CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Technical Report

Session Summary: Parallel Programming

Luis Miguel Pinho
Tullio Vardanega

CISTER-TR-181205

Session Summary: Parallel Programming

Luis Miguel Pinho, Tullio Vardanega

*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:

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

Abstract

Session Summary: Parallel Programming

Luis Miguel Pinho, *Session Chair*
Instituto Superior de Engenharia do Porto (ISEP-CISTER), Portugal
lmp@isep.ipp.pt

Tullio Vardanega, *Session Chair*
Università di Padova, Italy
tullio.vardanega@math.unipd.it

1 Introduction

The session centered on the discussion of three topics, each brought forward by a corresponding submission, backed by distinct participants in the Workshop:

1. Support for parallelism based on OpenMP, championed by Sara Royuela, Eduardo Quiñones, and Luis Miguel Pinho;
2. Synchronous signals, presented by Brad Moore;
3. Heterogeneous platforms, brought forward by Kristoffer Nyborg Gregertsen.

Before discussing the individual topics, Miguel and Brad provided a summary of the *status quo* for the model of parallelism that emanated from past editions of this Workshop and it is being considered by the Ada 202x language amendment process via the AIs listed in Table 1:

Table 1: Ada 202x AIs related with parallel programming

AI12-0119-1	Parallel blocks and loops	
AI12-0242-1	Reduce/Parallel_Reduce attributes	
AI12-0251-1	Explicit chunk index for parallel loops	dependent on AI12-0119-1
AI12-0251-2	Manual chunking operations	dependent on AI12-0119-1 and AI12-0266-1
AI12-0262-1	Map-reduce attributes	dependent on AI12-0242-1 and AI12-0212-1
AI12-0266-1	Parallel container iterators	dependent on AI12-0119-1
AI12-0267-1	Data race and blocking prevention	dependent on AI12-0064-2, AI12-0079-1, and AI12-0119-1

The general model that underpins those AIs is that the application code presents *potential opportunities for parallelism* (POPs), the compiler generates executable code for exploiting them, and the runtime schedules their execution. This model requires refining the notion of task, when it includes a parallel construct, to represent multiple logical threads of control that can proceed in parallel. Each such thread of control within a task is termed an *executor*, and the execution that it performs between synchronization points is termed a *tasklet*. When a task distributes its execution to a set of executors, it cannot proceed with its own execution until all the corresponding tasklets have completed. Those tasklets may synchronize by making protected operations, but cannot call blocking operations. The new Global (cf. AI12-0079-1) and Nonblocking (cf. AI12-0064-2) aspects may be used to facilitate the detection of such calls at compile time.

At the previous IRTAW this conceptual model was examined in the regard of the interaction of parallelism with tasking, determining the following semantics:

- Changing task attributes should be deferred until outside of the region of parallel execution;

- If parallel tasklets attempt to perform multiple changes on the same task attribute, one of them is selected arbitrarily to take effect;
- If multiple distinct operations are deferred during a parallel execution (such as, for example, a task attribute change and an exception), they should be applied as close as possible to what prescribed for them in sequential Ada;
- Whereas a per-CPU execution-time accounting would be desirable within parallel regions, the Workshop initially proposed a simpler model that only provides a per-task counter, which is updated at the end of the parallel region;
- Set_CPU and Get_CPU calls should be provided to specify CPUs where the tasklets of a task should execute.

2 Supporting Ada's parallelism with OpenMP

After recapping the current situation with the Ada 2020x AIs related with parallel programming, attention shifted to the first topic of discussion in the session agenda, i.e., the viability of supporting Ada's parallelism with OpenMP.

OpenMP is a widely used in the HPC domain and it is now also entering the embedded domain. The OpenMP solution has three parts: annotations (pragmas) applied to the user code to specify requirements for parallel execution; a compiler pass that acts on those annotations generating calls to the OpenMP runtime library; the runtime that manages parallel execution.

Guided in the discussion by Sara and Eduardo, the Workshop acknowledged that OpenMP's *tasking* model (with its unfortunate naming clash), when used in the classic fork-join semantics, maps quite well to Ada's *tasklet* model. In addition to that, however, OpenMP also allows for more flexible approaches to parallel programming, called "unstructured parallelism", which address scenarios where the progress of some execution within a parallel region is dependent on certain data-driven conditions to be met. OpenMP provides the pragma "task depend" to this end, to expresse out/in conditions for a particular flow of execution: for example, task A outputs an 'a' that task B uses, means that B cannot start before A completes. An intuition of how OpenMP's unstructured parallelism relates to Ada's strict fork-join model is depicted in Figure 1.

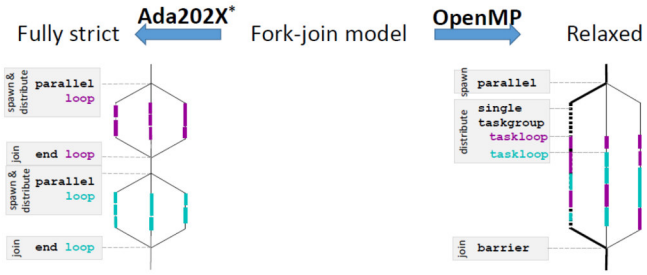

Figure 1: Ada's strict fork-join model vs. OpenMP's unstructured parallelism (picture taken from joint work by Sara, Miguel and Eduardo, presented at DATE 2018).

The Workshop acknowledged that, while very interesting, OpenMP's unstructured parallelism falls much outside of the scope of parallel programming in Ada 202x. With that notion in mind, the discussion moved on to the issue of how OpenMP's runtime could be delegated Ada's tasklet semantics, while staying under control of the Ada runtime for scheduling. In particular, the Workshop discussed what should happen when an Ada task executing a parallel region within the OpenMP runtime would be preempted. The problem here is that, as OpenMP's threads of control have no notion of priority, as do OpenMP's tasks, the Ada binding should propagate priorities down to the underlying Operating System threads by means of (scarcely attractive) ad-hoc extensions to the OpenMP runtime. Alternative mapping solutions were explored, but none was found to be convincing, especially when interaction with POs and task attributes were to be contemplated.

The conclusion of this very interesting discussion was that more work is needed to understand, from an OpenMP perspective, how to mix concurrency and parallelism. The IRTAW group should seek the opportunity to discuss with

the OpenMP community about this matter and strive to inject real-time concerns into it. There was consensus that this could be an interesting avenue to explore in the future.

3 Synchronous signals for parallel-sequential-parallel patterns of execution

Brad summarised the essence of his proposal on this point. In analogy with current Ada's Synchronous_Barriers library package added to the Real-Time Systems annex in 2012, Brad's idea was to provide a low-cost mechanism, named Synchronous_Signals, to allow sequential processing, within a concurrent unit, to interleave with parallel processing. One type of call to a Synchronous_Signals object would manage the transition from parallel to sequential code, and another call would manage the converse. In a region with N threads of control, one designated caller of the former API would wait until all other $N-1$ had made the same call, and then be able to proceed alone sequentially. On its call to the latter API, all of the N threads of control would resume parallel execution. The intent of this mechanism is to minimize the amount of synchronization in a parallel application and make better use of the available CPUs.

The Workshop concurred that the "parallel-sequential-parallel" sequence of execution is a common pattern in parallelism, which justifies looking at this problem. On the merit of Brad's proposal, the sentiment of the Workshop was twofold. On the one hand, it was felt that the naming of the mechanism was not appropriate, as the notion of signal is much overloaded. On the other hand, as the intended application semantics can be implemented with POs, it was felt that a faster runtime implementation would be an insufficient argument to justify a new language object, which instead would if HW support existed for the intended semantics.

However, the opportunity of investigating support for this feature via libraries (*à-la* Paraffin) was deemed useful and interesting because it allows user exploration of possible uses of the feature, while avoiding the need to define application-level syntax for it and solve all problems of interaction of it with tasklets. To this end, the Workshop recommended that Brad's Paraffin library should be updated with this and the other recent features, be uploaded to a public repository, and referenced from an Ada-Europe's page for the benefit of the general public.

4 Ada and heterogeneous processor architectures

The session concluded with Kristoffer reporting on his recent work on heterogeneous processor architectures, and his reflection on the role that Ada could have in them.

The first observation was that massive SMP architectures are not common outside of HPC. Much more frequent, instead, are heterogeneous architectures, where OpenCL, OpenACC or CUDA are the dominant software platforms. To argue this point Kristoffer presented a robotic use-case application that an ESA-funded project currently is developing. The gist of the argument was that it would be nice, in that systems context, that an Ada application would be able to command and control accelerators. The value added of it would be the better safety and reliability associated with the quality of the language. This integration might be achieved by a binding library, which would wrap an accelerator kernel, suitably determined by the corresponding cross-compiler, send it off to the target hardware, and eventually receive the result of that computation. Discussing possible implementation routes for this ambit, the Workshop concurred that there would be two principal options: to reuse existing support from e.g., OpenCL, but giving up on certification; or else to re-implement it all with Ada/SPARK. It is evident that the latter would be more attractive, but also much more costly to achieve.

In wrapping the session up on discussion of the rising importance of heterogeneous architectures, the Workshop noted that OpenMP's support for those types of systems increases the interest of exploring ways to integrate Ada with it.