A Configurable Shared Scratchpad Memory for GPU-like Processors

Alessandro Cilardo, Mirko Gagliardi, Ciro Donnarumma

Abstract During the last years Field Programmable Gate Arrays and Graphics Processing Units have become increasingly important for high-performance computing. In particular, a number of industrial solutions and academic projects are proposing design frameworks based on FPGA-implemented GPU-like compute units. Existing GPU-like core projects provide limited hardware support for shared scratchpad memory and particularly for the problem of bank conflicts, a major source of performance loss with many parallel kernels. In this paper, we present a configurable, GPU-like oriented scratchpad memory with built-in support for bank remapping. The core is fully synthetizable on FPGA with a contained hardware cost. We also validated the presented architecture with a cycle-accurate event-driven emulator written in C++ as well as an RTL simulator tool. Last, we demonstrated the impact of bank remapping and other parameters available with the proposed configurable shared scratchpad memory by evaluating the performance of two real-world parallelized kernels.

1 Introduction

Current trends in high-performance computing (HPC) are increasingly moving towards heterogeneous platforms [25], i.e. systems made of different computational units, like general-purpose CPUs, digital signal processors (DSPs), graphics pro-

Alessandro Cilardo
University of Naples Federico II and Centro Regionale ICT (CeRICT), Naples, Italy, e-mail: acilardo@unina.it

Mirko Gagliardi
University of Naples Federico II and Centro Regionale ICT (CeRICT), Naples, Italy, e-mail: mirko.gagliardi@unina.it

Ciro Donnarumma
University of Naples Federico II, Naples, Italy

cessing units (GPUs), co-processors, and custom acceleration logic, enabling significant benefits in terms of both power and performance.

While HPC covers today disparate applications [27, 7, 6], historically it has never extensively relied on FPGAs, mostly because of the reduced support for floating-point arithmetic. On the other hand, FPGAs and special-purpose hardware in general, e.g. used for arithmetic operations requiring specialized circuit solutions in various areas [11, 10, 15, 14], provide a huge potential for improved power efficiency compared to software-programmable platforms.

Furthermore, while numerous approaches exist for raising somewhat the level of abstraction for hardware design [18, 16], developing an FPGA-based hardware accelerator is still challenging as seen from a software programmer. Consequently, high-performance platforms mostly rely on general-purpose compute units such as CPUs and/or GPUs. However, pure general-purpose hardware is affected by inherently limited power-efficiency, i.e., low GFLOPS-per-Watt. Architectural customization can play here a key role, as it enables unprecedented levels of power-efficiency compared to CPUs/GPUs. This is the essential reason while very recent trends are putting more emphasis on the potential role of FPGAs.

In fact, recent FPGA families, such as the Xilinx Virtex-7 or the Altera Stratix 5, have innovative features, providing significantly reduced power, high speed, lower cost, and reconfigurability [24]. Due to these changes, in the very recent years many innovative companies, including Convey, Maxeler, SRC, Nimbix [25], have introduced FPGA-based heterogeneous platforms used in a large range of HPC applications, e.g. multimedia, bioinformatics, security-related processing, etc. [27, 25], with speedups in the range of 10x to 100x.

This paper explores the adoption of a deeply customizable scratchpad memory system for FPGA-oriented accelerator designs. At the heart of the proposed architecture is a multi-bank parallel access memory system for GPU-like processors. The proposed architecture enables a dynamic bank remapping hardware mechanism, allowing data to be redistributed across banks according to the specific access pattern of the kernel being executed, miminizing the number of conflicts and thereby improving the ultimate performance of the accelerated application. In particular, relying on an advanced configurable crossbar, on a hardware-supported remapping mechanism, and extensive parameterization, the proposed architecture can enable highly parallel accesses matching the potential of current HPC-oriented FPGA technologies. The paper describes the main insights behind by dynamic bank remapping as well as the key role that scratchpad memory might play for hardware-accelerated computing applications.

2 Related work

FPGAs have been used in a vast range of applications [5, 9], although the need for floating point operations has delayed their adoption in HPC. Recently Altera and Xilinx, the two prominent FPGA manufacturers, focused on overcoming FPGA floating-point limitations. In particular, Altera, now part of Intel Corporation, has developed a new floating-point technology (called Fused Datapath) and toolkit (DSP Builder) intended to achieve maximum performance in floating-point design implementing on Altera FPGAs [3]. The other historical problem with FPGAs is programmability. Designing a complex architecture on FPGA, as mentioned above, requires a highly-skilled hardware designer. To overcome this limitation, Altera and Xilinx are bringing the GPU programming model to the FPGA domain. The Altera SDK for OpenCL [1] makes FPGAs accessible to non-expert users. This toolkit allows a user to abstract away the traditional hardware FPGA development flow, effectively creating custom hardware on FPGAs for each instruction being accelerated. Altera claims that this SDK provides much more power-efficient use of the hardware than a traditional CPU or GPU architecture. On other hand, similar to the Altera SDK for OpenCL, Xilinx SDAccel [30], enables traditional CPU and GPU developers to easily migrate their applications to FPGAs while maintaining and reusing their OpenCL, C, and C++ code. Driven by these innovations, FPGAs are becoming increasingly attractive for HPC applications, offering a fine grained parallelism and low power consumption compared to other accelerators.

In line with the above trends, academic and industrial research is focusing on GPU-like paradigms to introduce some form of programmability in FPGA design. In the last years, a few GPU-like projects have appeared. Kingyens and Steffan [23] propose a softcore architecture inspired by graphics processing units (GPUs) mostly oriented to FPGAs. The architecture supports multithreading, vector operations, and can handle up to 256 concurrent threads. Nyami/Nyuzi [12] is a GPU-like RISC architecture inspired by Intel Larrabee. The Nyami HDL code is fully parameterizable and provides a flexible framework for exploring architectural tradeoffs. The Nyami project provides an LLVM-based C/C++ compiler and can be synthesized on FPGA. Guppy [4] (GPU-like cUstomizable Parallel Processor prototYpe) is based on the LEON3 parameterizable soft core. Guppy main feature is to support CUDA-like threads in a lock-step manner to emulate the CUDA execution model. MIAOW [8] (Many-core Integrated Accelerator Of Wisconsin) provides an open-source RTL implementation of the AMD Southern Islands GPGPU ISA. MIAOW's main goal is to be flexible and to support OpenCL-based applications.

Data movement and memory access has traditionally been an important optimization problem and in many classes of systems it may significantly impact perfomance, along with the interconnection subsystem [21, 22]. This also applies for GPU-like processors. Cache hierarchy has been the traditional way to alleviate the memory bottleneck. However, cache coherence mechanisms are complex and not needed in some applications. Many modern parallel architectures utilize fast non-coherent user-managed on-chip memories, called *scratchpad memories* (SPM). Since NVIDIA Fermi family [2], GPUs are equipped with this kind of memories. In NVIDIA architectures this memory can be used to facilitate communication across threads, and it is hence referred to as shared memory. Typically, scratchpad memories are organized in multiple independently-accessible memory banks. Therefore if all memory accesses request data mapped to different banks, they can be handled in parallel. Bank conflicts occur whenever multiple requests are made for data

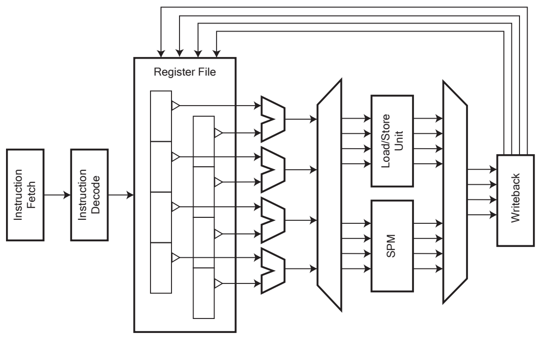

Fig. 1 High-level generic GPU-like core with scratchpad memory.

within the same bank [20]. If N parallel memory accesses request the same bank, the hardware serializes the memory accesses, causing an N-times slowdown [19]. In this context, a dynamic bank remapping mechanism, based on specific kernel access pattern, may help minimize bank conflicts.

Bank conflict reduction has been addressed by several scientific works during the last years. A generalized memory-partitioning (GPM) framework to provide high data throughput of on-chip memories using a polyhedral mode is proposed in [29]. GPM allows intra-bank offset generation in order to reduce bank conflicts. Memory partitioning adopted in these works are cyclic partitioning and block partitioning, as presented in [13]. In [17] the authors address the problem of automated memory partitioning providing the opportunity of customizing the memory architecture based on the application access patterns and the bank mapping problem with a lattice-based approach. While bank conflicts in shared memory is a significant problem, existing GPU-like accelerators [12, 8, 4, 23] lack bank remapping mechanisms to minimize such conflicts.

3 Architecture

SPM interface and operation. Figure 1 depicts a block diagram of the SPM in the context of a GPU-like core architecture. A GPU-like core has a SIMD structure with L multiple lanes. All lanes share the same control unit, hence in each clock cycle they execute the same instruction, although on different data. Every time a new instruction is issued, it is propagated to all execution lanes, each taking the operands from their corresponding portion of a vectorial register file addressed by the instruction. The typical memory instructions provided by a SIMD ISA offer gather and scatter operations. Such operations are respectively vectorial memory load and store memory accesses. If the SIMD core has a single-bank SPM with a single memory port, the previous instructions require al least L clock cycles. This is

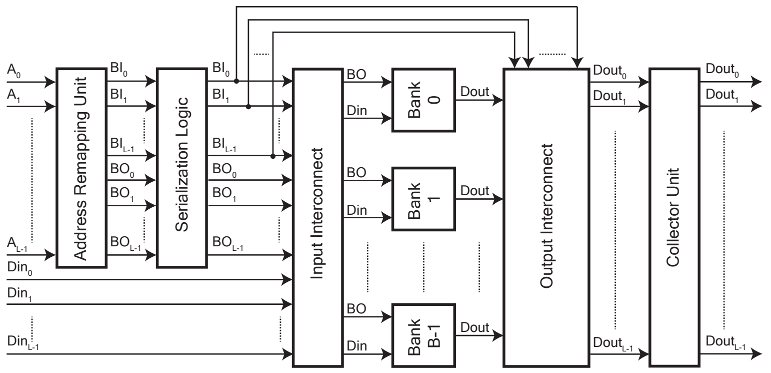

Fig. 2 Architecture overview.

because the L lanes cannot access a single memory port with different addresses in the same clock cycle.

Architecture. Figure 2 shows the internal architecture of the proposed SPM. The SPM takes as input L different addresses to provides support to the scattered memory access. It can be regarded as an FSM with two states: Ready and Busy. In the Ready state, the SPM is ready to accept a new memory request. In the Busy state, the SPM cannot accept any request as it is still processing the previous one, so in this state all input requests will be ignored. The Address Mapping Unit computes in parallel the bank index and the bank offset for each of the L memory addresses coming from the processor lanes. Bank index (BI_i in Figure 2) is the index of the bank to which the address is mapped. Bank offset (BO_i in Figure 2) is the address of the word into the bank. The Address Mapping Unit behaviour can be changed at run time in order to change the relationship between addresses and banks. This is a key feature in that it allows the adoption of the mapping strategy that best suits the executed workload. The Serialization Logic Unit performs the conflict detection and the serialization of the conflicting requests. Whenever an n-way conflict is detected, the Serialization Logic Unit puts the SPM in the busy state and splits the requests into n conflict-free requests issued serially in the next n clock cycles. When the last request is issued, the Serialization Logic Unit put the SPM in the ready state. Notice that for the Serialization Logic Unit, multiple accesses to the same address are not seen as a conflict, as in this occurrence a broadcast mechanism is activated. This broadcast mechanism provides an efficient way to satisfy multiple load requests for the same constant parameters. The Input Interconnect is an interconnection network that steers source data and/or control signals coming from a lane in the GPU-like processor to the destination bank. Because the Input Interconnect follows the Serialization Logic Unit, it only accepts one request per bank. Then, there are the B memory banks providing the required memory elements. Each memory bank receives the bank offset, the source data, and the control signal form the lane that addressed it. Each bank has a single read/write port with a byte-level write enable signal to support instructions with operand sizes smaller than word. Furthermore,

Cyclic mappig

Bank0	Bank1	Bank2	Bank3
0x00	0x04	0x08	0x0c
0x10	0x14	0x18	0x1c
0x20	0x24	0x28	0x2c
0x30	0x34	0x38	0x3c

Block mappig

Bank0	Bank1	Bank2	Bank3
0x00	0x10	0x20	0x30
0x04	0x14	0x24	0x34
0x08	0x18	0x28	0x38
0x0c	0x1c	0x2c	0x3c

Generalized Cyclic mappig

Bank0	Bank1	Bank2	Bank3
0x00	0x04	0x08	0x0c
0x1c	0x10	0x14	0x18
0x28	0x2c	0x20	0x24
0x34	0x38	0x3c	0x30

Fig. 3 The figure shows how addresses are mapped onto the banks. The memory is byte addressable and each word is four byte wide. In the case of generalized cyclic mapping the remapping factor is 1.

each lane controls a bit in an L-bit mask bus that is propagated through the Input Interconnect to the appropriate bank. This bit acts as a bank enable signal. In this way, we can disable some lanes and execute operations on a vector smaller than L elements. The Output Interconnect propagates the loaded data to the lane that requested it. Last, there is a Collector Unit which is a set of L registers that collect the results coming from the serialized requests outputting them as a single vector.

Remapping. As mentioned above, the mapping between addresses and banks can be changed at run time through the Address Mapping Unit. The technical literature presents several mapping strategies, including cyclic and block mapping [13, 29]. These strategies are summarized in Figure 3. Cyclic mapping assigns consecutive words to adjacent banks (Bank $B-1$ is adjacent to Bank 0). Block mapping maps consecutive words onto consecutive lines of the same banks. The block-cycle mapping is a hybrid strategy. With $B = 2^b$ banks, $W = 2^w$ bytes in a word, and $D = 2^d$ words in a single bank, a scratchpad memory address is made of $w+b+d$ bits. Figure 3 shows a cycling remapping, which can be easily obtained by repartitioning the memory address. The Address Mapping Unit described in this work implements a generalization of cyclic mapping, which we call *generalized-cyclic mapping*. By adopting this strategy, many kernels that generate conflicts with cyclic mapping change their pattern by accessing data on the memory diagonal, thereby reducing the number of conflicts.

Implementation details. The proposed configurable scratchpad memory was described in HDL and synthesized for a Xilinx FPGA device. In particular, we used Xilinx Vivado to synthesize the proposed architecture on a Xilinx Virtex7-2000t FPGA (part number xc7v2000tflg1925-2). We built our architecture with a variable number of banks and lanes. Figure 4 reports our SPM occupation in terms of LUTs and Flip-Flops (FFs) for a variable number of banks. On the other hand, Figure 5 reports our SPM occupation in terms of LUTs and Flip-Flops (FFs) for a variable number of lanes. Increasing the number of banks heavily affects LUTs occupation, while the number of lanes mostly affects the FF count.

The proposed SPM has been validated with the Verilator RTL simulator tool [28], which compiles synthesizable Verilog, SystemVerilog, and Synthesis assertions into a C++ behavioural model (called the *verilated* model), effectively converting RTL design validation into C++ program validation. In addition, a cycle accurate event-driven emulator written in C++ was purposely developed for verifying the proposed design. The SPM verilated model and the SPM emulator are integrated in a test-

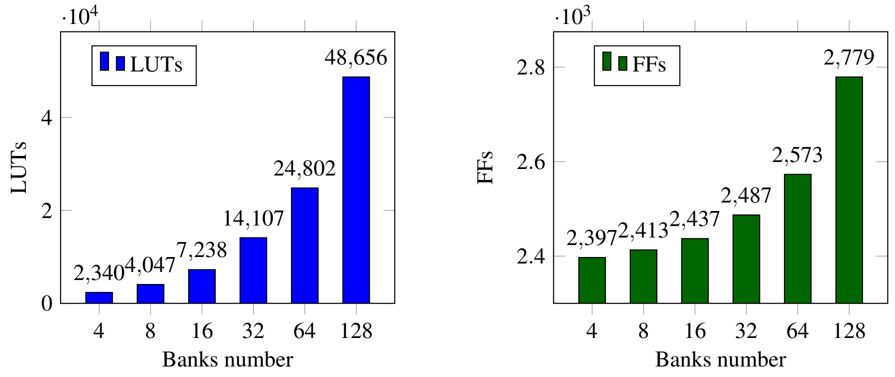

Fig. 4 LUTs and FFs occupation of the FPGA-based SPM design for a variable number of banks

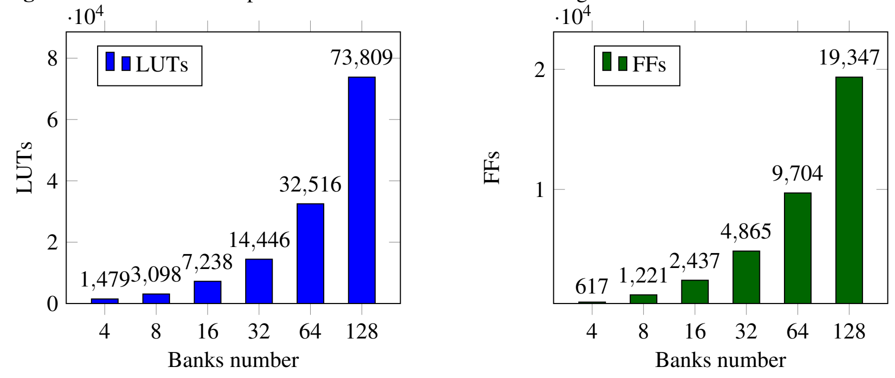

Fig. 5 LUTs and FFs occupation of the FPGA-based SPM design for a variable number of lanes

bench platform, that provides the same inputs to the emulator and the verilated model. The test platform compares the outputs from the two models at every simulation cycle, checking if the verilated model and the emulator generate the same responses. Notice that the Verilator tool supports the HDL code coverage analysis feature, which helped us create a test suite with full coverage of the SystemVerilog code.

Integration in a GPU-like core. The presented SPM, as previously explained, has a variable latency, which may be a potential complication for integration in GPU-like architectures. At issue time, before the operand fetch, the control unit is unaware of the actual latency of a scratchpad memory instruction and can not detect possible Writeback structural hazards. To avoid this problem, a core must support a dynamic on-the-fly structural hazard handler at Writeback stage.

4 Evaluation

The experimental evaluation was essentially meant to demonstrate to which extent the amount of bank conflicts can be reduced by changing the parameters in the proposed configurable scratchpad memory. In particular, to this end, our experiments assess how simultaneous memory accesses, as well as the bank remapping feature may affect the total bank conflict count.

Methodology. We first identified a few kernels that have potentially highly parallel memory accesses and that can benefit from the scratchpad memory support. Many such kernels exist in benchmark suites like PolyBench [26]. Next, we rewrote each of those kernels to increase the kernel memory access parallelism, as our aim was to study how conflicts vary with a variable number of parallel memory requests. We then extracted the access patterns for each kernel and we run it on our scratchpad emulator. The emulator is cycle-accurate, ensuring exact timings for the simulated accesses as the scratchpad memory and the emulator proceed in a lock-step fashion under the same inputs. Last, we collected the emulator response in terms of total bank conflicts for all the memory accesses issued by the kernel, through a counter that is incremented whenever a bank conflict occurs. We repeated this experiment for different remapping functions identified for the specific kernel as well as for a variable number of banks.

Matrix Multiplication. Square matrix-matrix multiplication is a classic bank conflict sensitive kernel. In this benchmark, we evaluated the square matrix access patterns and how the configurable parameters influence the scratchpad bank conflict count.

Listing 1 Matrix Multiplication parameterized on the number of lanes.

```
for (int i = 0; i < DIM; ++i)
  for (int j = 0; j < DIM; ++j)
    for (int k = 0; k < DIM/numLane; ++k)
          for (int lane = 0; lane < numLane; lane++){
                accessA[index][lane] = (i*DIM + k*numLane + lane)*4;
                accessB[index][lane] = ((k*numLane + lane)*DIM + j)*4;
          }
          index++;
```

We rewrote the code so as to maximize the exploitation of the available number of lanes in the target model of GPU-like processor. The inner cycle, shown in Listing 1, calculates which memory address will be accessed by each lane for both matrices. We have a fixed square matrix size $DIM = 128$. The number of hardware lanes is $numLane = [4,8,16,32]$ while the number of banks is $numBanks = [16,32,64,128,256,512,1024]$. The function bank remapping is $(Entry \cdot c + Bank)$ mod $(NUMBANK)$ with $c = [1,2,4,8,16]$.

The total scratchpad memory is kept constant and equal to $BANKnumber \times ENTRYperBank = 2 \times DIM^2$, so that the SPM can store both matrices completely.

Table 1 Matrix Multiplication results.

Lanes	Banks	Remapping factor				
		No Remap	1	2	4	8
4	16	262146	131072	262146	262146	262146
	32	262146	0	0	131072	262146
	64	262146	0	0	0	0
	128	262146	0	0	0	0
	256	131072	0	0	0	0
	512	0	0	0	0	0
	1024	0	0	0	0	0
8	16	183505	131073	183505	183505	183505
	32	183505	0	65536	131073	183505
	64	183505	0	0	0	65536
	128	183505	0	0	0	0
	256	131073	0	0	0	0
	512	65536	0	0	0	0
	1024	0	0	0	0	0
16	16	109230	91756	109230	109230	109230
	32	109230	32768	65538	91756	109230
	64	109230	0	0	32768	65538
	128	109230	0	0	0	0
	256	91756	0	0	0	0
	512	65538	0	0	0	0
	1024	32768	0	0	0	0
32	16	61696	58256	61696	61696	61696
	32	59768	32769	45878	54615	59768
	64	59768	0	16384	32769	45878
	128	59768	0	0	0	16384
	256	54615	0	0	0	0
	512	45878	0	0	0	0
	1024	32769	0	0	0	0

Results in Table 1 show that bank remapping has a greater impact than the other parameters. A remapping coefficient $c = 1$ drastically reduces bank conflicts, even with a limited number of banks, while adding little resource overhead compared to a solution relying on a large number of parallel banks.

Image Mean Filter 5×5. Mean filtering is a simple kernel to implement image smoothing. It is used to reduce noise in images. The filter replaces each pixel value in an image with the mean value of its neighbors, including itself. In our study a 5×5 square kernel is used.

Listing 2 shows our parallelized version of the mean filter. For this kernel we keep a fixed square matrix size $DIM = 128$ and a fixed number of lanes $numLane = 30$. The total scratchpad memory is kept constant and equal to $BANKnumber \times ENTRYperBank = DIM^2$. We evaluated the bank conflicts for a variable number of banks and for two bank remapping functions: no remap and $(Entry \cdot 5 + Bank) \bmod (NUMBANK)$. The results are shown in Table 2. As in the case of the

Listing 2 Image Mean Filter 5x5.

```
#define OFFSET(x, y) (((x)*DIM + y)*4)

for (int i = 2; i < DIM - 3; ++i)
  for (int j = 2; j < DIM - 3; ++j) {
        for (int w1 = -W1; w1 <=W1; w1++ ){
          for (int w2 = -W2; w2 <= W2; w2++){
                a = baseA.getAddress() + OFFSET(i+w1, j+w2);
                l = (w1 + 2)*5 + (w2 + 2);
                accessA[index][l] = a;
          }
        }
        index++;
  }
```

matrix multiplication kernel, the remapping function has the largest impact on the bank conflict count.

Table 2 Image Mean Filter 5x5.

Banks	No Remap	Remap
16	7565	1722
32	7565	0
64	7565	0
128	7565	0
256	0	0
512	0	0
1024	0	0

5 Conclusion

In this work we presented a configurable GPU-like oriented scratchpad memory fully synthesizable on FPGAs. Various architectural aspects like the number of banks, the number of lanes, the bank remapping function, and the size of the total memory are parameterized. Reconfigurability helped explore architectural choices and assess their impact. We described the SPM design in HDL and extensively validated it. We also developed a software cycle accurate and event-driven emulator of our SPM component to support the experimental evaluation with real code. Through two case studies, a matrix multiplication and a 5×5 image mean filter, we showed the performance implications with different configurations and demonstrated the benefits of using a dedicated hardware bank remapping function over other architectural parameters. As a long-term goal of this research, we aim to integrate our

SPM architecture in an open source GPU-like core, enabling it to take full advantage of the underlying reconfigurable hardware technologies.

Acknowledgments. This work is supported by the European Commission in the framework of the H2020-FETHPC-2014 project n. 671668 - MANGO: exploring Manycore Architectures for Next-GeneratiOn HPC systems.

References

1. The Altera SDK for open computing language (OpenCL). https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html
2. Nvidia's next generation cuda compute architecture. NVidia, Santa Clara, Calif, USA (2009)
3. An independent analysis of Altera's FPGA floating-point DSP design flow. Berkeley Design Technology, Inc (2011)
4. Al-Dujaili, A., Deragisch, F., Hagiescu, A., Wong, W.F.: Guppy: A GPU-like soft-core processor. In: Field-Programmable Technology (FPT), 2012 International Conference on, pp. 57–60 (2012)
5. Amato, F., Barbareschi, M., Casola, V., Mazzeo, A.: An FPGA-based smart classifier for decision support systems. Studies in Computational Intelligence **511**, 289–299 (2014)
6. Amato, F., Fasolino, A., Mazzeo, A., Moscato, V., Picariello, A., Romano, S., Tramontana, P.: Ensuring semantic interoperability for e-health applications. In: Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2011, pp. 315–320 (2011)
7. Amato, F., Mazzeo, A., Penta, A., Picariello, A.: Building RDF ontologies from semi-structured legal documents. pp. 997–1002 (2008)
8. Balasubramanian, R., Gangadhar, V., Guo, Z., Ho, C.H., Joseph, C., Menon, J., Drumond, M.P., Paul, R., Prasad, S., Valathol, P., Sankaralingam, K.: Enabling GPGPU low-level hardware explorations with MIAOW: An open-source RTL implementation of a GPGPU. ACM Trans. Archit. Code Optim. **12**(2), 21:21:1–21:21:25 (2015)
9. Barbareschi, M., Del Prete, S., Gargiulo, F., Mazzeo, A., Sansone, C.: Decision tree-based multiple classifier systems: An FPGA perspective. In: International Workshop on Multiple Classifier Systems, pp. 194–205. Springer (2015)
10. Barbareschi, M., Iannucci, F., Mazzeo, A.: Automatic design space exploration of approximate algorithms for big data applications. In: 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), pp. 40–45. IEEE (2016)
11. Barbareschi, M., Iannucci, F., Mazzeo, A.: An extendible design exploration tool for supporting approximate computing techniques. In: 2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1–6. IEEE (2016)
12. Bush, J., Dexter, P., Miller, T.N.: Nyami: a synthesizable GPU architectural model for general-purpose and graphics-specific workloads. In: Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on, pp. 173–182 (2015)
13. Chatterjee, S., et al.: Generating local addresses and communication sets for data-parallel programs. SIGPLAN Not. **28**(7), 149–158 (1993)
14. Cilardo, A.: Exploring the potential of threshold logic for cryptography-related operations. IEEE Transactions on Computers **60**(4), 452–462 (2011)
15. Cilardo, A., De Caro, D., Petra, N., Caserta, F., Mazzocca, N., Napoli, E., Strollo, A.: High speed speculative multipliers based on speculative carry-save tree. IEEE Transactions on Circuits and Systems I: Regular Papers **61**(12), 3426–3435 (2014)
16. Cilardo, A., Durante, P., Lofiego, C., Mazzeo, A.: Early prediction of hardware complexity in HLL-to-HDL translation. pp. 483–488 (2010)

17. Cilardo, A., Gallo, L.: Improving multibank memory access parallelism with lattice-based partitioning. ACM Transactions on Architecture and Code Optimization (TACO) **11**(4), 45 (2015)
18. Cilardo, A., Gallo, L., Mazzeo, A., Mazzocca, N.: Efficient and scalable OpenMP-based system-level design. pp. 988–991 (2013)
19. Coon, B., et al.: Shared memory with parallel access and access conflict resolution mechanism. U.S. Patent No. 8,108,625 (2012)
20. Farber, R.: CUDA application design and development. Elsevier (2011)
21. Fusella, E., Cilardo, A.: H^2ONoC: A hybrid optical-electronic NoC based on hybrid topology. IEEE Transactions on Very Large Scale Integration (VLSI) Systems (2016)
22. Fusella, E., Cilardo, A.: Minimizing power loss in optical networks-on-chip through application-specific mapping. Microprocessors and Microsystems (2016)
23. Kingyens, J., Steffan, J.: The potential for a GPU-like overlay architecture for FPGAs. International Journal of Reconfigurable Computing (2011)
24. Kuon, I., Rose, J.: Measuring the gap between FPGAs and ASICs. In: Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, FPGA '06, pp. 21–30. ACM, New York, NY, USA (2006)
25. Paranjape, K., Hebert, S., Masson, B.: Heterogeneous computing in the cloud: Crunching big data and democratizing HPC access for the life sciences. Intel Corporation (2010)
26. Pouchet, L.N.: Polybench: The polyhedral benchmark suite. http://www. cs. ucla. edu/pouchet/software/polybench (2012)
27. Sarkar, S., et al.: Hardware accelerators for biocomputing: A survey. In: Proceedings of 2010 IEEE International Symposium on Circuits and Systems (2010)
28. Snyder, W., Wasson, P., Galbi, D.: Verilator (2007)
29. Wang, Y., Li, P., Cong, J.: Theory and algorithm for generalized memory partitioning in high-level synthesis. In: Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays, FPGA '14, pp. 199–208. ACM, New York, NY, USA (2014)
30. Wirbel, L.: Xilinx SDAccel: a unified development environment for tomorrow's data center. The Linley Group Inc (2014)