www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

www.cister.isep.ipp.pt

CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

OpenMP tasking model for Ada: safety and correctness

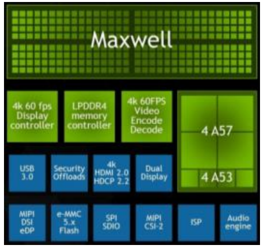
Sara Royuela, Xavier Martorell,
Eduardo Quiñones and Luis Miguel Pinho

Vienna (Austria) June 12-16, 2017

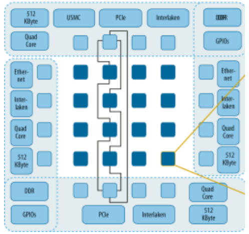
Parallel heterogeneous embedded architectures

1. Exploit its performance capabilities
2. Facilitate programmability
3. Ensure portability

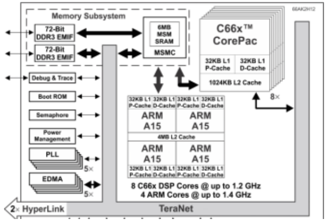
Parallel Programming Models



NVIDIA Tegra X1: 4-core A57 and A53, GPU (automotive)



Kalray MPPA: four 4-core K1, 256-core fabric (avionics)

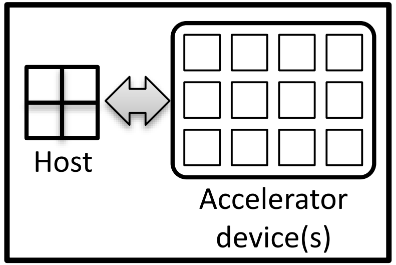


TI Keystone II: 4-core A15, 8-core DSP (industrial)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Parallel Programming Models

- Provides a **level of abstraction** to express parallelism while hiding processor complexities
  - Defines parallel regions and synchronization mechanisms
  - Couples host processor with accelerators devices (e.g., many-cores and DSP fabrics, GPUs, FPGAs)
- **Mandatory** as the number computing resources integrated increases



Generic parallel heterogeneous architecture

Ada and Parallel Programming Models

- There is a necessity to support parallelism in Ada capable of exploiting parallel heterogeneous embedded architectures

	Define an Ada's parallel model[1]	Adopt an existing parallel model
Pros	• Full control of the model • Incorporate safety issues in the model	• Very mature models • Portability • Develop "only" the Ada and parallel run-time connection
Cons	• Develop the complete parallel framework • Less portability	• No safety properties

[1] Michell, Moore, Pinho, ***Tasklets – a fine grained parallelism for Ada on multicores***, in Ada-Europe 2013
Pinho, Moore, Michell, Taft, ***Real-Time Fine-Grained Parallelism in Ada***, in ACM SIGAda Ada Letters 2015

Parallel Programming Model Challenges

- Productivity perspective (performance, programmability, portability)
  - Shared and distributed memory
  - Fine-grain task- and data-based parallelism
  - Heterogeneity
  - Load balancing
  - Efficient synchronization methods

- Safety perspective
  - Parallel programming is complex and error prone, compromising correctness and so safety
  - **Compiler and run-time techniques** to detect errors must be provided

OpenMP

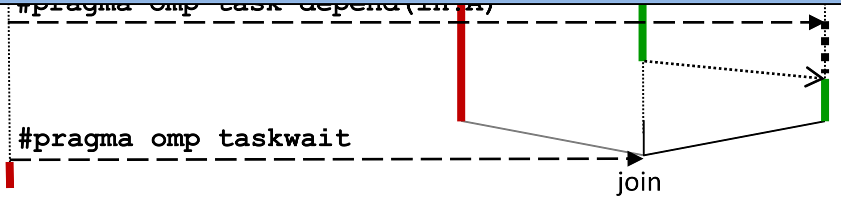
- **Mature language** constantly reviewed and augmented (last release Nov 2015)
- **Performance** and **efficiency**
  - Tantamount to other models (e.g. TBB, CUDA, OpenCL and MPI)
  - Support for fine-grain data- and task-parallelism
  - Features an advanced accelerator model for heterogeneous computing
- **Portability**
  - Supported by many chip and compiler vendors (Intel, IBM, ARM, TI, Kalray, Gaisler)
- **Programmability**
  - Currently available for C, C++ and Fortran (**`#pragma omp`**)
  - Allows incremental parallelization
  - Can be easily compiled sequentially (easing debugging)

OpenMP execution model

- **Fork-join** parallel model of execution
- **Task-centric** model

It is important not to confuse *OpenMP tasks, Ada tasks* and *Ada tasklets* are not the same thing

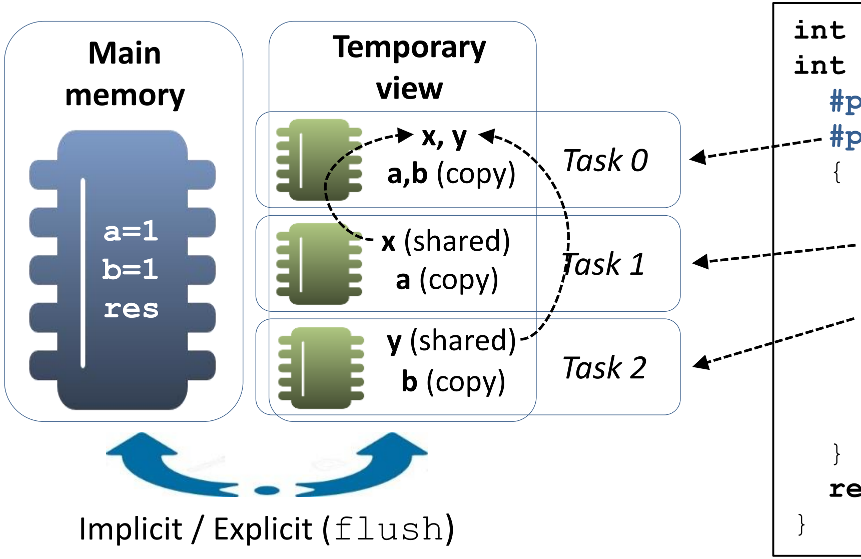
- Ada tasks are meant to exploit concurrency
- OpenMP tasks and Ada tasklets are meant to exploit fine-grain parallelism



OpenMP memory model

Relaxed-consistency memory model

Variables visibility defined by the programmer: **`shared`, `private`, `firstprivate`**



```
int a = 1, b = 1, res;
int foo() {
  #pragma omp parallel
  #pragma omp single
  {
    int x, y;
    #pragma omp task
     x = a*a;
    #pragma omp task
     y = b*b;
    #pragma omp taskwait
    res = x + y;
  }
  return res;
}
```

OpenMP and Safety

- Our vision is **that OpenMP enables to guarantee safety** requirements in terms of
  - Time predictability UpScale **(www.upscale.com)**
    - Reasoning about the timing behaviour of the parallel execution
  - Safety and correctness
    - Ensuring that the correct operation in response to its inputs
    - Support **reliability** and **resiliency** mechanisms

Compiler analysis techniques
for checking **correctness**

Error handling methodologies
to be added in the specification

Ada and OpenMP

Our proposal

1. **Extend OpenMP** to support Ada
2. **Extend Ada** to support OpenMP (e.g., including a new `pragma OMP`)
3. Add compiler and runtime mechanisms to ensure **correctness**

Example: Fibonacci computation

```
function Fibonacci(N: Integer) return Integer is
begin
  if N < 2 then
    return N;
  pragma OMP (parallel, shared=>X,Y,
                        firstprivate=>N);
  pragma OMP (single, nowait);
  begin
    pragma OMP (task, shared=>X,
                      firstprivate=>N);
      X:= Fibonacci(N - 2);
    pragma OMP (task, shared=>Y,
                      firstprivate=>N)
      Y:= Fibonacci(N - 2);
  end
  return X + Y;
end Fibonacci;
```

Fine-grained parallelism in Ada

	Blocks	Loops	Reductions	Tasks

Tasklet Model[1]	parallel
	  seq_of_stat_1
	and
	  seq_of_stat_2
	end parallel;
		for i in parallel lb..ub loop
		  seq_of_stat
		end loop;
			type t is new array
			  (parallel <>) of Float
			  with Reducer => "+", Identity => 0.0;
			Par_Sum : t := (others => 0.0);
			begin
			  for I in parallel Arr’Range loop
			    seq_of_stat
			  end loop;
			  Sum := Par_Sum(<>)’Reduced;
				-

OpenMP	pragma OMP (parallel);
	pragma OMP (single);
	begin
	  pragma OMP (task);
	    seq_of_stat_1
	  pragma OMP (task);
	    seq_of_stat_2
	end
		pragma OMP (parallel);
		pragma OMP (taskloop);
		for i in range lb..ub loop
		  seq_of_stat
		end loop;
			pragma OMP (parallel);
			pragma OMP (taskloop, reduction=>+,TOTAL);
			begin
			for i in range 0..MAX_I loop
			   seq_of_stat
			end loop;
			end
				pragma OMP (parallel);
				pragma OMP (single);
				begin
				  if cond then
				    pragma OMP (task);
				      seq_of_stat_1
				  else
				    pragma OMP (task);
				      seq_of_stat_2
				  end if;
				end

[1] Michell, Moore, Pinho, ***Tasklets – a fine grained parallelism for Ada on multicores***, in Ada-Europe 2013
Pinho, Moore, Michell, Taft, ***Real-Time Fine-Grained Parallelism in Ada***, in ACM SIGAda Ada Letters 2015

OpenMP challenges regarding Safety

✓ **Non-determinism**

2. Race conditions
3. Deadlocks
4. Fault tolerance

The specification must be restricted[1]

- Actions not defined by the specification, e.g.,
  - Directives/Clauses receives arguments out of range
    - E.g., **`num_threads(N)`**
  - Where and how some expressions have to be executed is not defined
    - E.g., the order in which the values of a reductions are combined
  - Compilers and runtimes are not forced to check the conformity of a program
    - E.g., the storage location specified in task dependencies must be identical or disjoint

[1] **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
Sara Royuela, Alejandro Duran, Maria A. Serrano, Eduardo Quiñones, Xavier Martorell

OpenMP challenges regarding Safety

✓ Non-determinism

2. **Race conditions**

3. Deadlocks

4. Fault tolerance

- Incorrect data scoping definition
- Incorrect usage of synchronization mechanism

x and **y** are not visible outside the tasks

A synchronization point is needed

```
int a = 1, b = 1, res;
int foo() {
  #pragma omp parallel shared(res) firstprivate(a,b)
  #pragma omp single
  {
    int x, y;
    #pragma omp task firstprivate(x, a)
     x = a*a;
    #pragma omp task firstprivate(y, b)
     y = b*b;

    res = x + y;
  }
  return res;
}
```

OpenMP challenges regarding Safety

✓ Non-determinism

✓ **Race conditions**

3. Deadlocks
4. Fault tolerance

- Incorrect data scoping definition
- Incorrect usage of synchronization mechanism

There exist compilation techniques capable of identifying (and solving) race conditions[1,2]

```
int a = 1, b = 1, res;
int foo() {
  #pragma omp parallel shared(res) firstprivate(a,b)
  #pragma omp single
  {
    int x, y;
    #pragma omp task shared(x) firstprivate(a)
     x = a*a;
    #pragma omp task shared(y) firstprivate(b)
     y = b*b;
    #pragma omp taskwait
    res = x + y;
  }
  return res;
}
```

[1] Royuela, Duran, Liao, Quinlan, ***Auto-scoping for OpenMP tasks***, in IWOMP 2012

[2] Lin, ***Static nonconcurrency analysis of openmp programs***, in IWOMP 2008

OpenMP challenges regarding Safety

- ✓ Non-determinism
- ✓ Race conditions
- ✓ **Deadlocks**

4. Fault tolerance

- OpenMP synchronization mechanisms might result in deadlocks

```
#pragma omp task
{
  #pragma omp barrier
  x = a*a;
}
#pragma omp task
  y = b*b;
```

Not all threads will execute it

- Possible solutions to avoid deadlocks
  - Check that programs are OpenMP conformant
  - Adapt already existing compiler methods to OpenMP[1]
  - Avoid OpenMP techniques in favor of Ada high-level concurrency mechanisms (e.g., protected objects)

[1] Kroening et. al. "**Sound static deadlock analysis for C/Pthreads**"

OpenMP challenges regarding Safety

- ✓ Non-determinism
- ✓ Race conditions
- ✓ Deadlocks
- ✓ **Fault tolerance**

- One major problem of OpenMP in safety environments is the lack of resiliency mechanisms
- Attemps to add error-handling mechanisms to the standard already exist[1]
  - Some proposals have already been adopted (cancellation constructs)

[1] Duran et. al., ***A proposal for error handling in OpenMP***
Wong et. al., ***Towards an error model for OpenMP***
Fan et. al., ***Exception handling with OpenMP in object oriented languages***

Conclusions

- There is a necessity to extend Ada with **fine-grained parallelism** to efficiently support parallel heterogeneous computing
- Our proposal: To adopt OpenMP as a parallel programming model for Ada
  - Very mature parallel programming model (20 years)
  - Performance, programmability and portability without jeopardizing **safety**
  - Parallel programming challenges regarding safety can be addressed

Future Work

- This can be complementary and compatible with the parallel Ada model
  - OpenMP tasks and Ada tasklets are similar
  - The interplay between Ada and OpenMP runtimes must be analyzed (e.g. OpenMP could be the runtime common to both)
- The Ada community **can influence** the OpenMP standard to address the challenges that impacts on safety

A proposal to extend OpenMP

A functional safety OpenMP for critical real-time embedded systems,

To be presented in the 13th International Workshop on OpenMP, celebrated in New York in September 18-19, 2017

Sara Royuela, Alejandro Duran, Maria A. Serrano, Eduardo Quiñones, Xavier Martorell

- Comments from reviewers
  - *“[..] the proposed extensions are a good step toward making the use of OpenMP in safety environments practical, and appear to provide real value [..]”*
  - *“[..] Even if OpenMP didn't care about embedded systems this analysis seems useful to help elucidate some of the issues inherent in the OpenMP specification [..]”*
  - *“[..] it is an interesting challenge for modification on current OpenMP [..] OpenMP ARB may consider this proposal [..]”*

www.bsc.es

www.cister.isep.ipp.pt

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

OpenMP tasking model for Ada: safety and correctness

Sara Royuela, Xavier Martorell, Eduardo Quiñones and Luis Miguel Pinho

For further information please contact:

sara.royuela@bsc.es
eduardo.quinones@bsc.es
lmp@isep.ipp.pt

Parallel programming models comparison

Type	Language	✓ Strengths	✗ Weaknesses
Hardware Centric	**Intel® TBB**	- Highly tunable - High-level (task concept)	- Portability - Mapping thread/core not part of the model
	NVIDIA® CUDA	- Highly tunable - Wrappers for many languages	- Low level (explicit data management) - Restricted to NVIDIA GPUs
Application Centric	**OpenCL**	- Automatic vectorization - Executes in host and accelerator	- Low level (explicit data management) - Full rewriting
	Pthreads	- Full execution control (thread concept) - Dynamic creation/destruction of threads	- Low level (reductions, work distribution, synchronization, etc. by hand)
Parallelism Centric	**OpenMP**	- High-level (task and data-flow concept) - Portable - Exploits parallelism at host and device	- No safety concepts

OpenMP challenges regarding Safety

Race conditions: general techniques

Dynamic	Analyze specific executions and possibly deliver false negatives There are algorithms capable of detecting at least one race when races are present
Static	Analyze all possibilities (NP-hard) and possibly deliver false positives Sound solutions exist for specific subsets of OpenMP • Fixed number of threads • Using affine constructs Also solutions for detecting non-concurrency
Hybrid	Combination of static and dynamic tool for more accurate results