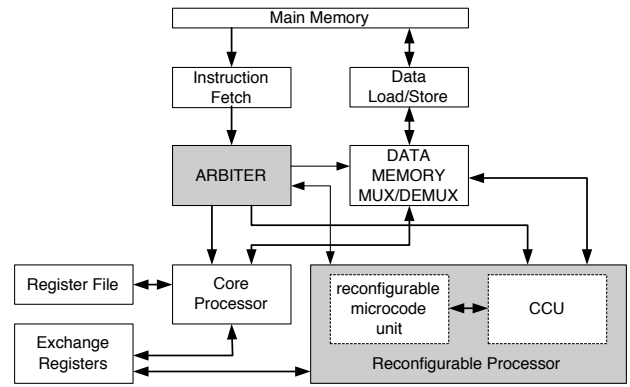


Figure 2. The Molen machine organization [8]

ther are implementations of a VLIW processor in FPGA fabric known. Implementations like Spyder [5] and [6], a NIOS II based implementation, do not offer the parameterizable VLIW clusters as the ρ -VEX is aimed to provide.

The choice for the VEX ISA was made because of the quality of the available toolchain (the Multiflow and Lx compilers are highly developed, and were/are commercially used), and the highly configurable ISA. These aspects allow us to suggest a design which both has a robust design, and achieves high performance.

2.2 The Molen Reconfigurable Computing Paradigm

The Molen paradigm provides a solution to the growing processor hardware design challenges, by reconfigurable processors (processors that adapt their micro-architecture according to the application’s requirements). This is being achieved by Custom Configured hardware Units (CCUs) and reconfigurable microcode ($\rho\mu$ -code) [13].

Figure 2 presents an overview of the Molen machine organization, including a General-Purpose Processor (GPP) with a fixed instruction set, and a reconfigurable co-processor. Application code is executed by default on the GPP, except for selected functions, which were proven to have a very efficient hardware implementation. Those are executed by the reconfigurable processor, by a CCU. A $\rho\mu$ -unit controls the control flow inside the reconfigurable processor.

The process [14] of application generation and execution for a Molen machine is as follows:

1. From given a application source code, (a) piece(s) should be determined and isolated for execution on a hardware CCU. This is being achieved by high-level to high-level instrumentation and benchmarking. This results in a set of candidate code pieces.
2. From these code pieces, it has to be determined which pieces are suitable for hardware execution, by means of the effort it takes to map them onto hardware.

3. A hardware implementation in an HDL should be generated either automatically, or manually (for timing-critical parts).
4. Calls to these particular hardware mapped functions should be replaced in the software code by calls to the hardware unit. Data exchange between the hardware units and the general-purpose processor is established by the means of exchange registers (XREGS).
5. Upon execution of the application, the reconfigurable fabric should be (re-)configured with the corresponding functions in time.

Design and implementation of (new) concepts for the Molen paradigm are very active topics at the Computer Engineering Laboratory of Delft University of Technology, ever since its conceptual introduction in 2001 [13].

A prototype [7] of the Molen processor is currently available, based on the Xilinx Virtex-II Pro [15] platform. This FPGA platform features two PowerPC 405 [16] general-purpose CPU cores embedded in the reconfigurable fabric. One processor 405 core running at a clock speed of 300 MHz serves as the general-purpose processor in the current prototype. This prototype will be the platform at which the ρ -VEX co-processor will be targeted. The ρ -VEX will be implemented as a CCU for the Molen processor, so as a part of the reconfigurable processor. The application code which is targeted for execution by the ρ -VEX will be directed to the VEX compiler, where binary executable code will be generated.

3 Scalability

The VEX architecture is designed to be (very) customizable. This Section deals with different VEX machine and cache configurations, as well a proposal for the ρ -VEX processor.

3.1 VEX Machine Configurations

As described in [2], a VEX machine is a highly customizable. Figure 3 depicts the structure of a VEX multi-cluster implementation. A well-defined VEX machine should have at least one cluster (cluster 0). For each cluster, the number of resources per instruction (ALUs, multipliers, issue width and memory ports) can be indicated, as well as the delay per computational element and the number of registers. Figure 1 depicts the default structure of a VEX cluster. We can distinguish some basic VEX machine variations:

- **1-cluster VEX** A 1-cluster configuration is a default configuration. The machine has one VEX cluster (cluster 0), for which the various parameters can be altered.
- **multi-cluster VEX** On a multi-cluster machine, cluster configurations can be altered per cluster. This

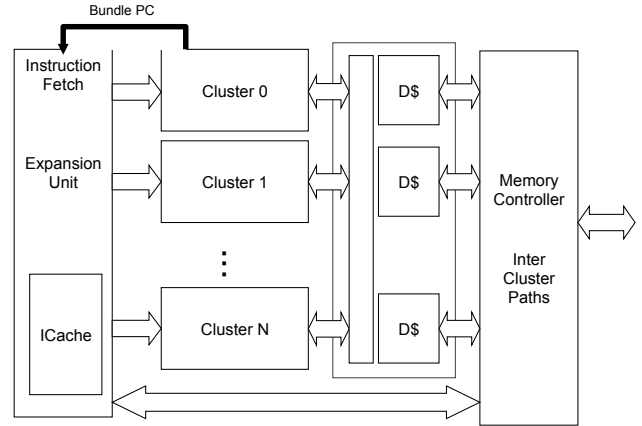


Figure 3. Structure of a VEX multi-cluster implementation [2]

means that cluster 2 and cluster 3 can both have a different ISA.

- **RISC VEX** This machine variation is actually a subset of the 1-cluster VEX machine models. The issue width is 1 by default, so only 1 ALU is available, 1 multiplier, etc.

3.2 Cache Configuration

As well as per-cluster configurations, global configurations like cache sizes and miss penalties can be defined. In the case of a simulation, these configurations are passed as directives to the simulator. Instruction- and data-cache sizes are of significant importance for the performance of a processor. Cache misses can heavily influence the number of executed, by introducing stall cycles. The following equations from [3] show us what the influences of cache misses are (IC is the Instruction Count):

$$\text{CPU time} = (\text{CPU cycles} + \text{Mem stall cycles}) \times \text{Cycle time} \quad (1)$$

$$\text{Mem stall cycles} = \text{IC} \times \frac{\text{Mem accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty} \quad (2)$$

It is clear that for applications with many memory accesses, or with badly constructed cache mechanisms (resulting in a high miss rate), the memory stall cycles can account for a significant part of the total number of execution cycles.

As for our research on the ρ -VEX processor, we suggest three configurations for which simulations should be done, in order to be able to define a good design space:

- **Configuration with default cache sizes** A default VEX machine has both an instruction- and data-cache of 32 KB. The cache mechanism is organized 4-way set associative.

- **Configuration without cache** To gain insight in VEX performances where no cache is available, simulations should be done on VEX machines where no cache is available. Currently, in the 3.41 version of the VEX toolchain, we could not de-activate the cache system totally for simulation purposes. We were able to bring it down to a 16 byte, direct mapped cache.
- **Configuration with cache trade-off** Because memories are expensive to implement in reconfigurable fabric, we wanted to propose a design which uses less cache memory than the default 32 KB. A 4 KB instruction- and data-cache seemed to be a good trade-off, performance-wise.

4 Performance Analysis

To explore the performance of the VEX ISA, we investigated results of earlier research, and we performed complementary measurements. We stated earlier that the VEX ISA a descendant is of the Lx ISA by HP/ST. Both architectures show many similarities. In earlier work, benchmarks have been performed on the Lx architecture. The results of these benchmarked are evaluated here.

We performed new simulations on the VEX simulator [4], with custom software kernels. The results of these benchmarks are presented and evaluated here.

4.1 Lx Analysis

Because the VEX ISA is very familiar to the Lx ISA by Hewlett-Packard and STMicroelectronics, we considered the performance measurements of Lx based processors with different ISA configurations of representable value.

In [1], performance measurements have been done on different Lx machine configurations against a cycle-accurate simulator. The benchmark set was a subset of the SPECINT'95 suite, consisting of application-specific benchmarks, as well as general-purpose benchmarks. Most of the application-specific benchmarks were optimized with source-level compiler pragmas, as well as code restructuring to expose more ILP.

Figure 4 depicts performance charts for an Lx processor running on a clock frequency of 300 MHz¹. The cluster-width of the Lx processor varies between 1,2 and 4 clusters (corresponding to an issue-width of 4, 8 or 16 operations). Performances are compared to an Intel Pentium II running at 333 Mhz, with a performance scale of 1.00.

These results clearly show that specializing for an application domain pays off in terms of performance gain, in contrast to general-purpose applications. This is expected behaviour, as general-purpose applications are not able to expose a lot of ILP. The general-purpose results are of no particular interest for our research, because the

¹In [1], performance of 200 and 400 MHz Lx processors was also measured, and all results were compared to a 275 MHz StrongARM processor. These have been left out because of no significant importance.

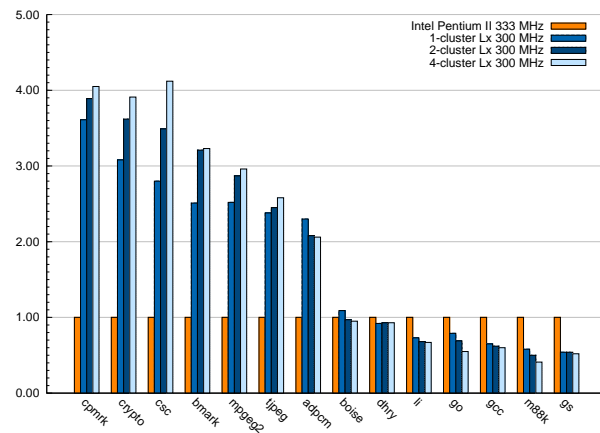


Figure 4. Lx performance chart, varying cluster width from 1 to 4 [1]

ρ -VEX will become an embedded co-processor within a Molen machine, dedicated to execute application-specific kernels. The general-purpose code will be executed by the general-purpose processor inside the Molen machine.

Except for the ‘adpcm’ benchmark, upscaling the issue width of the Lx processor improves the performance. This heavily depends per application. Both the largest and the smallest performance gain can be found in the ‘bmark’ benchmark: going from 1 to 2 clusters (issue width increases from 4 to 8) yields a performance gain of about 28%. Contrary, the gain from 2 clusters to 4 clusters can be neglected. Generally speaking, it can be concluded that the increase from 1 to 2 clusters has a larger influence on the achieved performance, than scaling from 2 to 4 clusters.

4.2 VEX Analysis

To gain more insight in the performance of ρ -VEX processor we performed benchmarks on the VEX ISA. To investigate the performance and real-life pay-off of different VEX configurations we performed the benchmarks on different machines:

- **RISC-VEX machine** A VEX machine in RISC configuration equals a VEX machine with an instruction issue width of 1. By default, a VEX cluster has an issue width of 4. We performed the benchmarks on a RISC-VEX configuration, because this would show clearly the advantage of a higher issue width when the results are compared to a 1- or 2-cluster VEX machine.
- **1-cluster VEX machine** The single cluster in this machine has a default configuration concerning computation resources.
- **2-cluster VEX machine** Both clusters of this machine have default resources configured. Because our origi-

	G.723			Matrix Multiply			Min/Max from Array			Moving Filter		
	DEF	NO	PROP	DEF	NO	PROP	DEF	NO	PROP	DEF	NO	PROP
VEX RISC	2108515	10782609	3363355	2320266	24803442	12337487	1017	3789	1332	1944	14458	2349
VEX-1	1148287	9871235	2442257	1277970	22492006	11431185	940	3878	1255	1517	14468	1922
VEX-2	1118928	10314585	2244582	1023833	22201002	11413672	976	3797	1291	1347	12929	1783
PPC-EDK	17213803			27015717			2170			6942		
PPC-Molen	N/A			N/A			2754			7530		
CCU	N/A			N/A			360			558		

Table 1. Number of clock cycles per system in the VEX benchmark set

nal intends are to design a 1- or 2-cluster ρ -VEX processor, this machine is included in the set.

- **PowerPC 405 at 300 MHz** This PowerPC is on of the direct ‘competitors’ of the ρ -VEX, because it is the GPP inside the current Molen prototype. The ρ -VEX has to perform at least better than this processor to expose advantage.
- **Molen hardware CCU** The other ‘competitor’ of the ρ -VEX is a Molen hardware CCU. The performance of is targeted to be in between the performances of the PowerPC and a CCU.

It is clear that we chose not to benchmark a 4-cluster VEX machine. There are two main reasons for this: a 4-cluster VEX machine is beyond the scope of the ρ -VEX project [12], and the performance gain of a 4-cluster compared to a 2-cluster Lx machine is minimal, as was shown in Section 4.1.

We performed all benchmarks with the VEX machines in 3 different configurations, the ones presented in Section 3: default (DEF) VEX cluster configuration (32 KB 4-way set associative I- and D-cache), the ‘no cache’ (NO) configuration (16 B direct mapped I- and D-cache), and the proposed (PROP) configuration (4 KB 4-way set associative I- and D-cache).

Except for performance, ρ -VEX also competes area-wise with a CCU, because they are both configured in the same reconfigurable fabric. A user may opt for a ρ -VEX co-processor in the reconfigurable fabric of the Molen machine, or save area to have more CCUs. The ρ -VEX is supposed to perform ‘pretty well’ on many application-specific kernels, whereas a CCU is supposed to perform ‘really well’ on one specific task. When making this trade-off, the user has to take into account the number of application-specific kernels his application has, and what the relative speedup is for a CCU compared to the ρ -VEX and to the PowerPC 405 processor. It should be noted that the current Molen prototype is configured without I- and D-cache for the PowerPC 405 processor, because of limitations in the used FPGA technology.

4.2.1 Benchmark Set

Our benchmark set consisted of 4 benchmarks, which are being discussed below. Two of our benchmarks are mod-

ified examples from the Molen prototype example set [9]. This means that a representable hardware CCU is available, and we can have benchmark results from all machines. The other 2 kernels are chosen within the (media) application domain, because ρ -VEX is meant to excel in this domain.

- **G.723 Audio Encode** The G.723 [11] audio encoding technique is used for the encoding of (voice) audio in Voice over IP telephony applications. For this benchmark, an 8-bit A-law input audio signal of 928 bytes is encoded to a 24 kbps G.723 encoded signal. Our used implementation of the G.723 encoder was released to the public domain by Sun Microsystems [10]. This benchmark was chosen because it is a real-life application in the domain the computing machine is targeted at.
- **Matrix Multiplication** We created an application which multiplies two 64x64 semi-randomly integer-filled matrices. We created this application because both ALU parallelism would be exposed, as well as prefetching possibilities.
- **Min/Max from Array** This application resides within the Molen prototype examples set [9]. It determines the minimum and maximum values of a semi-randomly filled array with 16 integers. This benchmark exposes a lot of branches and comparisons.
- **Moving Filter** This last application originates also from the Molen prototype examples set. It transfers a signal represented by a semi-randomly filled array of 32 integers through a filter. This benchmark was chosen, because it showed a lot of parallelism in terms of ALU usage.

The VEX simulator generates extensive log files after each simulation, of which we were able to fetch the cycle counts of the different VEX machines. To obtain the number of execution cycles for the PowerPC 405, we used the internal cycle counter of the PowerPC. We fetched the cycle count with inline assembly lines. The number of execution cycles for the Molen CCU are relative to the PowerPC. We let the PowerPC start counting cycles at the call to a CCU function, and we stopped the counter after obtaining a result.

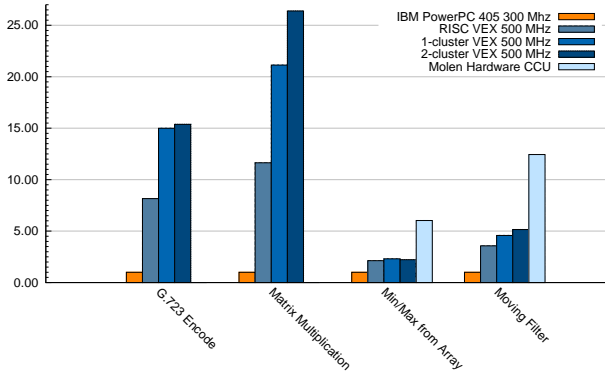


Figure 5. VEX performance chart based on default cache configurations (DEF)

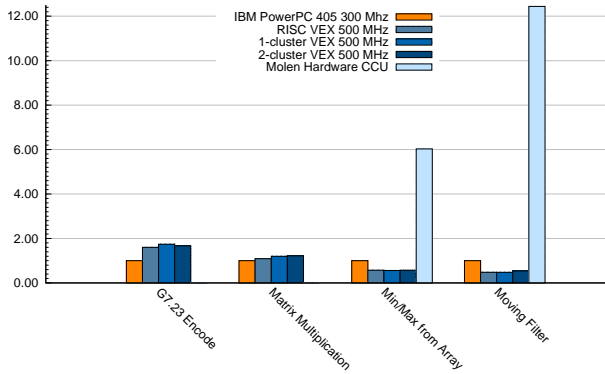


Figure 6. VEX performance chart based on 'no cache' configurations (NO)

All benchmarks have been performed with the compiler set to medium optimization. Because the Molen prototype gcc-based compiler 2.0 α (PPC-Molen) currently only compiles with low optimization, we redid those benchmarks with the gcc-based PowerPC which is bundled with Xilinx EDK 8.1i (PPC-EDK).

4.2.2 Benchmark Results

The results of our benchmarks are presented in Table 1. As the different cache configurations only apply to the VEX machines, the PowerPC and Molen CCU results are only once presented per benchmark. Figure 5 depicts the performance chart based on default VEX configurations. The performance in clock cycles of the PowerPC 405 processor (with application code compiled by the gcc compiler included with EDK) is set as baseline. Subsequently, Figure 6 depicts the performance chart based on the 'no cache' configurations. Finally, Figure 7 depicts the performance chart based on the proposed cache configurations.

It can be seen from the charts that cache size heavily influences the results of the VEX machines. The biggest differences can be seen in the Matrix Multiplication benchmark. With default cache sizes, a 2-cluster VEX machine

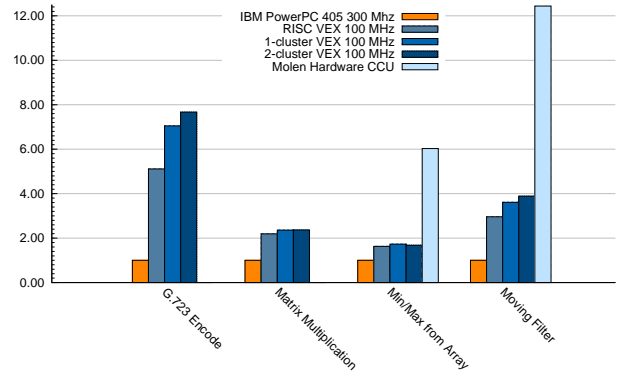


Figure 7. VEX performance chart based on proposed cache configurations (PROP)

outperforms the PowerPC processor 27 times. In contrast, when no cache memory is configured, the PowerPC and the VEX machines perform equally. When the proposed cache sizes of 4 KB are applied, the VEX machines outperform the PowerPC by a factor 2 approximately.

When looking at the performance difference between a VEX machine in RISC configuration and a 1-cluster VEX machine (in the default cache configuration benchmarks), we see performance improvements ranging from 10% to 100%. When the differences between a 1-cluster VEX machine and a 2-cluster VEX machine are observed, we see improvements ranging from -5% to 20%.

When no cache is used, we can see that VEX machines perform worse than the PowerPC, in the Min/Max and Moving Filter benchmarks.

The proposed configuration performs well, considering its relative small cache sizes. The single cluster VEX machine and the RISC VEX machine even beat their counterparts in the default configuration with 8 times the amount of cache memory available in the Matrix Multiply benchmark. Generally speaking, the performance gain of the VEX machines with 4 KB instruction- and data-cache ranges between 80% and 700% compared to the PowerPC.

It should be kept in mind that especially the Matrix Multiply and the Min/Max benchmarks benchmark are a really small part of a real life application. A speedup of the full application is heavily influenced by Amdahl's law:

$$S_i = \frac{T}{T - T_{PPC_i} + T_{VEX_i}} = \frac{1}{1 - (a_i - \frac{a_i}{s_i})} \quad (3)$$

In (3), S_i is the total speedup of the application, T is the total amount of execution cycles needed of the application in software only, T_{PPC_i} is the amount of execution cycles of the application kernel in software, T_{VEX_i} is the amount of execution cycles of the application kernel on a VEX machine. a_i is the percentage of time used by the application kernel in software ($\frac{T_{PPC_i}}{T}$), and s_i is the speedup of the application kernel on a VEX machine compared to software execution ($\frac{T_{PPC_i}}{T_{VEX_i}}$).

5 Implementation Proposal

Based on our performance measurements on different VEX machines, we propose to design the ρ -VEX processor in two phases:

1. **ρ -VEX processor with RISC configuration** We will first start designing a ρ -VEX processor in a RISC configuration. A RISC VEX processor seems to be a good starting point to evolve the processor to a more advanced one.
2. **1-cluster ρ -VEX processor** After we have successfully implemented a ρ -VEX processor with RISC configuration, we will expand this design to a 1-cluster VEX machine. All VEX configurations will in first instance be equal to the default 1-cluster VEX machine, to ensure optimal compatibility with the toolchain.

All VEX resource management parameters will be implemented in a robust way, so that the default configuration can be easily changed. We want to keep the openness of the VEX architecture in our design.

Initially, the implementations will not make use of cache memory. Eventually, a cache system will be designed and implemented. This cache system will initially be based on the suggested cache configuration (4 KB instruction- and data-cache). It will be a robust design, so that the amount of cache memory will be flexible, depending on the host FPGA platform.

6 Conclusions

In this paper we presented a preliminary analysis of the performance and configuration for the ρ -VEX project, about the design and implementation of a VLIW co-processor for the Molen reconfigurable machine prototype.

Our simulation results show that a ρ -VEX co-processor with enough cache memory could deliver substantially performance gains for different application kernels.

An appealing advantage of a ρ -VEX co-processor in contrast to a Molen hardware CCU is the fact that it can provide performance gains for a large number of application kernels, when using only a fixed amount of area in the reconfigurable fabric. A CCU can only perform one type of calculation (which results in a higher performance gain) on a fixed amount of area.

We suggest a design space in which first a ρ -VEX in RISC configuration is implemented, and then a 1-cluster configuration. The whole design is supposed to be very robust, so that most configuration parameters will be parameterizable in the hardware implementation of the ρ -VEX.

References

- [1] P. Faraboschi, G. Brown, J.A.Fisher, G. Desoli, and F. Homewood. Lx: A Technology Platform for Cus-

tomizable VLIW Embedded Processing. In *Proceedings of the 27th annual International Symposium of Computer Architecture*, pages 203–213, June 2000.

- [2] J.A. Fisher, P. Faraboschi, and C. Young. *Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools*. Morgan Kaufmann, 2004.
- [3] J.L. Hennessy and D.A Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, third edition, 2003.
- [4] Hewlett-Packard Laboratories. *VEX Toolchain*. <http://www.hpl.hp.com/downloads/vex/>.
- [5] C. Iseli and E. Sanchez. Spyder: a Reconfigurable VLIW Processor using FPGAs. In *FPGAs for Custom Computing Machines*, pages 17–24, January 1993.
- [6] A.K. Jones, R. Hoare, D. Kusic, J. Fazekas, and J. Foster. An FPGA-based VLIW Processor with Custom Hardware Execution. In *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on FPGAs*, pages 107–117, 2005.
- [7] G.K. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis. The Virtex II Pro MOLEN Processor. In *Proceedings of the 4th International Workshop on Computer Systems: Architectures, Modelling, and Simulation (SAMOS 2004)*, pages 192–202, July 2004.
- [8] G.K. Kuzmanov, G.N. Gaydadjiev, and S. Vassiliadis. The Molen Media Processor: Design and Evaluation. In *Proceedings of the International Workshop on Application Specific Processors, WASP 2005*, pages 26–33, September 2005.
- [9] Delft Computer Engineering Laboratory. *Molen Prototype Examples*. <http://ce.et.tudelft.nl/MOLEN/Prototype/>.
- [10] Sun Microsystems. *G.723 Software Implementation*. <http://www.sun.com/>.
- [11] International Telecommunication Union. *G.723 Audio Codec*. <http://www.itu.int/rec/T-REC-G.723/e>.
- [12] T. van As. ρ -VEX: A Parameterizable and Reconfigurable VLIW processor Core for Molen – MSc Project Schedule. December 2007.
- [13] S. Vassiliadis, S. Wong, and S. D. Cotofana. The MOLEN $\rho\mu$ -coded Processor. In *11th International Conference on Field-Programmable Logic and Applications (FPL), Springer-Verlag Lecture Notes in Computer Science (LNCS) Vol. 2147*, pages 275–285, August 2001.
- [14] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K.L.M. Bertels, G.K. Kuzmanov, and E. Moscu Panainte. The Molen Polymorphic Processor. pages 1363– 1375, November 2004.

[15] Xilinx. *Virtex-II Pro FPGA*. <http://www.xilinx.com/>.

[16] Xilinx. PowerPC 405 Processor Block Reference Guide. June 2005.