The HtComp research project: an overview

(Invited Paper)

Alessandro Cilardo
DIETI - University of Naples Federico II
Naples, Italy, Email: acilardo@unina.it

***Abstract*—This contribution reviews the main results of the HtComp project, a two-year research programme aiming at facilitating the integration of FPGA-based accelerators into general-purpose computing. The project covered the automated generation of HDL code from parallel applications written in traditional high-level software languages, as well as the customization of the processing, memory, and on-chip interconnect subsystems tailored on the application requirements. The ultimate outcome of the research was the introduction of methods and tools allowing software developers, particularly from the HPC domain, to access hardware-accelerated platforms incurring significantly reduced design complexity and overheads.**

I. Introduction

During the very recent years many trends have clearly indicated that reconfigurable hardware, i.e. FPGA devices, may potentially provide a profitable acceleration solution for datacenters, cloud servers, and high-performance computing [1], [2], particularly for special classes of applications like multimedia, networking, bioinformatics, security-related processing [3], [1], [4], [5]. However, programming such next-generation machines is extremely difficult as it needs architecture-specific code and specialized skills, typically requiring developers to master low-level hardware description languages such as VHDL or Verilog. We collectively indicate these challenges as the *programmability wall*. By failing to tackle this wall, we will miss the opportunity to completely exploit the computational power offered by future heterogeneous platforms, as we will restrict it to a limited élite of highly-skilled parallel programmers/hardware designers, excluding a vast base of potential users.

This paper reviews the main outcomes of a research project entitled **HtComp**, which explores methodologies and tools allowing the automated definition of FPGA-based accelerators from high-level software applications. The project introduced new ideas and research results revolving around the generation of HDL code from parallel programs as well as the automated customization of FPGA-based Multi-Processor Systems-on-Chip (MPSoCs) in terms of processing units, memory subsystems, and on-chip interconnects on a per-application basis.

The paper is organized as follows: Section II describes the main issues addressed at the level of programming models. Section III presents the main results related to the customized architecture generated through the **HtComp** flow. Section IV concludes the paper by outlining the future developments of the activity.

II. Programming models

HtComp targeted the well-known OpenMP [6] shared memory programming model as a design entry solution [7], [8] supporting high-level software programmers through the generation of customized hardware-accelerated systems. Targeting C code with OpenMP parallel extensions, the toolchain enables the reuse of a large body of existing code from the parallel and high-performance computing domain. The high-level programming solution was "connected" with a *high-level synthesis* (HLS) [9] toolflow used to automatically generate HDL descriptions from C-like code. In particular, the prototype **HtComp** toolchain was validated in conjunction with the Impulse CoDeveloper HLS tool as well as with the Xilinx VivadoHLS environment.

The project identified a few key issues involved in transforming high-level OpenMP parallel code to FPGA-based on-chip heterogeneous systems. We developed some architectural mechanisms for supporting OpenMP directives in an efficient and distributed way across the system. We introduced an additional step in the toolchain applying HLS-related code optimizations, i.e. code transformations aimed at improving the quality of results of the underlying HLS tool in terms of performance and number of hardware resources. An overview of the compilation flow is presented in Figure 1 [7], [8].

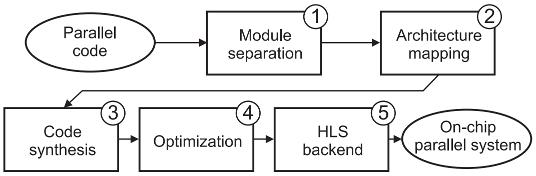


Fig. 1. Compilation flow.

In the reference architectural model of the generated system, the parallel OpenMP threads at the software level are mapped to either instruction processors or hardware cores generated by means of high-level synthesis, in addition to hand-coded components used for specific applications [13]. A few special components are used for particular tasks, including run-time support and system management, timers, and memory-mapped *atomic registers*, providing a hardware facility for thread synchronization.

The OpenMP main thread is mapped to a software subsystem, while the other threads can be mapped to either hardware or software subsystems. Currently, the flow only supports one thread per subsystem, but as a natural extension we envision a scenario where several OpenMP threads share the same processor using a multithreaded OS kernel. Notice that

Invited presentation for the 3rd International Workshop on Cloud and Distributed System Applications (CADSA-15) the project also explored alternative methodologies for high-level design entry and application mapping [14]. Hardware subsystems generated by HLS are memory mapped. They can be addressed using the slave port in a globally shared address space and they are also provided with DMA capabilities. Each hardware subsystem is provided with a thread ID held internally throughout the execution of an OpenMP `parallel` construct.

A number of techniques were implemented to support OpenMP constructs and programming styles, for example concerning the memory mapping of private and shared OpenMP variables exploiting either optimized on-chip or off-chip memory; the support for the `schedule(dynamic)` clause in `for` loops, the efficient implementation of barriers, used in a number of OpenMP constructs, the sharing of hardware blocks corresponding to function calls, loop interchanging, etc [7].

III. Architecture-level techniques

A. *Heterogeneous acceleration cores*

While heterogeneous accelerators like GPUs have so far relied on memory spaces separated from host devices, recent trends indicate a shift towards shared memory models [10], [11], with a potential impact on both programming- and architecture-level aspects. An essential objective of **HtComp** was to define the architecture of software-programmable accelerators, in addition to pure-hardware units generated through a HSL toolchain, and match the above trends by supporting *GPU-like* heterogeneous computing. In particular, the project envisioned a scenario where the architecture offers pre-defined GPU-like datapaths that can still be configured in hardware, allowing design-time deep customization of the accelerators driven by the application computing demand, yet preserving the possibility of software programming under a standard architectural model. Although the project did not cover these aspects, this standard model is likely to be closely related to the HSA virtual machine [10] or the SPIR intermediate language [12], [11], which provide some form of decoupling between heterogeneous applications and the details of the underlying platform, and let programmers generate their intermediate code from a variety of high-level entry points, including OpenMP, then converted to the native heterogeneous architecture by a (possibly runtime) *finalizer*. In a perspective scenario, thus, the **HtComp** GPU-like units will be accompanied by a dedicated software layer, acting as a dynamic finalizer specific of the **HtComp** architecture.

In its current state, the project has introduced the low-level GPU-like datapaths to be used as the basic building blocks for heterogeneous "Kernel Agents". The customizable cores form a parallel, floating-point intensive compute fabric that can be tailored on the application needs. The core architecture is in fact fully configurable, letting the developer set parameters like the number of floating-point ALUs in an accelerator core, the number of register banks, the memory capacity of a single module, etc.

The basic building block of the architecture is inspired by the architectural paradigm typical of GPU computing. The **HtComp** GPU-like core is in fact RISC-like, fully pipelined, oriented to SIMD floating-point operations and fully configurable as required by the target application. The core architecture, not described here in detail, is divided in four familiar stages, i.e. the Instruction Fetch/Decode, the Execution, the Memory, and the Write Back. Here, the EX stage contains P floating-point ALUs, configurable by the designer in terms of number of instances and supported operators, and compliant with the IEEE 754-2008 standard. The Fetch/Decode stage is in charge of handling N (user defined) Work-Group instruction words from different *User Agent Queues*, ensuring proper structural hazard checks, while data hazards are demanded to the software layer for preserving hardware scalability. The stage is also in charge of allocating dynamically one or more register banks to a Work-Group on request, translating the "virtual" registers used by a Work-Group to the real registers they are assigned to, as well as handling the different operation latencies in the floating-point ALUs. More details about the implemented unit are provided in [16].

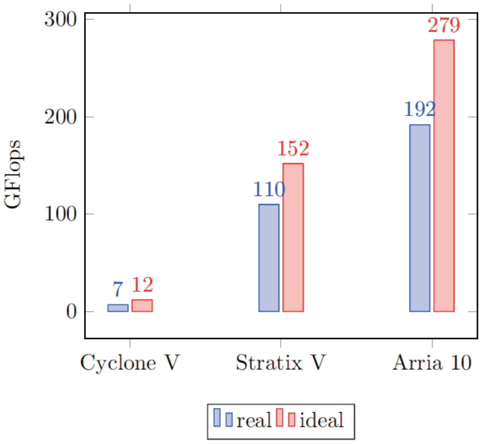
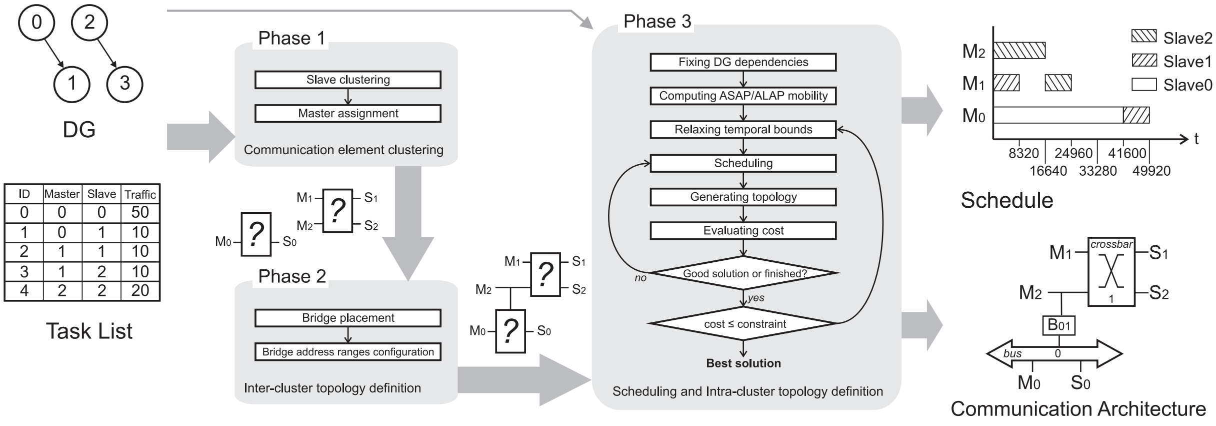


Fig. 2. Real GFLOPS vs ideal GFLOPS

To evaluate the hardware cost of our solution, we described the customizable GPU-like core in VHDL and implemented it on an Altera Cyclone V SoC 5CSEMA5F31C6 device mounted on a DE1-SoC evaluation board. Although we do not provide the full details here, we highlight a few results showing the potential of the approach. Table I summarizes the area results as the number of FP cores is increased. While adding a new core involves an overhead due to the registers needed by the core itself, the control part has a limited impact and increasing the number of FPU cores does not significantly affects its complexity. As a consequence, by increasing the FPU core number the design resources increase almost linearly. We also evaluated the proposed architecture on other Altera FPGAs, namely a Stratix V and an Arria 10 device, in addition to the Cyclone V device used for the above experiments. Figure 2 shows a comparison between the performance achieved and an upper-bound to the floating-point performance [15] for the different technologies. As highlighted

Cores	Alms	DSP	Mem
2	2244	2	8Kb
4	4170	4	16Kb
8	7927	8	32Kb
16	15597	16	64Kb
32	31040	32	128Kb

TABLE I. Design resources used



ID	Master	Slave	Traffic
0	0	0	50
1	0	1	10
2	1	1	10
3	1	2	10
4	2	2	20

Task List

Fig. 3. Proposed interconnect synthesis flow

by the figure, the proposed solution gets reasonably close to the performance upper-bound.

B. Customized memory subsystem

As a different branch followed by **HtComp**, the project investigated the opportunity of deriving automatically a customized memory subsystem from the application memory access patterns. In fact, reconfigurable hardware typically provide many fine-grained memory blocks which can be combined together to form an ad-hoc memory architecture, highly configurable in terms of number of banks and independent ports, width, depth, and so forth. Based on this observation, the project aimed to introduce suitable memory partitioning strategies to take full advantage of the memory subsystem. The basic assumption is that a customized hardware unit, possibly generated through HLS, has multiple input/output *ports* accessing different data concurrently. The structure of the memory subsystem and the way data are distributed across the different banks affects the access patterns issued by the application-specific unit and, consequently, it directly impacts the number of conflicts arising whenever concurrent accesses from different ports collide on the same physical bank, causing serialized operations and hence degraded time and energy efficiency.

The essential idea pursued by **HtComp** is to pre-process the high-level source code (specifically C/C++ code), reorganizing the memory layout of the data structure it handles in such a way that the HLS tool will generate a custom hardware unit with optimized access patterns which minimize the conflict counts. The works target static code under the contraints of polyhedral analysis [17], i.e. loop nests where loop bounds and memory references are affine functions of the loop iterators, which is common in a wide range of HPC and scientific program kernels. In particular, we proposed a technique based on the Z-polyhedral model for program analysis, relying on lattice-based data partitioning. In essence, we regard the data structure, e.g. an array in the C/C++ code, as a portion of the n-dimensional $\mathbb{Z}^n$ space (where n is the dimensionality of the array). Then, we identify an *integer lattice* in $\mathbb{Z}^n$, which can informally be regarded as a set of equally spaced points in $\mathbb{Z}^n$. It can be shown that a lattice, along with a finite number of translated affine lattices, cover the whole set $\mathbb{Z}^n$, i.e., the whole memory space which includes the array to be partitioned. Each translated lattice identifies the points of $\mathbb{Z}^n$, i.e. the memory locations of the original array, that will be mapped to *a distinct physical bank*. We developed a procedure to enumerate all possible distinct lattice-based partitioning solutions for a given number of physical banks, evaluate them, and pick one of those solutions that incur the minimum number of conflicts [18].

Lattice-based partitioning proved to offer the largest search space for partitioning choices minimizing access conflicts and the related performance/energy penalty incurred by data movement. In fact, it includes other solutions recently proposed in the literature, like hyperplane-based partitioning, as special cases [18], [19].

C. Customized interconnect architecture

As implied by the previous sections, the interconnection infrastructure is a critical part of the generated system. In fact, highly parallel systems, like those targeted by the **HtComp** methodology, must rely on a single optimized crossbar [20], [21] or cascaded crossbars [22], which ensures improved scalability for larger systems. Constraining the interconnect architecture to using only crossbars, as opposed to shared buses, might however results in higher area costs, particularly for FPGA-based implementations [23], so the **HtComp** toolflow relied on heterogeneous interconnects encompassing both shared buses and –possibly cascaded– crossbars. In particular, similar to the customization of the memory architecture, the **HtComp** methodology yields a communication topology tailored on the application communication patterns. Interestingly, unlike several previous works on automated interconnect synthesis, **HtComp** takes into account *dependency constraints* between the communication tasks of the application. Since such dependencies limit the inherent degree of parallelism across the communication tasks, taking them into consideration allows us to instantiate only the interconnect resources that are actually needed, avoiding underutilized hardware and hence wasted area on the chip. Driven by this observation, the proposed methodology addresses jointly the communication scheduling and the interconnect synthesis optimization.

The methodology takes as input a communication Task List (TL) containing, for each task, the master and the slave involved and the amount of bytes to be transferred, along with a Dependency Graph (DG) describing possible inter-task precedence relationships, i.e. dependency constraints. Under a given area constraint, the methodology finds a synthesizable topology specification based on a heterogeneous bus/crossbar architecture minimizing the target cost function, together with a compatible minimum-latency communication task schedule. The proposed topology synthesis flow, shown in Figure 3, consists of three phases, described below. The first step (Phase 1) is the clustering of communicating elements in local domains. We attempt to place the nodes that communicate more frequently closer to each other, minimizing the traffic between communicating elements and matching the localized traffic patterns induced by a given application. In Phase 2 the clusters are connected in order to make all inter-cluster communications feasible by means of bridges. By properly setting the mapping of the address spaces in each bridge, furthermore, multiple physical paths between different domains can be realized. Finally, we need to decide how single clusters will be implemented (Phase 3). This step is performed jointly with the communication scheduling: an iterative procedure finds an optimal communication tasks schedule (in terms of latency) and a global topology containing enough resources to execute the identified schedule.

The methodology was demonstrated on an FPGA platform. This branch of the activity relied on an FPGA board equipped with a device of the Xilinx Zynq™-7000 family, embedding reconfigurable hardware and a hard-core ARM processor. The custom interconnect and the communicating elements are mapped onto the reconfigurable fabric. The communication architecture synthesis flow uses the Xilinx AXI components compliant with the AMBA® AXI version 4 specification from ARM. We tested our method for six synthetic benchmarks as well as a Canny edge detection algorithm. The results showed that the highly application-driven interconnect synthesis process ensures increased levels of resource utilization and energy efficieny [24].

IV. Developments and Conclusions

HtComp explored new methodologies and tools that contributed to raising the level of abstraction for today's acceleration platforms, exposing them in forms that are familiar to general programmers. The main contributions brought by the project included the automated generation of customized hardware from high-level parallel applications, particularly OpenMP code, as well as the application-driven optimization of the memory architecture and the on-chip interconnects. Raising the level of abstraction as seen by the programmers, the above innovations can create an easily accessible entry-point to the development of parallel applications based on FPGA hardware accelerators paired with standard platforms like multi-core CPUs and GPUs.

The funding programme supporting **HtComp** (*STAR* call) required beneficiaries to scale up the size of the project by competing for large-scale funding at the European level. This objective was ultimately achieved, as various ideas and preliminary results developed by **HtComp** contributed to shaping a Horizon 2020 Future Emerging Technologies (FET) HPC project proposal, successfully selected for funding by the European Community. The project is called *MANGO: exploring Manycore Architectures for Next-GeneratiOn HPC systems* and started October 2015. It is a large-scale European project exploring heterogeneous manycore architectures for HPC with a total budget of 5.8 Million euro, based on the cooperation of the Polytechnical University of Valencia, University of Naples Federico II/CeRICT, Politecnico di Milano, University of Zagreb, Philips Medical, Thales, EPFL, PRO-DESIGN, and Eaton. Its essential aim is to achieve extreme resource efficiency in future QoS-sensitive HPC settings through the definition of high-performance, power-efficient, heterogeneous architectures with native mechanisms for isolation and quality-of-service, along with an innovative two-phase passive cooling system. The MANGO investigation will involve many inter-related mechanisms at various architectural levels, also including heterogeneous computing cores, memory architectures, interconnects, programming models, and run-time resource management. While the **HtComp** project has almost come to its end as of writing this paper, its future developments have been embodied by the ambitious research objectives set by MANGO at the architecure level.

Acknowledgments

HtComp is a start-up project co-funded by the private Compagnia San Paolo foundation and the University of Naples Federico II in the context of the *Sostegno Territoriale alle Attività di Ricerca (STAR)* call.

The presentation of this work is also 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] K. Paranjape, S. Hebert, and B. Masson, "Heterogeneous computing in the cloud: Crunching big data and democratizing HPC access for the life sciences," Intel Corporation, Tech. Rep., 2010.

[2] A. Putnam and other, "A reconfigurable fabric for accelerating large-scale datacenter services," in *41st Annual International Symposium on Computer Architecture (ISCA)*, June 2014. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=212001

[3] S. Sarkar, T. Majumder, A. Kalyanaraman, and P. Pande, "Hardware accelerators for biocomputing: A survey," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, May 2010, pp. 3789–3792.

[4] A. Cilardo and N. Mazzocca, "Exploiting vulnerabilities in cryptographic hash functions based on reconfigurable hardware," *Information Forensics and Security, IEEE Transactions on*, vol. 8, no. 5, pp. 810–820, May 2013.

[5] A. Cilardo, "New techniques and tools for application-dependent testing of FPGA-based components," *Industrial Informatics, IEEE Transactions on*, vol. 11, no. 1, pp. 94–103, Feb 2015.

[6] OpenMP Architecture Review Board. (2011) OpenMP application program interface, v3.1. [Online]. Available: www.openmp.org

[7] A. Cilardo, L. Gallo, and N. Mazzocca, "Design space exploration for high-level synthesis of multi-threaded applications," *Journal of Systems Architecture*, vol. 59, no. 10, pp. 1171–1183, 2013.

[8] A. Cilardo, L. Gallo, A. Mazzeo, and N. Mazzocca, "Efficient and scalable OpenMP-based system-level design," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 988–991.

[9] P. Coussy and A. Morawiec, *High-Level Synthesis from Algorithm to Digital Circuit*. Springer, 2008.

[10] H. Foundation. (2015) HSA programmer's reference manual: HSAIL virtual ISA and programming model compiler writer, and object format. [Online]. Available: http://www.hsafoundation.com/

[11] L. Howes and A. Munshi, “The OpenCL specification version: 2.0 revision 26,” Khronos Group, Tech. Rep., 2014.

[12] “Standard portable intermediate representation (SPIR) v2.0,” Khronos Group, Tech. Rep., 2015.

[13] A. Cilardo, “Efficient bit-parallel GF(2^m) multiplier for a large class of irreducible pentanomials,” *Computers, IEEE Transactions on*, vol. 58, no. 7, pp. 1001–1008, July 2009.

[14] A. Cilardo, D. Socci, and N. Mazzocca, “ASP-based optimized mapping in a Simulink-to-MPSoC design flow,” *Journal of Systems Architecture*, vol. 60, no. 1, pp. 108 – 118, 2014.

[15] “Designing and using FPGAs for double-precision floating-point math - white paper,” Altera, Tech. Rep., 2007.

[16] A. Cilardo, J. Flich, M. Gagliardi, and R. Gavila, “Customizable heterogeneous acceleration for tomorrow’s high-performance computing,” in *Proceedings of the 12th IEEE International Conference on Embedded Software and Systems (ICESS)*, August 2015.

[17] C. Bastoul, “Code generation in the polyhedral model is easier than you think,” in *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 7–16.

[18] A. Cilardo and L. Gallo, “Improving multibank memory access parallelism with lattice-based partitioning,” *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 4, pp. 45:1–45:25, Jan. 2015.

[19] ——, “Interplay of loop unrolling and multidimensional memory partitioning in HLS,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, March 2015, pp. 163–168.

[20] S. Pasricha, N. Dutt, and M. Ben-Romdhane, “Constraint-driven bus matrix synthesis for MPSoC,” in *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*. IEEE Press, 2006, pp. 30–35.

[21] S. Murali, L. Benini, and G. De Micheli, “An application-specific design methodology for on-chip crossbar generation,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 7, pp. 1283–1296, 2007.

[22] M. Jun, D. Woo, and E.-Y. Chung, “Partial connection-aware topology synthesis for on-chip cascaded crossbar network,” *Computers, IEEE Transactions on*, vol. 61, no. 1, pp. 73–86, 2012.

[23] A. Cilardo, E. Fusella, L. Gallo, and A. Mazzeo, “Automated synthesis of FPGA-based heterogeneous interconnect topologies,” in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*. IEEE, 2013, pp. 1–8.

[24] ——, “Exploiting concurrency for the automated synthesis of MPSoC interconnects,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 3, p. 57, 2015.