Scalable Embedded Systems: Towards the Convergence of High-Performance and Embedded Computing

Roberto Giorgi
University of Siena
Siena, Italy
Email: giorgi@dii.unisi.it

***Abstract*—Embedded System toolchains are highly customized for a specific System-on-Chip (SoC). When the application needs more performance, the designer is typically forced to adopt a new SoC and possibly another toolchain. The rationale for not scaling performance by using, e.g., two SoCs, is that maintining most of the operations on-chip may allow for higher energy efficiency.**

We are exploring the feasibility and trade-offs of designing and manufacturing a new Single Board Computer (SBC) that could serve flexibly for a number of current and future applications, by allowing scalability through clusters of SBCs while keeping the same programming model for the SBC.

This board is based on FPGAs and embedded processors, and its key points are: i) a fast custom interconnect for board-to-board communication and ii) an easily programmable environment which would allow both the off-loading of code into accelerators (either soft-IP blocks or hard-IP blocks) and, at the same time, the distribution of computation across boards.

A key challenge to successfully deploying this paradigm is to properly distribute the threads across several boards without the explicit intervention of the programmer.

In this paper we describe how to dynamically and efficiently distribute the computational threads in symbiosis with an appropriate memory model to allow the system scalability, so that we can double the performance by simply connecting two boards without i) changing the basic hardware components (e.g., to a different System-On-Chip) and ii) changing the programming model to follow the vendor specific toolchain. Our approach is to reduce data movement across boards. Our initial experiments have confirmed the feasibility of our approach.

***Index Terms*—Cyber-Physical Systems; Reconfigurable Systems; Cluster Programming; FPGA Programming; Distributed Shared Memory; Programming Model; Performance Evaluation.**

I. Introduction

Traditionally, High-Performance Computing (HPC) and Embedded Computing have quite different objectives in terms of design, programmability and energy efficiency. However, due to the ever increasing demand for performance, in turn led by the need to support more powerful applications - such as, e.g. smart video-surveillance and more in general mobile and Cyber-Physical System applications - those objectives tend to adopt similar solutions [1]. ARM which traditionally manufactured embedded processors is now entering the micro-server arena and Intel which traditionally manufactured higher performance products is now introducing their HPC processors, e.g. Xeon E3-1500M, into the mobile market.

In the context of the project AXIOM [2], [3], [4] we are exploring the feasibility and trade-offs in designing and manufacturing a new Single Board Computer that could serve flexibly for a number of current and future applications. This board is based on FPGAs and embedded processors, e.g. Zynq [5], [6] and its key points are: i) a fast custom interconnect for board-to-board communication and ii) an easy programmable environment which could allow us both to off-load code into accelerators (either soft-IP blocks or hard-IP blocks) and, at the same time, to distribute the computation across boards.

AXIOM is an open-hardware open-source initiative. Therefore the board is targeting performance over cost through the use of off-the-shelf components, it is governed by a Linux operating system and the programming model is OmpSs [7], with the aim to bring enough simplicity for the programmer.

Other projects such as Montblanc, P-Socrates, SHAPES investigated the convergence of HPC and Embedded Computing. Nevertheless, efficiently scaling the performance of a computing system while maintaining easy programmability is still an open problem [8].

In order to have a better control of the programmable hardware toolchains need to be complemented with appropriate high level synthesis tools [9], [10].

Multi-core SoCs (MPSoCs) are now the norm, but thread management is source of lot of inefficencies. Their management has to consider not only the order of execution and the time quantum per thread, but also the implications of allocating a thread on a given core. This aspect becomes critical as the whole architecture grows in complexity, configuring distributed computational resources, memory hierarchies, interconnects. Moreover, since energy-efficiency has emerged as a key requirement, thread management goals cannot be limited only to the maximization of performance.

In this paper we present some initial results regarding the

This work has been supported by the European Commission under the AXIOM H2020 project (id. 645496) and HiPEAC network of excellence (id. 287759).

distribution of threads *across multiple SoCs* (or SBCs): the results demonstrate the feasibility of achieving scalability in embedded systems by the combination of elements such as FPGAs and concepts derived from the HPC domain, while maintining a simpler programming model like OmpSs.

The contribution of this work are:

i) exploring the concept of "scalable embedded system", while showing some initial result;
ii) indicating a way to achieve such scalability by supporting special threads called Data-Flow Threads (DF-Threads);
iii) iillustrating how this concepts are integrated in the AXIOM project, which is focused to build a scalable Single Board Computer.

The rest of the paper is organized as follows: in Section II we highlight some related work; in Section III we explain why supporting threads is becoming more and more important; in Section IV we illustrate the pillars of the AXIOM platform; in Sections V and VI we illustrate some experiments and finally we conclude the paper.

II. Related Work

FPGAs are largely employed in prototyping Embedded Systems and are becoming a key component for accelerating kernels in the HPC domain, as they promise a better energy efficiency [11].

Concerning reconfigurable hardware platforms based on FPGA, several works have proposed solutions to address the problem of dynamic allocation of tasks to general-purpose multi-core processors [12], or reconfigurable logic (hardware kernels) [13]. However, such approaches have been successfully explored only on single and multi-core super-scalar architectures, so far.

In recent systems, the large adoption of many-cores accelerators (i.e., GPUs) has worsen the problem, introducing synchronization issues also among different types of computing elements. Research activity has been always active in this specific context, providing solutions that attempt to efficiently solve the synchronization problem. A more general scheduling unit helps distribute the workload efficiently, not only among CPU cores, but also on the specific accelerators.

Data-Flow Threads (DF-Threads [14]) offer a simple and effective solution to address the need of reducing data transfers by moving data where it is requested for a certain computation, expressed by the DF-Thread. While most of computations can be performed in a producer-consumer modality, there is the specific need of accessing mutable shared data. The memory model offered by DF-Threads [14] is encompassing Transactional Memory [15], which is currently also adopted by manufacturers such as IBM and Intel.

Data-flow execution models had been studied widely studied [16] as they provide a simple an elegant way to efficiently move data from one computational thread to another one [17], [18].

In the context of the TERAFLUX project [19], [20], [21] such data-flow model had been extended to multiple nodes executing seamlessly thanks to the support of an appropriate memory model [22], [14]. In such memory model a combination of consumer-producer patterns [23], [24] and transactional memory [25], [26] permits a novel combination of data-flow concepts and transactions in order to address the consistency across nodes, where each node is assumed to be cache-coherent, i.e., like in a classical multi-core. Data-flow models also allows the system to take care in a distributed way of faults that may affect a node [27], [28]: in essence a data-flow thread may be re-executed without side effects since we retain its input before scheduling anything else on the same core.

The AXIOM project is the context where this research is currently developed: other recent papers describe more in detail the hardware framework [2] and the software layers [3].

III. Thread management

Using a general paradigm to manage threads can lead to good performance, such as in the case of P-threads, Cilk, OpenMP. Howevwer, these models suffer performance penalties when synchronization and distribution of data is not managed properly [7]. By re-organzing the execution is such a way the threads follow more closely the data flow of the program, such as with DF-Thrads, better scalability can be achievied [14].

DF-Threads are best implemented in hardware through the use of a Distributed Thread Scheduler [29] (DTS). The DTS tries to solve the following challenges:

- at the system level, all the available resources and the healthiness of the whole system must be considered in a distributed fashion: if a part breaks the remaining of the system should continue to work [28];
- at low-level, the fine-grain threads coming from the adoption of the data-flow execution model must be distributed across the computing elements (CPUs, FPGAs).

This means to understand at run-time what is the best resource assignment (scheduling/mapping on CPU or reconfigurable HW) to a task (or thread), according to multiple goals (e.g., performance/QoS, power consumption minimization, thermal hotspots). The policies should operate effectively both in a single application and a mixed workload scenario. The scheduler can be further extended to enable it distributing fine-grain threads across the different boards or MPSoCs.

In order to reduce the thread management overhead, the DTS needs to be accelerated in hardware, by mapping its structure into the FPGA. The hardware thread support is represented in Figure 1 by the eXtended Shared Memory (XSM) block. Standard high-speed and low-latency interconnections (e.g., PCIe 3.0) may provide enough bandwidth, but the exact interconnects is under exploration [2].

The aim of the AXIOM project is also towards an energy-efficient improvement of the performance of applications, along with benefits in terms of modular scalability of the platform. In the next sections we will describe the first experiments that enabled us to have more confidence with this approach (see Section VI).

IV. The AXIOM platform

The AXIOM platform is architected on the following pillars (see also Figure 1):

- P1 FPGA, i.e. large Programmable Logic for acceleration of functions, soft-IPs, implementing specific AXIOM support for interconnects and scaling,
- P2 General Purpose Cores, to support the OS and for running parts that make little sense on the other accelerators,
- P3 High-Speed, Inexpensive Interconnects to permit scalability and deverticalise the technology, e.g., for toolchains,
- P4 Open-Source Software Stack,
- P5 Lower-Speed Interface for the Cyber-Physical world, such as Arduino [30] connectors, USB, Ethernet, WiFi.

Below we illustrate those pillars more in details.

[P1] In the first phase we will adopt one of the existing solutions such as the Xilinx Zynq [5], (Zynq is a chip-family, the chip can include a dual ARM Cortex-A9@1GHz, 4@6.25Gbps to 16@12.5Gbps transceivers, low-power programmable logic from 28k to 444k logic cells + 240 to 3020 KB BRAM + 80 to 2020 18x25 DSP slices, PCI express, DDR3 memory controller, 2 USB, 2 GbE, 2 CAN, 2SDIO, 2 UART, 2 SPI, 2 I2C, 4x32b GPIO, security features, 2 ADC@12bit 1Msps). The central hearth of the board is the FPGA SoC, so that it can make possible to integrate all the features, to provide customized and reconfigurable acceleration of the specific scenario where the board is deployed and to provide the substrate for board-to-board communication. In our roadmap, we are also considering other options that may be available soon such as the Xilinx Ultrascale+ [31].

[P2] The general purpose cores are used for supporting a number of activities such as the Operating System (or a system task) but also whenever there is a sequential task which needs for more Instruction Level Parallelism rather than other forms of acceleration.

[P3] To keep the cost low we are initially oriented to use the FPGA transceivers and use standard and inexpensive (multiple) connectors such as the SATA connectors (without necessarily use the SATA protocol). Similar solutions had been adopted in the FORMIC board [32].

[P4] The recent success of SBCs such as the UDOO [33] and RaspberryPi further demonstrated the need for using open-source software. Linux has already become a reference example of how open-source software can widen the benefits at any level. While there is not yet a final consensus on which parallel programming model is best, we believe that adopting OmpSs [7] can easy the programmability by providing techniques familiar to the HPC programmer into the Embedded Computing community.

[P5] In order to interface with the physical world the platform includes support for Arduino connectors for General Purpose I/O and other standard interfaces such as the USB, Ethernet, WiFi. Not less important is the capability of interfacing with sensors and actuators or any other type of external shields as in the Arduino platform.

Moreover, DF-Threads make possible to bring together in a single platform all those elements and tackling cross-issues such as a better real-time scheduling: as the inputs should be available before execution of the DF-Threads, the system can be more predictable too.

V. Methodology

In order to flexibly fit the need of designing both the hardware and the software of the AXIOM system, we used the COTSon simulator [34]. COTSon can model the main AXIOM components of Figure 1. Among the important features, COTSon performs full-system simulation: the designer can run, e.g., an off-the-shelf Linux distribution and model in a decoupled way the desired functionalities and their timing behavior. This models a more realistic situation where the OS is interacting with the user programs and includes also any interrupts, exceptions, virtual memory management.

In particular, the key parameters of the modeled cores are described in Table I.

TABLE I: Multicore architectural parameters.

Parameter	Description
SoC	4-cores connected by a shared-bus, IO-hub, MC, high-speed transceivers
Core	1GHz, in-order superscalar
Branch Predictor	two-level (history length=14bits, pattern-history table=16kB, 8-cycle missprediction penalty)
L1 Cache	Private I-cache 32 KB, private D-cache 32 KB, 2 ways, 3-cycle latency
L2 Cache	Private 512 KB, 4 ways, 5-cycle latency
L3 Cache	Shared 4MB, 4 ways, 20-cycle latency
Coherence protocol	MOESI
Main Memory	1 GB, 100 cycles latency
I-L1-TLB, D-L1-TLB	64 entries, full-associative, 1-cycle latency
L2-TLB	512 entries, direct access, 1-cycle latency
Write/Read queues	200 Bytes each, 1-cycle latency

Additionally, the simulator has been extended to support DF-Threads [14]. This means that the simulator is also modeling the Distributed Thread Scheduler [29], which is implemented on the Programmable Logic through the block XSM (eXtended Shared Memory) of Figure 1.

As for the interconnects among SoCs, we are currently exploring several options as offered by the latest technologies. In the COTSon simulator we are performing limit-study experiments assuming that we can achieve enough bandwidth and low latency at a reasonable cost. This part is explored in detail within the AXIOM project, but will not be illustrated here.

A. Matrix Multiplication Benchmark

We selected the Matrix Multiplication kernel to test the performance evaluation infrastructure and to verify the feasibility of supporting DF-Threads on the AXIOM platform.

The Matrix Multiplication benchmark has the following characteristics:

- Blocked matrix multiplication using the classical 3 nested loops algorithm.
- Square matrices of size $n \times n$, where $n = 200, 250, 320, 400, 500, 640, 800$.
- Block size $b = 10$ and $b = 25$.

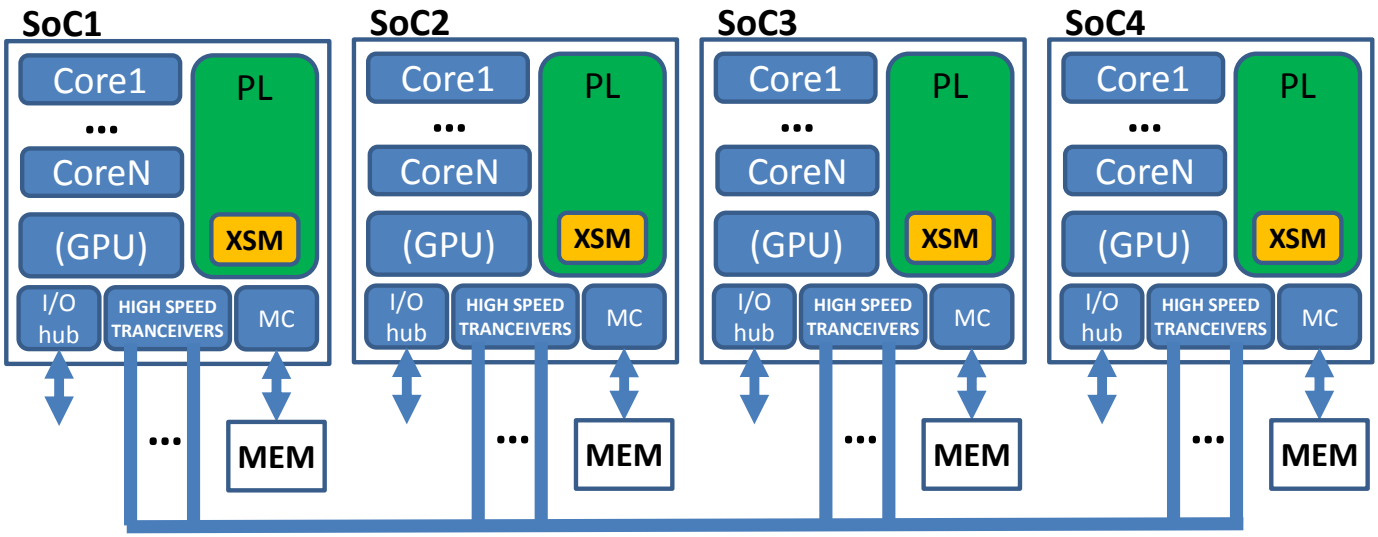

Fig. 1: AXIOM Scalable Architecture. An instance consisting of four boards, each one based on the same System-on-Chip (SoC). GPU is an optional component. MC=Memory Controller. PL=Programmable Logic. XSM=eXtended Shared Memory

Since, in this case, the number of operations is $O(n^3)$, the size n of the matrix has been chosen in such a way that the cubed size of each number of the size sequence is approximately the double of the cubed size of the previous number, i.e., $250^3 \approx 2 \times 200^3$ and so on. This is useful to perform the weak scaling tests (Figure 4).

The DF-Threads are generated in such a way that each thread performs the matrix multiplication of each block, therefore we can expect a number of threads equal to n/b.

VI. Experiments

In order to verify the feasibility of running programs not only on the single board but also on multiple boards thanks to the adopted programming model (OmpSs [7]) and the underlying runtime system a key point is being able to schedule and execute the generated DF-Threads across the boards. This implies that the memory which is local to each node (see Figure 1) has to be managed in such a way that it appears as shared to the rest of the SoCs/boards.

The XSM block of Figure 1 serves to that goal by bookkeeping the DF-Threads and by appropriately moving the data where is needed.

We performed two classical tests to verify that the proposed paradigm can permit the distribution of the threads:

- *S*trong Scaling tests,
- *W*eak Scaling tests.

With the strong scaling tests, we increase the number of SoCs (for simplicity we refer to the single SoC as if it were a board) and we want to verify if the speedup t1/tN (being t1 the time to execute the program on a single SoC and tN the time to execute the program on N SoCs) is close to the ideally linear speedup (Figures 2, 3 and 5).

With the weak scaling tests, we increase *both* the number of SoCs and the quantity of work to be executed, in the same proportion.

As explained in the Subsection V-A, the number of operations varies as $O(n^3)$ where n is the size of the square matrix. Therefore, we have to increase the size of the matrix by a factor $\sqrt[3]{2}$, as we increase the number of SoCs in order to perform the weak scaling tests (Figure 4). In the latter case, the ideal curve is a horizontal line with value 1, which (ideally) means that as we increase the quantity of work and the SoCs (in the same proportion) the time tN equals the time t1, i.e., the scaled systems keeps up with the increased volume of data.

As we can see in Figure 2, as the number of SoCs is increased from 1 to 2 and then 4, the scalability is good enough (close to ideal), especially for higher matrix sizes (e.g., 320). In fact, for higher matrix sizes, the number of avaialble DF-threads is also higher.

The deviation from ideal behavior is mainly due to:

- Too few DF-Threads from the program,
- Increased data movement.

In fact, as the number of DF-Threads is equal to n/b (see Subsection V-A), in the case of $n = 200$ we only have 5 threads to be assigned to the each of the SoCs; moreover, since our SoC has 4 cores, some of the cores may remain idle. This is visible in Figure 2 for the curve for $n = 200$ and 4 SoCs with a drop in the scalability. We reported the strong scaling curves for some other values (200, 250, 320, 400) to verify the sensitivity to the input data; in the tests of Figure 2 the block size is kept constant.

One other aspect regards the influence of the Operating System. The curves in Figure 2 reflect the execution of only the User part of the program. We extracted also the strong scaling curves that reflect *both* the User and the Kernel instructions (Figure 3).

As we can see, the strong scaling is affected by the OS: all the curves of Figure 2 are now compressed towards the bottom part of the Figure 3, as the time spent in Kernel mode ranges from 6% to 60% in those tests. This indicates a strong

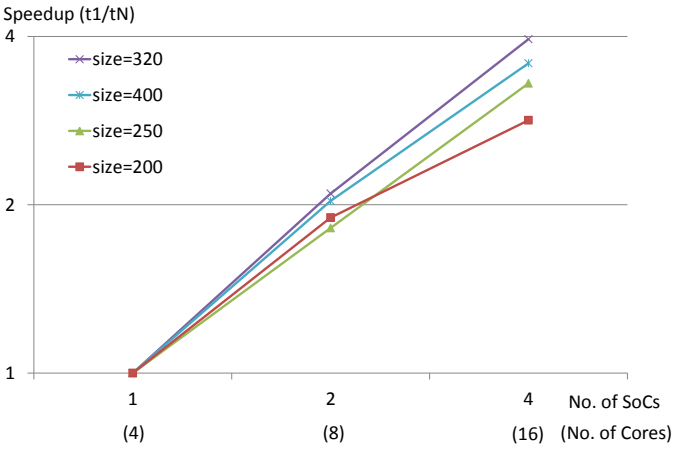

Fig. 2: Strong Scaling for benchmark Dense Matrix Multiplication (Square Matrices). Matrix size varies: 200,250,320,400. Block size is constant and equal to 10. The time used for calculating the speedup accounts only for the User Time (without Kernel Time).

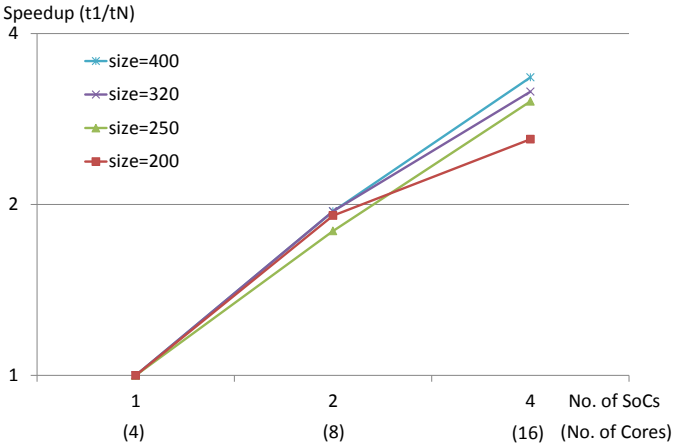

Fig. 3: Strong Scaling for benchmark Dense Matrix Multiplication (Square Matrices). Matrix size varies: 200,250,320,400. Block size is constant and equal to 10. The time used for calculating the speedup accounts for both the User Time and the Kernel Time.

need for using a full-system simulator and not neglecting the OS activities for a proper platform design.

In the weak scaling tests of Figure 4, we observe that for some matrix size ($n = 400$) we have a high efficiency, as it was for the strong scaling tests.

However, please note that in the weak scaling tests, each curve corresponding to three different matrix sizes for each of the X-axis values (the number of SoCs). For example, for the curve corresponding to $n = 400$ the sequence of data is: 1 SoC $\rightarrow n = 400$, 2 SoCs $\rightarrow n = 500$, 4 SoCs $\rightarrow n = 640$. In particular, for 4 SoCs the efficiency drops as it has an implied matrix size of $n = 640$ and the data set is large enough to cause a significant miss rate increase in the L2 cache (not shown in the figures).

Finally, we explored the sensitivity to the thread granularity, by choosing a larger block size. A larger block generates longer DF-threads to process such matrix block. In Figure 5,

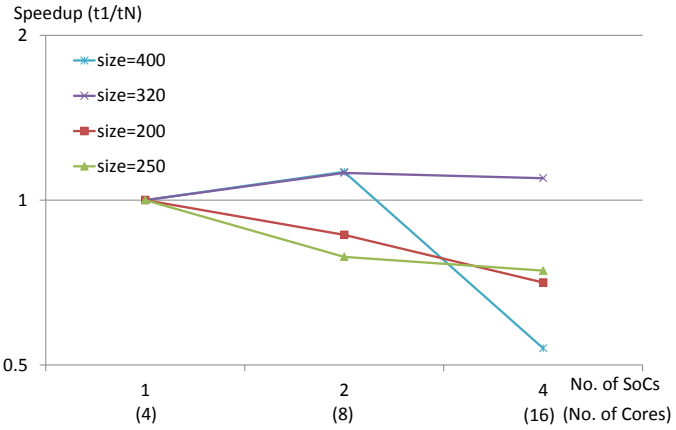

Fig. 4: Weak Scaling for benchmark Dense Matrix Multiplication (Square Matrices). Matrix size varies: 200,250,320,400 on each single SoC (to keep the work almost constant on each core/SoC). Block size is constant and equal to 10. The time used for calculating the speedup only accounts User Time (without Kernel Time).

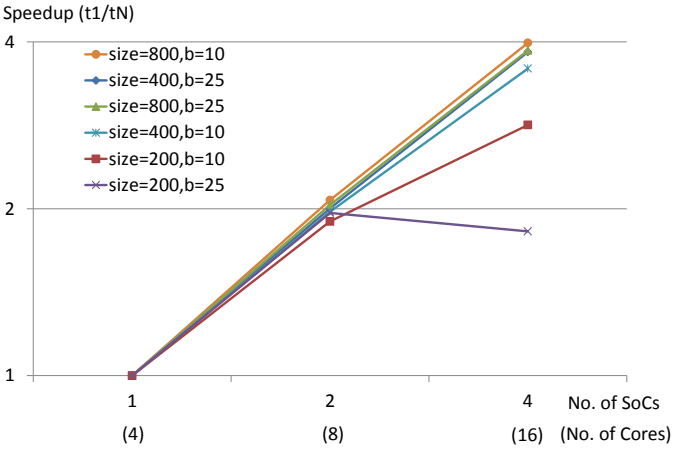

Fig. 5: Effect of the thread granularity (through the block size) on strong scaling for benchmark Dense Matrix Multiplication (Square Matrices). Matrix size varies: 200,250,320,400. Block size assumes values 10 and 25. The time used for calculating the speedup accounts only for the User Time (without Kernel Time).

we analyzed the situation for three matrix sizes ($n = 200$, $n = 400$, $n = 800$) and while the block size b is equal to 10 and 25.

As we can see, having larger threads implies less opportunities for parallelism, as the number of DF-threads is smaller. This is particularly evident for the curve for $n = 200$ and $b = 25$. Moreover, there are combinations of n and b (e.g., $n/b = 16$) where the scaling is better as the number of available threads is a multiple of the number of cores. For larger matrix sizes (e.g. 800) as noticed, the L2 cache tend to suffer more misses, thus affecting the performance.

Strong and weak scaling tests are therefore useful to analyze the performance of the embedded system constituted of N SoCs. The current results show a good potential for achieving scalability across SoCs.

VII. Conclusions

In this paper, we advocate the deployment of Scalable Embedded Systems. Solutions studied in the domain of High-Performance Computing are more and more available also in the Embedded Computing domain.

In particular, we have shown that DF-Threads could be deployed to permit scalability across boards, while keeping a data-flow based interface to the OmpSs programming model, an extension of the OpenMP programming model. We presented our experience in the context of the AXIOM project.

The availability of open-source and open-hardware platforms combined with the above concepts could permit performance scalability of Embedded Systems.

References

[1] Duranton, Marc and De Bosschere, Koen and Cohen, Albert and Maebe, Jonas and Munk, Harm, "HiPEAC Vision 2015." [Online]. Available: https://www.hipeac.net/assets/public/publications/vision/hipeac-vision-2015_Dq0boL8.pdf

[2] D. Theodoropoulos *et al.*, "The AXIOM project (agile, extensible, fast i/o module)," in *IEEE Proc. 15th Int.l Conf. on Embedded Computer Systems: Architecture, MOdeling and Simulation*, July 2015.

[3] C. Alvarez, E. Ayguade, J. Bueno, A. Filgueras, D. Jimenez-Gonzalez, X. Martorell, N. Navarro, D. Theodoropoulos, D. N. Pnevmatikatos, C. Scordino, P. Gai, C. Segura, C. Fernandez, D. Oro, J. R. Saeta, P. Passera, A. Pomella, A. Rizzo, and R. Giorgi, "The AXIOM software layers," in *IEEE Proc. 18th EUROMICRO-DSD*, 2015, pp. 117–124.

[4] P. Burgio, C. Alvarez, E. Ayguad, A. Filgueras, D. Jimnez-Gonzlez, X. Martorell, N. Navarro, and R. Giorgi, "Simulating next-generation cyber-physical computing platforms," in *Springer Proc. snd Workshop Challenges and New Approaches for Dependable and Cyber-Physical System Engineering (De-CPS'15)*, Jun. 2015.

[5] Xilinx Inc., "Zynq Series." [Online]. Available: http://www.xilinx.com/content/xilinx/en/products/silicon-devices/soc/zynq-7000.html

[6] A. Filgueras, D. Jiminez, C. Alvarez, X. Martorell, J. Langer, J. Noguera, and K. Vissers, "Coarse-Grain Performance Estimator for Heterogeneous Parallel Computing Architectures like Zynq All-Programmable SoC," in *FSP*, Sep. 2015.

[7] J. Bueno, L. Martinell, A. Duran, M. Farreras, X. Martorell, R. Badia, E. Ayguade, and J. Labarta, "Productive cluster programming with OmpSs," *Euro-Par 2011 Parallel Processing*, pp. 555–566, 2011.

[8] S. Wesner, L. Schubert, R. Badia, A. Rubio, P. Paolucci, and R. Giorgi, "Special section on terascale computing," *ELSEVIER Future Generation Computer Systems*, vol. 53, pp. 88–89, July 2015.

[9] J. M. P. Cardoso, T. Carvalho, J. G. F. Coutinho, R. Nobre, R. Nane, P. C. Diniz, Z. Petrov, W. Luk, and K. Bertels, "Controlling a complete hardware synthesis toolchain with LARA aspects," *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 37, no. 8-C, pp. 1073–1089, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2013.06.001

[10] J. M. P. Cardoso, J. G. F. Coutinho, T. Carvalho, P. C. Diniz, Z. Petrov, W. Luk, and F. Gonalves, "Performance-driven instrumentation and mapping strategies using the lara aspect-oriented programming approach," *Software: Practice and Experience*, pp. n/a–n/a, 2014. [Online]. Available: http://dx.doi.org/10.1002/spe.2301

[11] M. Milutinovic, J. Salom, N. Trifunovic, and R. Giorgi, *Guide to DataFlow Supercomputing*. Berlin, DE: Springer, Apr 2015.

[12] W. Ahmed, M. Shafique, L. Bauer, and J. Karlsruhe, "Adaptive resource management for simultaneous multitasking in mixed-grained reconfigurable multi-core processors," in *Proc. Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Oct 2011, pp. 365–374.

[13] J. Clemente, V. Rana, D. Sciuto, I. Beretta, and D. Atienza, "A hybrid mapping-scheduling technique for dynamically reconfigurable hardware," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept 2011, pp. 177–180.

[14] R. Giorgi and P. Faraboschi, "An introduction to DF-Threads and their execution model," in *IEEE MPP*, Paris, France, Oct. 2014, pp. 60–65.

[15] M. Herlihy and J. E. B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, ser. ISCA '93. New York, NY, USA: ACM, 1993, pp. 289–300.

[16] F. Yazdanpanah, C. Alvarez-Martinez, D. Jimenez-Gonzalez, and Y. Etsion, "Hybrid dataflow/von-neumann architectures," *IEEE Trans. on Parallel and Distrib. Systems*, vol. 25, no. 6, pp. 1489–1509, June 2014.

[17] L. Verdoscia, R. Vaccaro, and R. Giorgi, "A clockless computing system based on the static dataflow paradigm," in *Proc. IEEE Int.l Workshop on Data-Flow Execution Models for Extreme Scale Computing (DFM-2014)*, Edmonton, Canada, Aug. 2014, pp. 30–37.

[18] ——, "A matrix multiplier case study for an evaluation of a configurable dataflow-machine," in *ACM CF'15 - LP-EMS*, May 2015, pp. 1–6.

[19] M. Solinas, M. Badia, F. Bodin, A. Cohen, P. Evripidou, P. Faraboschi, B. Fechner, G. Gao, A. Garbade, S. Girbal, D. Goodman, B. Khan, S. Kolia, F. Li, M. Lujn, A. Mendelson, L. Morin, N. Navarro, A. Pop, P. Trancoso, T. Ungerer, M. Valero, S. Weis, S. Zuckerman, and R. Giorgi, "The teraflux project: Exploiting the dataflow paradigm in next generation teradevices," in *IEEE Proc. 16th EUROMICRO-DSD*, Santander, Spain, 2013, pp. 272–279.

[20] R. Giorgi, R. Badia, F. Bodin, A. Cohen, P. Evripidou, P. Faraboschi, B. Fechner, G. Gao, A. Garbade, R. Gayatri, S. Girbal, D. Goodman, B. Khan, S. Kolia, J. Landwehr, N. L. Minh, F. Li, M. Lujn, A. Mendelson, L. Morin, N. Navarro, T. Patejko, A. Pop, P. Trancoso, T. Ungerer, I. Watson, S. Weis, S. Zuckerman, and M. Valero, "Teraflux: Harnessing dataflow in next generation teradevices," *ELSEVIER Microprocessors and Microsystems*, vol. 38, no. 8, Part B, pp. 976–990, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933114000490

[21] A. Mondelli, N. Ho, A. Scionti, M. Solinas, A. Portero, and R. Giorgi, "Dataflow support in x86_64 multicore architectures through small hardware extensions," in *IEEE Proc. DSD*, August 2015, pp. 526–529.

[22] R. Giorgi, "Teraflux: Exploiting dataflow parallelism in teradevices," in *ACM Computing Frontiers*, Cagliari, Italy, May 2012, pp. 303–304.

[23] N. Ho, A. Portero, M. Solinas, A. Scionti, A. Mondelli, P. Faraboschi, and R. Giorgi, "Simulating a multi-core x86_64 architecture with hardware isa extension supporting a data-flow execution model," in *IEEE Proc. AIMS-2014*, Madrid, Spain, Nov. 2014, pp. 264–269.

[24] N. Ho, A. Mondelli, A. Scionti, M. Solinas, A. Portero, and R. Giorgi, "Enhancing an x86_64 multi-core architecture with data-flow execution support," in *ACM Computing Frontiers*, Ischia, Italy, May 2015, pp. 1–2.

[25] R. Giorgi, "Accelerating haskell on a dataflow architecture: a case study including transactional memory," in *Proc. Int.l Conf. on Computer Engineering and Applications (CEA)*, Dubai, UAE, Feb. 2015, pp. 91–100. [Online]. Available: http://www.wseas.us/e-library/conferences/2015/Dubai/CEA/CEA-12.pdf

[26] ——, "Transactional memory on a dataflow architecture for accelerating haskell," *WSEAS Trans. Computers*, vol. 14, pp. 794–805, 2015.

[27] S. Weis, A. Garbade, J. Wolf, B. Fechner, A. Mendelson, R. Giorgi, and T. Ungerer, "A fault detection and recovery architecture for a teradevice dataflow system," in *Proc. IEEE Int.l Workshop on Data-Flow Execution Models for Extreme Scale Computing (DFM)*, Oct. 2011, pp. 38–44.

[28] S. Weis, A. Garbade, B. Fechner, A. Mendelson, R. Giorgi, and T. Ungerer, "Architectural support for fault tolerance in a teradevice dataflow system," *Springer Int.l Journal of Parallel Programming*, pp. 1–25, May 2014. [Online]. Available: http://dx.doi.org/10.1007/s10766-014-0312-y

[29] R. Giorgi and A. Scionti, "A scalable thread scheduling co-processor based on data-flow principles," *ELSEVIER Future Generation Computer Systems*, vol. 53, pp. 100–108, July 2015.

[30] M. Banzi, *Getting Started with Arduino*. Sebastopol, CA: Make Books - Imprint of: O'Reilly Media, 2008.

[31] Xilinx Inc., "Xilinx UltraScale Architecture." [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp435-Xilinx-UltraScale.pdf

[32] S. Lyberis, G. Kalokerinos, M. Lygerakis, V. Papaefstathiou, D. Tsaliagkos, M. Katevenis, D. Pnevmatikatos, and D. Nikolopoulos, "Formic: Cost-efficient and scalable prototyping of manycore architectures," in *FCCM*, 2012, pp. 61–64.

[33] E. Palazzetti, *Getting Started with UDOO*, iResha Raman, Ed. Packt Publishing, 2015.

[34] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "COTSon: infrastructure for full system simulation," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, pp. 52–61, 2009.