An Optimized Task-Based Programming Model for Embedded Many-core Computing Platforms

Nowadays multi- and many-core computing platforms are widely adopted as a viable solution to accelerate compute-intensive workloads. Inter alia, heterogeneous platforms including a general-purpose host processor and a parallel programmable accelerator have the potential to dramatically increase the peak performance/Watt of embedded platforms. Parallel accelerators provide tens to hundreds of small processing elements (PEs), typically organized in clusters sharing on-chip L1 memory and communicating via low-latency, high-throughput on-chip interconnections. PEs are simpler w.r.t. common multi-core architectures to offer a better tradeoff between parallel computation and power consumption. Parallel accelerators differ from GPGPUs in two main traits. First, PEs are not restricted to run the same instruction on different data, in an effort to improve execution efficiency of branch-rich computations and to support a more flexible workload-to-PE distribution. Second, embedded many-core accelerators do not rely on massive multithreading to hide memory latency, but they rely instead on DMA engines and double buffering, which give more control on the bandwidth vs. latency tradeoff, but require more programming effort. The adoption of these devices highly complicates application development, whereas it is widely acknowledged that software development is a critical activity for the platform design, as it affects development cost and time-to-market. The introduction of parallel architectures raises the need for programming paradigms capable of effectively leveraging an increasing number of processors. According to software engineering principles, programming models should expose high-level constructs for outlining the available parallelism in applications, without the need for programmers to handle performance scalability issues by expertizing on low-level hardware details. In this scenario the study of optimization techniques to program parallel accelerators is paramount for two main objectives: first, improving performance and energy efficiency of the platform, which are key metrics for embedded computing systems; second, enforcing software engineering practices with the aim to guarantee code quality and reduce software costs.

In this technical presentation we discuss the use of OpenMP tasking as a general-purpose programming model to support the execution of diverse workloads, and we introduce a set of runtime-level techniques to support fine-grain tasks on many-core accelerators. The tasking abstraction provides a powerful conceptual framework to exploit irregular parallelism in embedded applications, but its practical implementation requires sophisticated runtime support, which typically implies important space and time overheads. The applicability of this approach is often limited to applications exhibiting work units which are coarse-grained enough to amortize these overheads. While this is often the case for general-purpose systems and associated workloads, things are different when considering embedded computing systems. Minimizing runtime overheads is thus a primary challenge to enable the benefits of tasking on these systems. We designed an optimized runtime environment supporting the OpenMP tasking model on an embedded shared-memory cluster [1] [2], validating our work on a cycle-accurate virtual platform and then performing tests on hardware platforms (ST Microelectronics STHORM [4] [3]), Kalray MPPA and PULP [6]). We also provide support to *untied tasks*, which can be resumed by any available thread, thus significantly increasing the potential for parallelism exploitation. On top of this extended runtime, we implement support for work-first-scheduling (WFS) and associated cutoff policies. Experimental results on compute-intensive applications highlights three main benefits. First, our solution can achieve the *maximum speed-up with an average task granularity of 7500 cycles*, while previous approaches require about 100000 cycles to achieve the same performance level. Second, WFS enables significantly higher speedups (up to 60%) when untied tasks are used in recursive patterns. Third, cutoff policies on top of the provided support for untied tasks allow to achieve nearly-ideal speedups for recursive patterns around 5K cycles. These features enable the adoption of OpenMP tasking in embedded runtime environments, including state-of-the-art applications in the time-critical domain [5].

References

[1] P. Burgio, G. Tagliavini, A. Marongiu, and L. Benini, "Enabling fine-grained OpenMP tasking on tightly-coupled shared memory clusters," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, 2013, pp. 1504–1509.

[2] P. Burgio, G. Tagliavini, F. Conti, A. Marongiu, and L. Benini, "Tightly-coupled hardware support to dynamic parallelism acceleration in embedded shared memory clusters," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 156.

[3] A. Marongiu, A. Capotondi, G. Tagliavini, and L. Benini, "Improving the programmability of STHORM-based heterogeneous systems with offload-enabled OpenMP," in *Proceedings of the First International Workshop on Many-core Embedded Systems*. ACM, 2013, pp. 1–8.

[4] ——, "Simplifying many-core-based heterogeneous soc programming with offload directives," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 4, pp. 957–967, 2015.

[5] L. M. Pinho, E. Quiones, M. Bertogna, A. Marongiu, J. P. Carlos, C. Scordino, and M. Ramponi, "P-SOCRATES: A Parallel Software Framework for Time-Critical Many-Core Systems," in *2014 17th Euromicro Conference on Digital System Design*. IEEE, 2014, pp. 214–221.

[6] D. Rossi, I. Loi, F. Conti, G. Tagliavini, A. Pullini, and A. Marongiu, "Energy efficient parallel computing on the pulp platform with support for openmp," in *Electrical & Electronics Engineers in Israel (IEEEI), 2014 IEEE 28th Convention of*. IEEE, 2014, pp. 1–5.