Combining the tasklet model with OpenMP

Luis Miguel Pinho*
Eduardo Quiñonez
Sara Royuela

*CISTER Research Centre

CISTER-TR-180301

2018/04/18

Combining the tasklet model with OpenMP

Luis Miguel Pinho*, Eduardo Quiñonez, Sara Royuela

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: lmp@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Previous workshops have discussed a proposal to augment Ada with fine-grained parallelism, based on the notion of tasklets, a lightweight parallel entity. Recent works have shown the convergence of this model with the OpenMP tasking model and have proposed their coexistence. In this paper we provide a status of the existent works, and describe how these models could be combined.

Combining the tasklet model with OpenMP

Luis Miguel Pinho
CISTER/ISEP
Portugal
lmp@isep.ipp.pt

Eduardo Quiñones, Sara Royuela
BSC
Spain
{eduardo.quinones,sara.royuela}@bsc.es

Abstract

Previous workshops have discussed a proposal to augment Ada with fine-grained parallelism, based on the notion of tasklets, a lightweight parallel entity. Recent works have shown the convergence of this model with the OpenMP tasking model and have proposed their coexistence. In this paper we provide a status of the existent works, and describe how these models could be combined.

1 The Tasklet Model

The existent proposal to extend Ada with a fine-grained parallelism model is based on the notion of tasklets [1], lightweight computation units, which would allow the specification of potential parallelism, not fully controlled by the programmer, but under the control of the compiler and the runtime. In that regard, the tasklet model follows the same principle of other parallel tasking models used in the general purpose and high-perfomance domains, in which the programmer uses special syntax to indicate where parallelism opportunities occur in the code, whilst the compiler and runtime co-operate to provide parallel execution, when possible.

In the tasklet model, each Ada task is a graph of multiple tasklets using a fully-strict fork-join model [2] (Figure 1). Tasklets can be spawned by other tasklets (fork), and need to synchronize with the spawning tasklet (join). Tasklets as defined are orthogonal to Ada tasks and execute within the semantic context of the task from which they have been spawned, whilst inheriting the properties of the task such as identification, priority and deadline. The concept is that the model allows a complete graph of potential parallel execution to be extracted during the compilation phase.

Together with the Global aspects proposed in [3], it is thus possible to manage the mapping of tasklets and data allocation, as well as prevent unprotected parallel access to shared variables. Although not a topic addressed in [3], the work considers that issues such as data allocation and contention for hardware resources are key challenges for parallel systems, and therefore compilers and tools must have more information on the dependencies between the parallel computations, as well as data, to be able to generate more efficient programs.

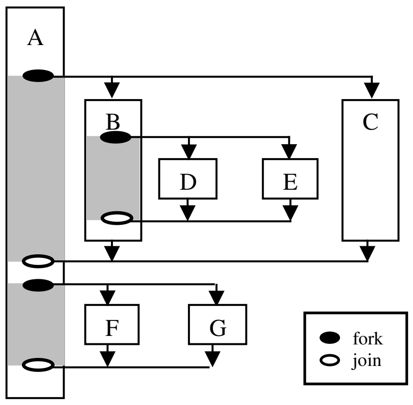


Figure 1. Task DAG example [4]

The tasklet execution model (Figure 2) is based on the notion of abstract executors [4], which carry the actual execution of Ada tasks in the platform, under different progress guarantees that the compiler and runtime need to provide to the parallel execution. The model also specifies that calls by different tasklets of the same task into the same protected object are treated as different calls resulting in distinct protected actions, enabling to synchronize tasklets with protected operations [4].

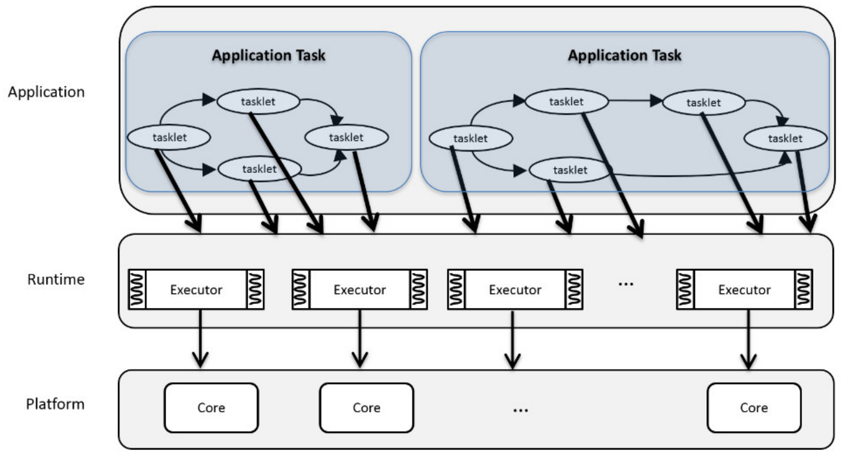


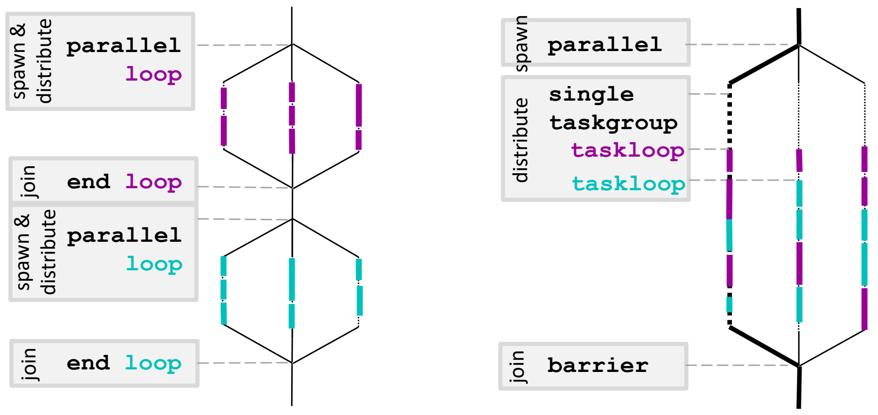
Figure 2. Execution model [4]

The tasklet model is being currently considered for standardization, with a set of Ada Issues (AI) being discussed within the Ada Rapporteur Group (ARG). The tasklet model itself, as well as the Ada constructs for parallel execution, are specified in AI12-0119-1 [5]. A relevant difference to the model proposed in [4] is that tasklets are not allowed to perform potentially blocking operations.

2 Combining the OpenMP Tasking Model and the Tasklet Model

Recently, a work has analysed [6] the similarities of the tasklet model with the OpenMP Tasking model [7]. In this work, the term task in OpenMP is not related to Ada tasks but to tasklets, as OpenMP tasks are lightweight parts of the code that can be executed in parallel by worker threads,. The tasking model appears in OpenMP 3.0 from the need of efficiently and easily implementing certain types of parallelism: unbounded loops, recursion, unstructured parallelism, etc, which clearly complement the structured parallelism approach of the tasklet model.

The OpenMP tasking model follows the same principle of the tasklet model, where the compiler and the runtime system are the ones responsible for generating and executing the OpenMP tasks, based on specific OpenMP constraints (e.g. control-flow or data-flow dependencies) and thread availability. As shown in [8], the forms of parallelism defined by OpenMP are compatible with those proposed for Ada tasklets. Furthermore, the relaxed fork-join model defined in OpenMP, in front of the strict model defined for Ada tasklets, allows for a more flexible execution, as *spawn* and *distribution operations* are not done at the same point of the execution (Figure 3). That is, in Ada, the same parallel statement, i.e., `parallel do` and `parallel loop`, is in charge of spawning and distributing the computation among the executors (Figure 2a). OpenMP, instead, defines the `parallel` construct to spawn work, and several constructs (e.g., `for, task, taskloop`) to distribute this work to threads (Figure 2b shows how two parallel loops can run in parallel with each other).



(a) (b)

Figure 3. Tasklet and OpenMP forms of parallelism [8]

Interestingly, the work in [8] allowed verifying that the execution model and the memory model of both OpenMP and Ada tasklets are compatible, hence enabling the OpenMP runtime to be used to implement the Ada tasklet model, as well as to introduce additional functionalities, in particular for unstructured fine-grained parallelism (with fine-grained synchronization mechanisms such as task dependences) and heterogeneity (with the `target` construct for offloading synchronous and asynchronous tasks). Furthermore, [8] proves that the use of OpenMP functionalities not provided in the tasklet model allow enhancing the performance possibilities of non-embarrassingly parallel applications.

More recently this work has been extended [9] to show that compiler techniques can identify race conditions that can potentially appear in Ada programs parallelized with both OpenMP and/or Ada tasks. Moreover, this works can be used for checking the correctness of the OpenMP directives regarding dependence clauses and data-sharing attributes. Overall, this work enables to ensure safety in the presence of parallel computation.

To sum up, it has been proven that it is possible to provide Ada with two separate, but compatible, models:

- The tasklet model, to be used for homogenous regular parallelism, and that is included in the language core, as a key mechanism for parallelism.
- OpenMP, as an external model to the language, which can be used both as a runtime to support the tasklet model, as well as a complementary parallel programming model, providing more complex and flexible parallelism, and the support for heterogeneity.

For the latter, the integration of OpenMP and Ada would not be done in the Ada standard, but preferably as an additional language in the OpenMP specification.

Nevertheless, the integration of the Ada runtime with the OpenMP library requires understanding and controlling the interaction between Ada tasks and OpenMP threads, which raises several challenges [5]:

- Scheduling decisions. For instance, when an Ada task executing parallel code within the OpenMP runtime is preempted, the parallel computation should be also preempted, but OpenMP threads do not have the concept of priority (this concept is attributed to the tasks, in OpenMP). A mechanism needs to be provided that allows for communication between the two worlds.
- Protected objects access. An Ada tasklet, being executed by the OpenMP runtime, can access Protected Objects (POs), which requires the OpenMP library to be aware of the use of POs.
- Access to task attributes. Calls to Ada Task attributes need to use the context of the Ada Task, but can be made from within OpenMP threads.

Another aspect analyzed in [8] is the possibility of blocking and preempting Ada tasklets and hence OpenMP tasks. That work shows how OpenMP enables to mimic the use of blocking operations within tasklets by means of introducing task scheduling points (moments at which a thread can stop executing a specific task and start executing a different one). Nevertheless, the current proposal for the Ada standard [5] does not allow the use of blocking operations within parallel blocks (thus disabling preemption), so this is no longer an issue.

It is worth mentioning that there are ongoing discussions within the OpenMP community regarding the use of OpenMP in safety critical environments [10], and in particular the tasking model, which could be a good application domain for this integration.

Acknowledgements

This work was supported by National Funds through FCT (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (CEC/04234), the European Union's Horizon 2020 research and innovation programme under the grant agreement No 780622, and the Spanish Ministry of Science and Innovation under contract TIN2015-65316-P.

References

[1] S. Michell, B. Moore, L. M. Pinho, “Tasklettes – a Fine Grained Parallelism for Ada on Multicores”. International Conference on Reliable Software Technologies – Ada-Europe 2013, LNCS 7896, Springer, 2013.

[2] L. M. Pinho, B. Moore, S. Michell, “Parallelism in Ada: status and prospects”. International Conference on Reliable Software Technologies – Ada-Europe 2014, LNCS 8454, Springer, 2014.

[3] S. T. Taft, B. Moore, L. M. Pinho, S. Michell, “Safe Parallel Programming in Ada with Language Extensions”, Proceedings of the 2014 ACM SIGAda annual conference on High integrity language technology (HILT '14). ACM, New York, NY, USA, http://dx.doi.org/10.1145/2663171.2663181.

[4] L. M. Pinho, B. Moore, S. Michell, S. T. Taft, “An Execution Model for Fine-Grained Parallelism in Ada”, Proceedings of the 20th Ada-Europe International Conference on Reliable Software Technologies, Madrid Spain, June 22-26, 2015, http://dx.doi.org/10.1007/978-3-319-19584-1_13.

[5] AI12-0119-1/08, Parallel operations, http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai12s/ai12-0119-1.txt, last accessed April 2018

[6] S. Royuela, C. Martorell, X, E. Quiñones, L. M. Pinho, "OpenMP tasking model for Ada: safety and correctness", 22nd International Conference on Reliable Software Technologies (Ada-Europe 2017). 12 to 16, Jun, 2017, pp 184-200. Vienna, Austria. DOI: 10.1007/978-3-319-60588-3_12.

[7] OpenMP Architecture Review Board, “OpenMP Application Program Interface”, Version 4.5, November 2015, available at http://www.openmp.org/mp-documents/openmp-4.5.pdf, last accessed January 2018.

[8] S. Royuela, L. M. Pinho, E. Quinones, “Converging Safety and High-performance Domains: Integrating OpenMP into Ada”, In the Design, Automation, and Test in Europe conference (DATE). Dresden (Germany), March 19-23, 2018

[9] S. Royuela, X. Martorell, E. Quiñones, L. M. Pinho, “Safe Parallelism: Compiler Analysis Techniques for Ada and OpenMP”, 23rd International Conference on Reliable Software Technologies (Ada-Europe 2018). June, 2018, Lisbon, Portugal.

[10] S. Royuela, A. Duran, M. A. Serrano, E. Quiñones, X. Martorell, “A functional safety OpenMP for critical real-time embedded systems”, In the 13th International Workshop on OpenMP (IWOMP), New York (USA), September 18-19, 2017