Garcia, J.D. Refactoring software to heterogeneous parallel platforms. *J Supercomput* 75, 3997–4000 (2019)

DOI: 10.1007/s11227-019-02942-2

Refactoring software to heterogeneous parallel platforms

J. Daniel Garcia[1]

1 Introduction

The evolution of approaches for parallel programming models in last years has been a constant and seems to be a trend for the upcoming years. That evolution is not only driven by the pervasive use of multi-/many-core systems but by the fact that more and more systems have become heterogeneous. The classical scenario where OpenMP [1] would be the dominant approach at the node level, while MPI [2] would do the same thing with clusters seems now to be competing with a variety of solutions.

According to the latest list in Top 500 supercomputers [3], released June 2019, the number of cores per socket seems to be moderately increasing while the number of systems using some accelerator is in high proportion. In those systems using an accelerator, the most common one seems to be GPU, which seems to have become the dominant accelerator. Outside the supercomputing world, the use of heterogeneous systems is also increasing its momentum for improving performance in single-node applications ranging from scientific software in workstations to many embedded applications.

A major challenge comes from the fact that many legacy applications need be transformed so that they can be delivered in new heterogeneous architectures. Because there is no clear dominant heterogeneous architecture, the cost of each new port seems to contain many architecture-specific issues. As there is a wide variety of programming models for heterogeneous systems, one of the possible techniques seems to be software refactoring [4]. Software refactoring is known as the process of restructuring existing code without changing the external behavior but improving non-functional attributes of the software. In this case, the main non-functional attribute to be improved is performance. Over the years, solutions for software refactoring [5] to parallel systems have focused mainly in homogenous systems.

With the recent emergence of many different programming models, languages and frameworks for expressing parallelism, the refactoring approach has become more prominent. Besides, the programming models for multi-core, a number of

josedaniel.garcia@uc3m.es

[1] University Carlos III of Madrid, Leganes, Spain

specific models for GPU and other accelerators need to be considered. An answer to this problem has come from the parallel patterns community [6]. Parallel patterns can be seen as a mechanism to express parallelism in existing sequential applications. They allow to raise the abstraction level and to ensure that application logic and implementation details can be kept separate as distinct aspects of the software. Many typical parallel patterns [7] can be used to express typical algorithms. Moreover, many of those patterns are easily exploitable by different heterogeneous parallel architectures. In fact, parallel patterns have become an excellent way of expressing algorithms in a portable way between traditional multi-cores and more innovative accelerator-based systems.

1.1 Special issue presentation

This special issue includes 12 new research contributions. Four of these correspond to extended research contributions from the RePara 2017 workshop that was held in conjunction with the ParCo 2017 Conference, held in Bologna, Italy, during September 2017. The remaining eight contributions were selected from an open call for papers.

In "Prediction models for performance, power, and energy efficiency of software executed on heterogeneous hardware" [8], Bán et al. make use of both static source code metrics, and dynamic execution measuring time, power and energy to build predictive models on improvements. Using those models for training, they found that using static code metrics to predict concrete continuous values of dynamic properties cannot be achieved in general. However, they obtained good results in terms of category prediction which in most cases is enough to make refactoring decisions.

In "Supporting structured parallel program design, development and tuning in FastFlow" [9], Gazarri and Danelutto focus on the separation of concerns provided by structured parallel programming. They describe a shell that allows to explore the design space of functionally equivalent parallel compositions with different non-functional properties.

In "Stream parallelism with ordered data constraints on multi-core systems" [10], Griebler et al. propose a new technique that can be easily integrated into different C++ parallel programming frameworks supporting stream parallelism. The strategy focuses on those cases where ordering is relevant in stream parallelism with an irregular number of tasks in different stages.

In "SpExSim: assessing kernel suitability for C-based high-level hardware synthesis" [11], Oppermann et al. introduce techniques for performing surveys on existing legacy C codebases that could be accelerated by FPGA-based compute units. Their approach focuses on high-level synthesis of source code to minimize development costs and efforts.

In "Simultaneous multiprocessing in a software-defined heterogeneous FPGA" [12], Núñez-Yáñez et al.'s reductions of overheads are investigated to enable the utilization of all CPU cores and an FPGA in a heterogeneous environment, when a high-level general-purpose language as C++ is used.

In “Hybrid static–dynamic selection of implementation alternatives in heterogeneous environments” [13], David del Rio et al. focus on the combination of static and dynamic techniques to select mappings of software components from an application to different computing devices in a heterogeneous system. Their technique is capable of generating a compile-time decision tree that can be used to select the best mapping.

In “On dynamic memory allocation in sliding-window parallel patterns for streaming analytics” [14], Torquati et al. study the issue of dynamic memory management for streaming parallelism. Their study shows that the default memory management mechanisms provided by the C++ standard are not the most adequate for this subset of applications. They provide alternate techniques combining custom allocators with variants of smart pointers to improve the performance of pipelines and other streaming patterns.

In “Experiences with implementing parallel discrete-event simulation on GPU” [15], Sang et al. focus on porting discrete-event simulators to run them on GPU. They compare two open-source approaches to provide interfaces that are similar to existing C++ Standard Template Library which is quite in line with current parallel STL and gives a path toward upcoming parallelism extensions in new versions of C++.

In “Multi-objective algorithms for the application mapping problem in heterogeneous multiprocessor embedded system design” [16], Sinaei and Fatemi address the problems in Electronic System-Level design where both simulation and design space exploration are critical performance steps. A special focus is given to two specific multi-objective optimization algorithms.

In “Toward a software transactional memory for heterogeneous CPU–GPU processors” [17], Villegas et al. introduce APUTM, a software transactional memory solution for APUs (Accelerated Processing Units) where CPU and GPU are integrated into a single chip. APUTM allows experimenting and better understanding the trade-offs in this category of platforms.

In “A hybrid sample generation approach in speculative multithreading” [18], Li et al. apply machine learning techniques to speculative multithreading to perform thread partitions. They apply those techniques to benchmark applications and compare them to techniques using heuristic rules-based approaches, which cannot generate adaptive samples.

In “Toward fault-tolerant hybrid programming over large-scale heterogeneous clusters via checkpointing/restart optimization” [19], Chen et al. focus on programming models for large clusters where traditional models, as MPI + X, have been more concerned about the performance and reliability. Their approach is supported by in-memory checkpointing providing new capabilities for heterogeneous applications and hence simplifying application-level checkpointing. It is quite interesting that results were validated on different benchmarks and applications on the Tianhe-2 supercomputer.

In summary, the papers included in this special issue are representative of the progress achieved by the research community at various levels from the very high level using parallel patterns to lower levels using, for example, transactional software memory. Also the integration of GPUs and FPGAs in the landscape is essential to achieve better performance in different categories of applications. All these innovative research directions will contribute to better achieve the long-term goal of better refactoring of existing applications to new and evolving parallel heterogeneous architectures.

References

1. Dagum L, Menon R (1998) OpenMP: an industry standard API for shared-memory programming. IEEE Comput Sci Eng 5(1):46–55
2. Gropp WD, Gropp W, Lusk E, Skjellum A (1999) Using MPI: portable parallel programming with the message-passing interface. MIT Press, London
3. TOP500 Supercomputer sites (2018) www.top500.org. Accessed 24 June 2019
4. Giswold WG (1992) Program restructuring as an aid to software maintenance. Ph.D. dissertation. University of Washington
5. Dig Danny (2011) A refactoring approach to parallelism. IEEE Softw 28(1):17–22
6. McCool M, Reinders J, Robinson A (2012) Structured parallel programming: patterns for efficient computation. Morgan-Kaufmann, Burlington
7. Gorlatch S, Cole M (2011) Parallel skeletons. In: Padua DA (ed) Encyclopedia of parallel computing. Springer, New York, pp 1417–1422
8. Bán D, Ferenc R, Siket I, Kiss A, Gyimóthy T (2019) Prediction models for performance, power, and energy efficiency of software executed on heterogeneous hardware. J Supercomput. https://doi.org/10.1007/s11227-018-2252-6
9. Gazarri L, Danelutto M (2019) Supporting structured parallel program design, development and tuning in FastFlow. J Supercomput. https://doi.org/10.1007/s11227-018-2448-9
10. Griebler D, Hoffmann RB, Danelutto M, Fernandes LG (2019) Stream parallelism with ordered data constraints on multi-core systems. J Supercomput. https://doi.org/10.1007/s11227-018-2482-7
11. Oppermann J, Sommer L, Koch A (2019) SpExSim: assessing kernel suitability for C-based high-level hardware synthesis. J Supercomput. https://doi.org/10.1007/s11227-017-2101-z
12. Nunez-Yanez J, Amiri M, Hosseinabady M, Rodriguez A, Asenjo R, Navarro A, Suarez D, Gran R (2019) Simultaneous multiprocessing in a software-defined heterogeneous FPGA. J Supercomput. https://doi.org/10.1007/s11227-018-2367-9
13. Astorga DR, Dolz MF, Fernandez J, Garcia-Blas J (2019) Hybrid static–dynamic selection of implementation alternatives in heterogeneous environments. J Supercomput. https://doi.org/10.1007/s11227-017-2147-y
14. Torquati M, Mencagli G, Drocco M, Aldinucci M, De Matteis T, Daneultto M (2019) On dynamic memory allocation in sliding-window parallel patterns for streaming analytics. J Supercomput. https://doi.org/10.1007/s11227-017-2152-1
15. Sang J, Lee C-R, Rego V, King C-T (2019) Experiences with implementing parallel discrete-event simulation on GPU. J Supercomput. https://doi.org/10.1007/s11227-018-2254-4
16. Sinaei M, Fatemi O (2019) Multi-objective algorithms for the application mapping problem in heterogeneous multiprocessor embedded system design. J Supercomput. https://doi.org/10.1007/s11227-018-2442-2
17. Villegas A, Navarro A, Asenjo R, Plata O (2019) Towards a software transactional memory for heterogeneous CPU–GPU processors. J Supercomput. https://doi.org/10.1007/s11227-018-2347-0
18. Li Y, Zhao Y, Sun L, Shen M (2019) A hybrid sample generation approach in speculative multithreading. J Supercomput. https://doi.org/10.1007/s11227-017-2118-3
19. Chen C, Du Y, Zuo K, Fang J, Yang C (2019) Toward fault-tolerant hybrid programming over large-scale heterogeneous clusters via checkpointing/restart optimization. J Supercomput. https://doi.org/10.1007/s11227-017-2116-5