Polymorphic Architectures - from Media Processing to Supercomputing

Georgi Kuzmanov

Abstract: *This paper reveals the evolution of the polymorphic architectures in the context of ever increasing computational demands of the user applications and the need for formal architectural abstraction covering the emerging reconfigurable technologies. The base architecture presented is the Molen polymorphic processor - a synergism between a general purpose processor (GPP) and a reconfigurable accelerator. Its operation is based on the co-processor architectural paradigm and employs the concept of the traditional microcode control. Experiments with popular media applications, such as MJPEG, MPEG-2, MPEG-4 suggest that a Molen architecture can provide speedups closely approaching the theoretically maximum obtainable figures, determined by Amdahl's law. However, while media applications are predominantly integer based, scientific applications, typically run on supercomputers, rely on floating-point arithmetic. Therefore, a natural evolution of the Molen platforms towards supercomputing, i.e., towards floating-point highly demanding operations, is in progress. The author presents some experimental results obtained for a typical supercomputing kernel, matrix multiplication, implemented as a Molen reconfigurable accelerator. These results demonstrate the advantages of the polymorphic approach agains traditional GPP based scientific computing in terms of high performance. Finally, the paper proposes author's vision for the future evolution of high-end polymorphic architectures.*

Key words: *Polymorphism, Reconfigurable computing, Media applications, Supercomputing.*

INTRODUCTION

Historically, the most popular type of reconfigurable hardware - the Field Programmable Gate Arrays (FPGA) - emerged in the early 1980s and for more than a decade were considered mainly for prototyping small hardware designs. By the middle of 1990's, FPGAs technological improvement allowed them not only to be considered as powerful prototyping platforms for large volume Application Specific Integrated Circuits (ASIC), but also to be employed as main computational components in end-product specialized computer designs when small production volumes were needed. Further growth of the computational capabilities of the FPGAs made them a good candidate for co-processors of General Purpose Processors (GPPs), which in turn led to embedding the latter in the silicon fabric of the so-called platform FPGAs. Currently, multiple GPP cores are being included as hard-cores in modern FPGAs, marking the beginning of the multicore era in the reconfigurable computer design. This tremendous technological growth of the reconfigurable hardware industry and its overlap with the GPP world rose the necessity to consider the FPGAs, along with their specific features, from an architectural prospective. In this work, we adopt the terminology provided by [3], which defines the architecture of any computing system to be its conceptual view and functional behavior as seen by its immediate user, i.e., by the programmer. The other, underlying layers of the computer design, defined in [3] are the implementation (i.e., the logical organization of the architecture, also referred to as microarchitecture) and the realization (physical technology, not considered further in this paper). One of the first and rather complete attempts to classify the variety of reconfigurable proposals from the architectural point of view was provided in [14]. There, the authors proposed a taxonomy of the field-programmable custom computing machines (FCCM - a GPP extended with reconfigurable hardware) with respect to the existence of an explicit SET instruction that configures the hardware. Additionally, a discussion on the microcoding prospective of the

reconfigurable machines was provided in [14]. The inclusion of the microcode as a powerful means to manage both configurations and executions on reconfigurable hardware was first proposed with the Molen polymorphic processors [17, 18]. The idea of reconfigurable microcoded ($\rho\mu$-coded) polymorphic processors was further applied in the multimedia domain, as well as in cryptography and in other computationally demanding fields, it proved to be performance efficient, flexible and designer friendly. In the context of this presentation, the term 'polymorphic' refers to the architectural aspects of the computing systems implemented on any reconfigurable technology, namely to the functional behavior of a computing system as seen by its programmer. Further increase of the number of computational resources on the contemporary FPGAs allowed achieving higher level of parallelism and computational density per chip compared to GPPs. This introduced the reconfigurable hardware in modern supercomputing designs [7], mainly as powerful and flexible application specific accelerators.

In this paper, we summarize the results of our earlier architectural research on accelerating highly demanding applications using reconfigurable hardware. We describe the Molen $\rho\mu$-coded processors and provide some results for their typical application domains such as media and cryptography. More specifically, our experimental results on real prototyping platforms based on Xilinx Virtex II Pro technology suggest application speedups closely approaching the theoretically maximum attainable ones, as determined by Amdahl's law [1]. We provide a short discussion on the programmability of the described polymorphic architectures and present the Delft Workbench - a current tool development effort targeting to facilitate the design process for reconfigurable hardware. Furthermore, we describe an implementation of the Molen polymorphic paradigm in the supercomputing domain, briefly illustrated by a matrix implementation example, which significantly outperforms related work given limited bandwidth and hardware resources. Finally, the paper is concluded with a discussion on open research directions, such as reconfigurable multithreaded architectures, scientific floating-point kernels, reconfigurable memory subsystems, tool development.

THE MOLEN $\rho\mu$-CODED POLYMORPHIC PROCESSOR

Before describing the Molen processors, we provide a brief theoretical study on the potentials for speeding up applications based on the Amdahl's law. We use Amdahl's law to determine the interval of the speedups, potentially achievable and worthy to be targeted by FCCM designers.

Riding Amdahl's curve. Generally, the maximum theoretically attainable (i.e., the potentially maximum) speedup, with respect to the parallelizable portion of the program code, is determined by Amdahl's law [1]:

$$S_{MAX} = \frac{1}{(1-a)} \tag{1}$$

Where S_{MAX} denotes the overall application speedup and a - the parallelizable fraction of the program from the total execution time. Amdahl's curve, graphically illustrated in Figure 1, suggests that if, say, half of an application program is parallelized and that its entire parallel fraction is assumed to execute in zero time, the speedup would potentially achieve a factor of two. Moreover, Amdahl's curve suggests that to achieve an order of magnitude speedup, a designer should consider for parallelization over 90% of the application execution time. In such cases, when over 90% of the application workload is considered for parallelization, it is practical to create an ASIC, rather than utilizing programmable GPPs. The design cycle of an ASIC, however, is extremely inflexible and very expensive. Therefore, ASICs may not appear to be an efficient solution when we consider smaller portions (i.e., $a < 0.9$) of an algorithm for hardware acceleration. Obviously, there exist potentials for new hardware proposals that perform considerably better than GPPs (say $a > 0.5$) and are a more flexible

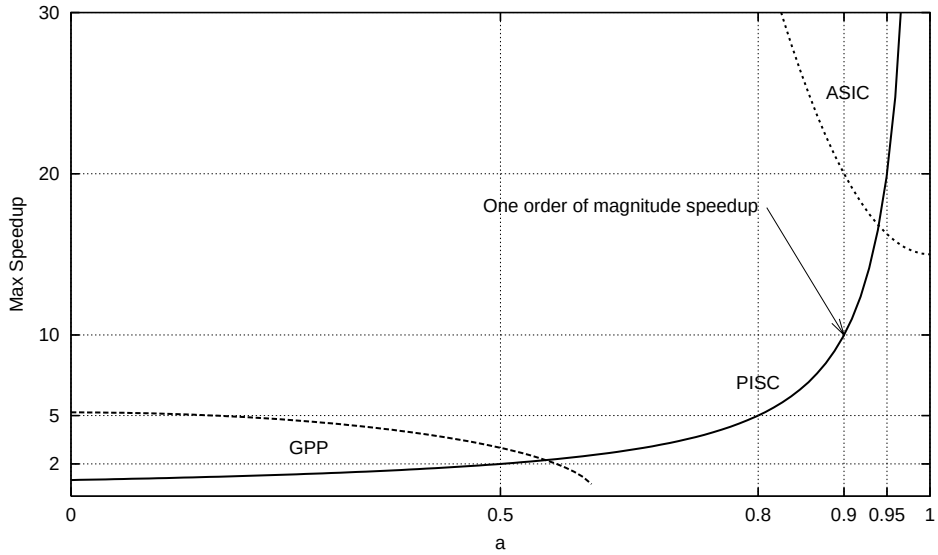

Figure 1: The Amdahl's curve and PISC.

alternative than ASICs. The design gap between the GPPs and the ASICs, conditionally considered to be $0.5 < a < 0.9$ (Figure 1), is therefore found to be efficient for FCCMs. The Molen $\rho\mu$-coded FCCMs are based on an architectural paradigm targeting the existing gap between GPPs and ASICs in terms of flexibility and performance. In consistence with the classical RISC and CISC paradigms, we refer to the Molen architectural paradigm as to a Polymorphic Instruction Set Computer (PISC) [16], see Figure 1.

The Molen $\rho\mu$-coded FCCM. The Molen paradigm provides a designer with potentials to benefit from the best of two worlds, i.e., a synergism between purely programmable solutions on GPPs and reconfigurable hardware. The main idea is that the GPP controls both the configuration and the execution of computationally intensive kernels on the reconfigurable co-processor using the concept of the traditional microcode control. That is, the infinite flexibility of the programmable GPPs combined with reconfigurable accelerators results into a PISC - a programmable system that substantially outperforms the GPPs.

Processor organization: The two main components in the Molen machine organization (depicted in Figure 2) are the *Core Processor*, which is a general-purpose processor (GPP), and the *Reconfigurable Processor* (RP). The ARBITER performs a partial decoding on the instructions in order to determine where they should be issued. General purpose instructions are issued to the GPP. Instructions for custom execution are redirected to the RP. Data transfers from(to) the main memory are handled by the *Data Load/Store* unit. The *Data Memory MUX/DEMUX* unit is responsible for distributing data between either the reconfigurable or the core processor. The reconfigurable processor consists of the *reconfigurable microcode* ($\rho\mu$-code) unit and the *custom computing unit* (CCU). The CCU consists of reconfigurable hardware and memory, intended to support additional and future functionalities that are not implemented in the core processor. Pieces of application code can be implemented on the CCU in order to speed up the overall execution of the application. A clear distinction exists between code that is executed on the RP and code that is executed on the GPP. The parameter and result passing between the RP targeted code and the remainder application code is performed utilizing the *exchange registers* (XREGs), depicted in Figure 2. For more details on each functional block of the Molen organization, the interested reader is referred to [18].

Polymorphic operations: An operation, executed by the RP, is divided into two distinct phases: *set* and *execute*. The set phase is responsible for reconfiguring the CCU for the operation. In the execute phase, the actual execution of the operations is performed. No specific instructions are associated with specific operations to configure and execute on the CCU. Instead, pointers to *reconfigurable microcode* ($\rho\mu$-code) are utilized. The $\rho\mu$-code

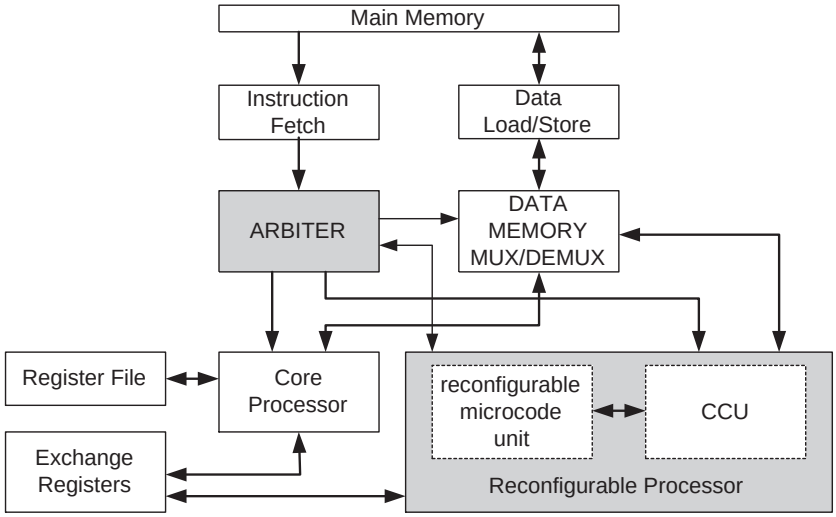

Figure 2: The Molen machine organization

emulates both the configuration and the execution of CCU implementations resulting in two types of microcode: 1) reconfiguration microcode that controls the CCU configuration; and 2) execution microcode that controls the execution of the CCU configured implementation.

The polymorphic instruction set: The complete list of the eight instructions, supporting the Molen paradigm (for details see [15]) is denoted as polymorphic instruction set architecture (πISA). These instructions are: 1) partial set (p-**set** $<address>$) performs common and frequently used configurations; 2) complete set (c-**set** $<address>$) *completes* the CCU's configuration to perform less frequent functions; 3) **execute** $<address>$: controls the execution of the operations on the CCU configured by the set instructions; 4) **set prefetch** $<address>$, and 5) **execute prefetch**: prefetch the needed microcodes responsible for CCU reconfigurations and executions into a local on-chip storage (the $\rho\mu$-code unit); 6) **break**: synchronizes the parallel execution of the RP and the GPP; 7) **movtx** $\text{XREG}_a \leftarrow \text{R}_b$, and 8) **movfx** $\text{R}_a \leftarrow \text{XREG}_b$: move the content of general-purpose register R_b to/from XREG_a. The $<address>$ field denotes the location of the reconfigurable microcode responsible for the configuration and execution processes.

Prototyping and experimental evaluation. A prototype of a Molen organization was initially developed for Xilinx Virtex II Pro technology and later transferred for Virtex 4 FX. More details on the prototype implementation can be found in [11] and on-line at `http://ce.et.tudelft.nl/MOLEN/Prototype/`. To evaluate the performance of the Molen processor, we first considered the overall speed-up S of the application with respect to multiple kernels, determined by Amdahl's law for a real case:

$$S = \frac{1}{(1-a) + \sum_i \frac{a_i}{s_i}} \tag{2}$$

Where a_i is the fraction taken by kernel i from the total time of the sequential software execution, and s_i is the (local) speedup of kernel i in the Molen execution scenario. We note that both a_i and s_i are experimentally obtained from the prototype. The Virtex II Pro Molen prototype and (2) were used for experimental platform evaluation with different media applications, such as MJPEG, MPEG-2, and MPEG-4.

Evaluation methodology: A real experimental evaluation approach is employed. In this approach, the original application code is compiled and run on the core processor as a pure software. The duration of the entire program execution is measured in number of processor clock cycles obtained from the built-in PowerPC timers. In the following step, the benchmark program is annotated to support the reconfigurable implementations of some application specific kernels. Those kernel implementations are loaded into the Molen prototype and the

annotated program is then executed on the prototype. The ratio between the measured cycle numbers for the pure software execution and the program execution in the Molen execution scenario represents the actual (real) speedup of the application.

Reconfigurable units considered: Regarding the MJPEG testbench, we utilized four computationally demanding operations of the encoding algorithm, namely *block input*, *pre-shift*, *2D-DCT* and *block output*. Regarding MPEG-2, we considered the Sum-of-Absolute-Differences (SAD) operation, the Discrete-Cosine Transform (DCT), and its inverse one (IDCT). Furthermore, we have implemented two MPEG-4 specific reconfigurable units, supporting the repetitive padding and the ACQ function. We clock the DCT, IDCT, Padding, and ACQ CCUs at mem_clk frequency (100 MHz). The SAD designs are clocked by PPC_clk at 300MHz.

MJPEG experiments: We considered an MJPEG source code written in C++ and the real experimental approach to evaluate the performance gains. For the experiments, we considered an image size of 48×48 and 4:2:2 YUV macroblock format. In Table 1, column 5 ("impl."), the real experimental speedups are presented for three picture sequences: *tennis*, *barbara*, and *artemis*. The DCT* software kernel constitutes roughly 61% of the total execution time of the pure software MJPEG encoding algorithm. The theoretical maximum speedups, according to Amdahl's law are reported in column 6. Finally, in column 7, we estimate how close the experimental measurements are to the theoretical maximum speedups (speedup efficiency in %).

Table 1: Real MJPEG speedup by the DCT* Molen CCU implementation.

sequence	frames number	Total MJPEG execution [cycles]		Speedup		Speedup efficiency
		Software	DCT* CCU	impl.	theory	%
tennis	8	84434216	40263576	**2.10**	2.57	81.49
barbara	1	85371112	41131512	**2.08**	2.53	81.94
artemis	1	85577112	41354208	**2.07**	2.52	82.01

MPEG-2 experiments: We target the Berkeley implementation of the MPEG-2 encoder and decoder from libmpeg2. The considered reconfigurable designs of SAD, DCT and IDCT are embedded in the Molen prototype. Table 2 presents the measured real kernel speedups and the projected overall MPEG-2 speedups. Columns labeled "theory" present the theoreti-

Table 2: MPEG-2 kernel speedups and overall speedups

	Kernel			Overall					
	SAD128	DCT	IDCT	MPEG2 encoder			MPEG2 decoder		
				theory	impl.	efficiency%	theory	impl.	efficiency%
carphone	18.9	302.3	24.4	2.85	**2.64**	93	2.02	**1.94**	96
claire	23.9	302.2	24.4	2.99	**2.80**	94	1.60	**1.56**	98
container	35.2	302.1	24.4	3.12	**2.96**	95	1.68	**1.63**	97
tennis	35.0	302.1	32.3	3.37	**3.18**	94	1.68	**1.65**	98

cally attainable maximum speedup calculated with respect to Amdahl's law. Columns labeled with "impl." contain data for the measured speedups on the considered Molen prototype. For the MPEG-2 encoder, the simultaneous configuration of the SAD128, DCT, and IDCT operations has been considered. For the MPEG-2 decoder, only the IDCT reconfigurable implementation has been employed. The results suggest that the real experimental MPEG-2 speedups closely approach the theoretically estimated maximum attainable speedups (see columns "efficiency").

MPEG-4 experiments: We consider the publicly available source code of the MoMuSys project, which strictly implements the MPEG-4 video verification model. We also consider

Table 3: Considered MPEG-4 scenarios - kernel (s_i) and projected overall (S) speedups.

Scenario	a	Padding		ACQ		SAD		DCT		IDCT		$\bar{s}_{av}$	S	S_{MAX} theory
		a_i	s_i	a_i	s_i	a_i	s_i	a_i	s_i	a_i	s_i			
MPEG-4 Encoder														
Sc.1 [10]	0.84	0.038	100	0.104	46	0.660	28	0.006	300	0.010	26	31.7	**5.48**	6.41
Sc.2 [9]	0.82	0.039	100	0.104	46	0.660	28	0.005	300	0.008	26	30.7	**4.75**	5.43
Sc.3 [4]	0.97	0.001	100	0.064	46	0.900	28	0.004	300	0.005	26	28.9	**16.74**	38.46
Sc.4 [5]	0.96	0.007	100	0.056	46	0.880	28	0.008	300	0.007	26	29.0	**13.28**	23.64
MPEG-4 Decoder														
Sc.1 [10]	0.24	0,155	100	N.A.	46	N.A.	28	N.A.	300	0.089	26	49.1	**1.31**	1.32
Sc.2 [9]	0.24	0,155	100	N.A.	46	N.A.	28	N.A.	300	0.088	26	49.2	**1.31**	1.32
Sc.3 [4]	0.27	0,042	100	N.A.	46	N.A.	28	N.A.	300	0.226	26	29.4	**1.35**	1.37
Sc.5.1 [2]	0.16	0,160	100	N.A.	46	N.A.	28	N.A.	300	0.001	26	98.3	**1.19**	1.19
Sc.5.2 [2]	0.28	0,270	100	N.A.	46	N.A.	28	N.A.	300	0.010	26	90.8	**1.38**	1.39
Sc.5.3 [2]	0.23	0,140	100	N.A.	46	N.A.	28	N.A.	300	0.090	26	47.3	**1.29**	1.30

profiling results, reported in the literature [2, 4, 5, 9, 10], to establish different MPEG-4 scenarios. From the experimentally measured real kernel speedups we determine an average kernel speedup ($\bar{s}_{av}$), which takes into account the individual contribution of each kernel with respect to its speedup and its part from the entire application execution time. The kernel and the overall application speedups of the same MPEG-4 high profile applications in the considered scenarios are presented in Table 3. Scenarios 5.1, 5.2, and 5.3 (extracted from [2]) correspond to bitstreams L6, L5, and Children, respectively.

PROGRAMMABILITY AND DESIGN TOOLS

A set of Molen design tools and compilers [13] is included in the so-called Delft Workbench (`http://ce.et.tudelft.nl/DWB/`), schematically illustrated in Figure 3. First, the code to be executed on the reconfigurable hardware must be determined. This is achieved by high-level to high-level instrumentation and benchmarking. This results in several candidate pieces of code. Second, we must determine which piece of code is suitable for implementation on the reconfigurable hardware. The suitability is solely determined by whether the piece of code is implementable (i.e., "fits in hardware"). Those parts can then be mapped into hardware via a hardware description language (HDL). In case the HDL corresponds to "critical" hardware in terms of, for instance, area, performance, memory and power consumption, the translation will be done manually. Otherwise, the translation can be done automatically using, e.g., the DWARV tool [19] or extracted from a library.

RECONFIGURABLE SUPERCOMPUTING

Traditional supercomputing systems consist of vast numbers of GPPs interconnected in various network topologies. Such system setups allow resolving complex scientific problems requiring enormous computational power, such as fluid dynamics simulations, weather forecast, nuclear fusion and fission simulations, various biochemical modeling applications, etc. The recent development of the reconfigurable technology concentrated substantial computational power in a single FPGA device, including on-chip hard and soft general purpose cores. The huge amount of computational resources, combined with the high processing parallelism that can be obtained by the recent reasonably sized reconfigurable chips, make the latter suitable for efficiently performing supercomputing tasks using floating-point operations. This concentration of resources potentially allows higher computational density per device when reconfigurable technology is employed compared to traditional GPP-based systems. An additional advantage of the FPGAs is their flexibility and adaptability to application-specific computational problems.

Matrix multiplication example: We shall demonstrate how a typical supercomputing problem, such as floating-point general matrix multiplication (GEMM), can be efficiently solved by a Molen-based polymorphic design. We employ as a Molen CCU the multi pro-

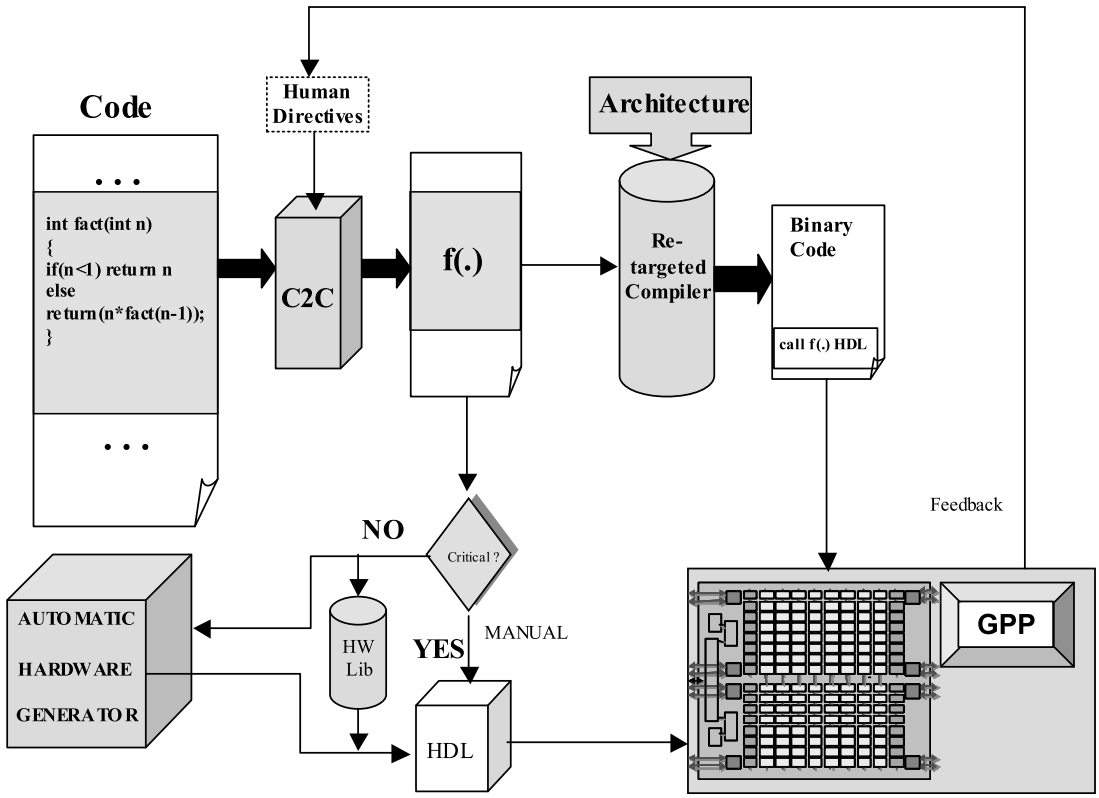

Figure 3: The Delft Workbench design flow.

cessing element (PE) design for FPGAs, initially proposed in [6] and sketched in Figure 4. More details on the implementation of the GEMM CCU can be found in [12].

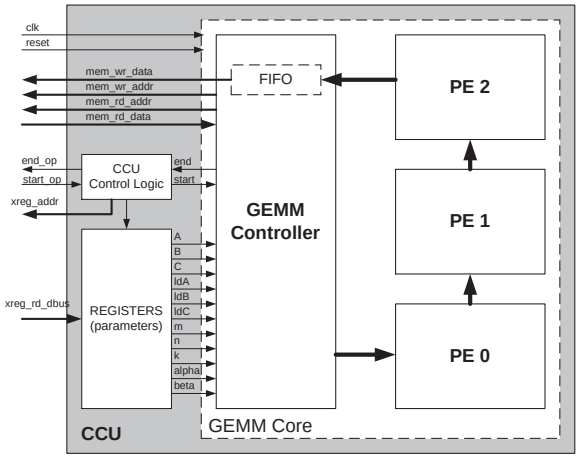

Figure 4: The GEMM core with the CCU interface.

Table 4: CCU implementation of GEMM on XC2VP30–6 (post place-and-route)

PEs	S_i	S_j	Slices	Frequency	P_{peak}@100MHz
1	96	64	2844	101.092 MHz	200MFLOPS
2	96	64	4313	100.301 MHz	400MFLOPS
3	96	64	5726	100.321 MHz	600MFLOPS
4	64	64	7317	100.251 MHz	800MFLOPS
5	80	64	8964	100.271 MHz	1000MFLOPS
6	96	64	10688	100.010 MHz	1200MFLOPS
7	112	64	11843	100.241 MHz	1400MFLOPS
8	64	64	12296	100.251 MHz	1600MFLOPS
9	72	64	13429	100.050 MHz	1800MFLOPS

In software, the Molen programming paradigm constitutes the matrix multiplication as a normal function call. Our function prototype closely resembles the double-precision general matrix multiply (dgemm) routine of the Basic Linear Algebra Subprograms (BLAS) [8]. The BLAS GEMM is defined as follows: **C** $\leftarrow \alpha$**AB** $+ \beta$**C**, where **A**, **B** and **C** are matrices of dimensions $m \times k$, $k \times n$ and $m \times n$, respectively; α and β are scaling factors. To solve **C** $\leftarrow$ **AB** $+$ **C**, we employed the block matrix multiplication algorithm [6], which can be efficiently implemented on a linear array of processing elements (PE). The result matrix C is computed in blocks of $S_i \times S_j$ words, denoted as **C'**. Each block **C'** is the product of S_i rows of matrix **A** and S_j columns of matrix **B**, denoted as submatrices **A'** and **B'**, respectively. The data from matrix **A'** are loaded in column-major, the data from **B'** - in row-major order in the PEs. Each element of **A'** or **C'** is transferred to/from one PE only. The data from matrix **B** are sent to all PEs, in order to compute S_i products in parallel. Our Molen CCU implementation

Table 5: Square matrix multiplication performance in MFLOPS. The CCU runs at 100 MHz.

n	1 PE	2 PEs	3 PEs	4 PEs	5 PEs	6 PEs	7 PEs	8 PEs	9 PEs
10	143	223	251	287	332	332	332	332	330
20	178	320	420	532	613	613	723	723	723
30	187	348	490	586	728	828	828	961	960
40	190	363	496	659	789	875	982	1120	1120
50	192	369	524	663	827	901	991	1099	1235
60	193	374	544	703	853	994	1084	1198	1323
70	197	388	558	718	883	998	1148	1241	1350
80	197	390	570	760	938	1063	1225	1447	1562
90	198	391	580	749	962	1122	1281	1380	1629
100	199	395	577	765	970	1103	1247	1424	1534
300	199	399	599	799	998	1198	1391	1573	1758
500	200	399	599	799	999	1190	1388	1585	1784
1000	200	400	599	799	999	1197	1398	1599	1785
P_{peak}	200	400	600	800	1000	1200	1400	1600	1800

(see Figure 4) has the following 11 parameters: `a`, `b` and `c` are the start addresses of the matrices; `m`, `n` and `k` are the dimensions of the matrices; `alpha` and `beta` are scaling vectors. We currently support $\alpha \in \{-1, +1\}$ and $\beta \in \{0, 1\}$ only. When a function is called in the Molen environment, instead of executing its software code, the processor will execute a code of specific Molen instructions. This code copies the function parameters to the exchange registers and then executes the corresponding hardware operation on the CCU.

Experimental Results: We have implemented a Molen prototype design with 9 PEs in a Xilinx XC2VP30–6 FPGA specifying 10ns (100 MHz) as a timing constraint. Table 4 contains the number of PEs, the parameters S_i and S_j, the resource usage (slices), and the frequency *after place-and-route*. Apparently, 100 MHz was achieved for all configurations, corresponding to the P_{peak} column of Table 4. The measured results for the sustained performance for square matrix multiplications on the Molen prototype are reported in Table 5. They include the software overhead, the GPP-CCU synchronization, and the parameters transfers. Clearly, the sustained performance approaches peak performance for large problem sizes and depends on the number of PEs. For example, with one PE, 95% of the peak performance is sustained for $n = 41$, with 9 PEs - for $n = 142$.

Performance evaluation: A complete comparison of the GEMM to related works has been provided in [6, 12], which suggest that the design, described above, significantly outperforms prior related works. In the context of this presentation, we would like to emphasize on the advantages the polymorphic approach has against GPP based GEMM implementations. We compared our design against three GPP systems: AMD Athlon 64 X2 3800+, 2 GHz, 64 kB L1 cache, 512 KB L2 cache and 1 GB DDR; Intel Pentium 4, 2.8 GHz, 8 kB L1 cache, 512 kB L2 cache and 1 GB DDR; and VIA C3, 1 GHz, 64 kB L1 cache, 64 kB L2 cache and 256 MB DDR. The comparison results are presented in Figure 5. All these systems indicated performance degradation for large matrix sizes, contrary to our design which scales better for large problems. This is a consequence of the limitations of the GPP cache systems.

CONCLUSIONS AND RESEARCH DIRECTIONS

We presented an overview of the polymorphic architectures evolution from the application prospective. The practical applicability boundaries of the polymorphic instruction set computers were discussed in the context of Amdahl's law. We presented the Molen polymorphic processor and illustrated its advantages over traditional general purpose computers in terms of performance. In our experiments, we briefly covered a wide range of applications - from fixed-point media processing to floating-point matrix multiplication, a typical supercomputing operation. Experimental results clearly suggest that polymorphic processors are a performance and cost efficient alternative to traditional general purpose computers in the considered computationally intensive application domains. Many new architectural challenges for the whole computing society, such as multithreading support, efficient memory

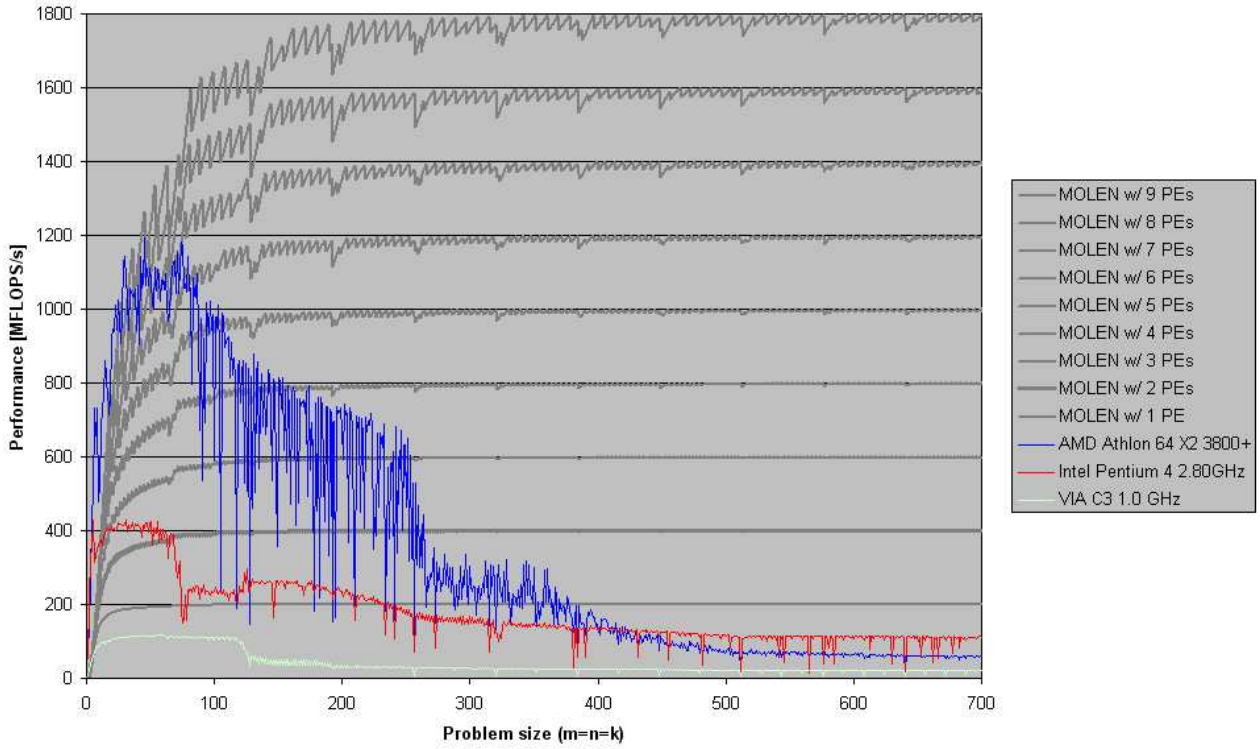

Figure 5: Performance the Polymorphic GEMM versus traditional GPP implementations.

subsystems, multicore organizations, increased on-chip computational density, but also the programmability of complex systems, could be successfully addressed in the context of the polymorphic architectures. Amdahl's law describes the performance envelope determined by the parallelizable and the strictly sequential portions of a single program, which in turn depend on the data dependencies in the algorithm. Yet, the polymorphic hardware has potentials for significant acceleration even beyond the application domain of Amdahl's law, e.g., in multithreading applications where low or no dependances between the executing threads exist. Managing reconfigurable resources in a multithreading context is a promising research direction towards more performance and cost efficient application specific computing. The illustrated example for polymorphic dense matrix multiplication indicates another interesting direction for performance efficient floating-point scientific computing including sparse matrix computations and various complex mathematic operators on reconfigurable co-processors. Reconfigurable technologies provide also a promising solution for flexible memory subsystem organizations, adaptable to the specific application bandwidth requirements. And last, but not least - new design methodologies,resulting to appropriate tools and toolsets should be devised to assist the designers in coping with the rapidly growing complexity and flexibility of the polymorphic computing platforms.

ACKNOWLEDGMENTS

To the memory of prof. Stamatis Vassiliadis who unfortunately deceased in 2007 leaving a huge emptiness in the scientific society of the computer architects. The described work would not be possible without the foundations he laid for it. The presented research has been part of a collective effort, therefore the author would like to acknowledge all his colleagues and students from the Computer Engineering Lab at TU Delft who collaborated with him. This work was partially sponsored by the Dutch Technology Foundation STW, applied science division of NWO and the Technology Program of the Dutch Ministry of Economic Affairs (project DCS.7533).

References

[1] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the AFIPS 1967 Spring Joint Computer Conference*, 1967, pp. 483–485.

[2] M. Berekovic, H.-J. Stolberg, M. B. Kulaczewski, P. Pirsh, H. Moler, H. Runge, J. Kneip, and B. Stabernack, "Instruction set extensions for mpeg-4 video," *Journal of VLSI Signal Processing*, vol. 23, no. 1, pp. 27–49, October 1999.

[3] G. Blaauw and F. Brooks, *Computer Architecture: Concepts and Evaluation*. Addison-Wesley, 1997.

[4] H.-C. Chang, L.-G. Chen, M.-Y. Hsu, and Y.-C. Chang, "Performance analysis and architecture evaluation of MPEG-4 video codec system," in *IEEE International Symposium on Circuits and Systems*, vol. II, 28-31 May 2000, pp. 449–452.

[5] H.-C. Chang, Y.-C. Wang, M.-Y. Hsu, and L.-G. Chen, "Efficient algorithms and architectures for MPEG-4 object-based video coding," in *IEEE Workshop on Signal Processing Systems*, 11-13 Oct 2000, pp. 13–22.

[6] Y. Dou, S. Vassiliadis, G. Kuzmanov, and G. N. Gaydadjiev, "64-bit Floating-Point FPGA Matrix Multiplication," in *ACM/SIGDA Thirteenth International Symposium on Field Programmable Gate Arrays (FPGA 2005)*, February 2005, pp. 86–95.

[7] S. G. Inc, "Reconfigurable application specific computer user's guide," 2008, `http://techpubs.sgi.com/library/`.

[8] S. H. J. J. Dongarra, Jeremy Du Croz and I. S. Duff, "A set of level 3 basic linear algebra subprograms," *ACM Trans. Math. Softw.*, vol. 16, no. 1, pp. 1–17, 1990.

[9] J. Kneip, S. Bauer, J. Vollmer, B. Schmale, P. Kuhn, and M. Reissmann, "The MPEG-4 video coding standard - a VLSI point of view," in *IEEE Workshop on Signal Processing Systems,(SIPS98)*, 8-10 Oct. 1998, pp. 43–52.

[10] P. Kuhn and W. Stechele, "Complexity analysis of the emerging MPEG-4 standard as a basis for VLSI implementation," in *SPIE Visual Comunications and Image Processing (VCIP)*, vol. 3309, Jan. 1998, pp. 498–509.

[11] G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis, "The MOLEN Processor Prototype," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)*, April 2004, pp. 296–299.

[12] G. Kuzmanov and W. van Oijen, "Floating-point matrix multiplication in a polymorphic processor," in *International Conference on Field Programmable Technology (ICFPT)*, December 2007, pp. 249–252.

[13] E. Moscu Panainte, K. Bertels, and S. Vassiliadis, "Compiling for the Molen Programming Paradigm," in *13th International Conference on Field Programmable Logic and Applications (FPL)*, vol. 2778. Springer-Verlag Lecture Notes in Computer Science (LNCS), Sep 2003, pp. 900–910.

[14] M. Sima, S. Vassiliadis, S. D. Cotofana, J. T. J. van Eijndhoven, and K. A. Vissers, "Field-programmable custom computing machines - a taxonomy," in *Proceedings of the 12th International Conference on Field-Programmable Logic and Applications (FPL 2002). Reconfigurable Computing Is Going Mainstream*, September 2002, pp. 79–88.

[15] S. Vassiliadis, G. Gaydadjiev, K. Bertels, and E. Moscu Panainte, "The Molen Programming Paradigm," in *Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation*, July 2003, pp. 1–7.

[16] S. Vassiliadis, G. Kuzmanov, S. Wong, E. M. Panainte, G. N. Gaydadjiev, K. Bertels, and D. Cheresiz, "PISC: Polymorphic Instruction Set Computers," in *Proceedings of the International Workshop on Applied Reconfigurable Computing (ARC 2006)*, March 2006, pp. 274–286.

[17] S. Vassiliadis, S. Wong, and S. Cotofana, "The MOLEN $\rho\mu$-Coded Processor," in *11th International Conference on Field Programmable Logic and Applications (FPL)*, vol. 2147. Springer-Verlag Lecture Notes in Computer Science (LNCS), Aug 2001, pp. 275–285.

[18] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The Molen Polymorphic Processor," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1363–1375, November 2004.

[19] Y. D. Yankova, G. Kuzmanov, K. Bertels, G. N. Gaydadjiev, Y. Lu, and S. Vassiliadis, "DWARV: DelftWorkbench Automated Reconfigurable VHDL Generator," in *In Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL07)*, August 2007, pp. 697–701.

ABOUT THE AUTHOR

Georgi Kuzmanov, PhD, Computer Engineering, Faculty EEMCS, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands, G.K.Kuzmanov@TUDelft.NL, http://ce.et.tudelft.nl/.