Barcelona Supercomputing Center
Centro Nacional de Supercomputación

HPC Parallel Programming Models for Real-Time Control Systems

Eduardo Quiñones
{eduardo.quinones@bsc.es}

5th workshop on Real-time Control for Adaptative Optics
Paris, 25 October 2018

Agenda

- The importance of parallel programming models
- The OpenMP parallel programming model
  - Timing guarantees and functional correctness in OpenMP
  - The real-time control (RTC) loop

Towards the Convergence of High-Performance and Real-Time Computing Domains

High Performance Computing

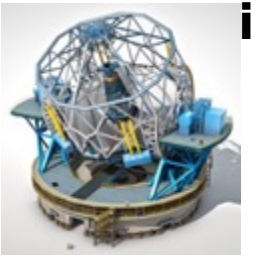
Computing Spectrum

Real-time Embedded Computing

Systems must operate **as fast as possible**

Converging of system irements

Systems must operate correctly in response to its inputs from a **timing** and **functional** perspective



Adaptiv e Optics

Autonomous driving

Towards the Convergence of High-Performance and Real-Time Computing Domains

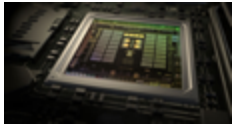
- **Parallel computing** is key to cope with performance requirements

HPC Domain (~300W)

NVIDIA TitanV
(5120 CUDA cores)

Intel Xeon Phi KNL
(72-core fabric)

Embedded Domain (~10-15W)



NVIDIA Tegra
(256 CUDA Cores)

Kalray MPPA
(256-core fabric)

Parallel Programming Models

- Mandatory to enhance **productivity**
  - ***Programmability.*** Provides an abstraction to express parallelism while hiding processor complexities
    - Defines parallel regions and synchronization mechanisms
  - ***Portability/scalability.*** Allows executing the same source code in different parallel platforms
  - ***Performance.*** Rely on run-time mechanisms to exploit the performance capabilities of parallel platforms
    - Including accelerators devices (e.g., FPGAs, GPUs, DSPs)

Parallel Programming Models

Conventional Models

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)

OpenMP

- **Mature language** constantly reviewed (last release Nov 2018, v5.0)
  - Defacto industrial standard in HPC shared memory processor architectures
- **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, NVIDIA, TI)
- **Programmability**
  - Currently available for C, C++ and Fortran (`#pragma omp`)
  - Allows incremental parallelization
  - Can be easily compiled sequentially (easing debugging)
- Very active research community

OpenMP

Sequential version

```
int x,y;
f1(&x,&y);
f2(x);
f3(y);
```

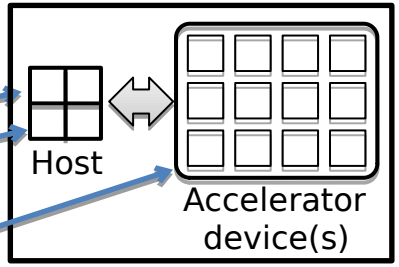
OpenMP version

```
#pragma omp parallel {
#pragma omp single{
  int x,y;
  #pragma omp task depend(out:x,y)
  { f1(&x,&y); }
  #pragma omp task depend(in:x)
  { f2(x); }
  #pragma omp target map(to:y) depend(in:y)
  { f3(y); }
}}
```

1. Start a new parallel region

2. Task executed on the host

3. Tasks executed on the host and accelerator when f1 completes



Generic parallel heterogeneous architecture

Principle behind OpenMP

- *Developers specify what the application does and not how it is done*
  - Computation is not fully controlled by the programmer but by the parallel framework
- **Complicates deriving timing analys and functional correctness!**

OpenMP can guarantee:

- ***Time predictability.*** Reasoning about the timing behaviour of the parallel execution
- ***Correctness.*** Ensuring a correct operation in response to its inputs

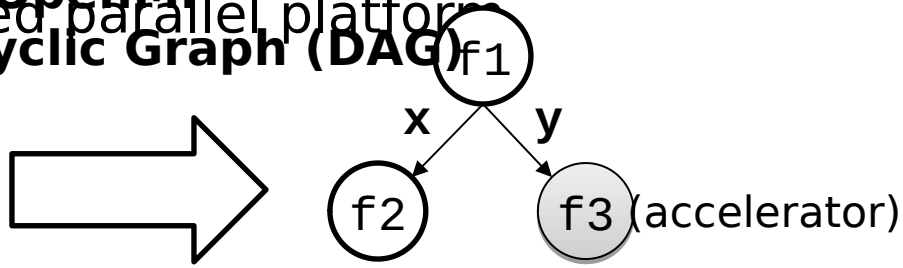
"I'm a software engineer, so I can confirm it works by magic."

Representation of Parallel Execution

- **Static analysis methods** to extract a complete representation of the parallel execution of OpenMP programs
  - Includes all the information for timing and functional correctness
  - Independent of the targeted parallel platform

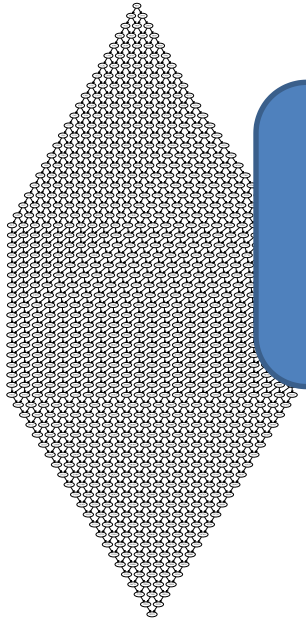
```
#pragma omp parallel{
#pragma omp single{
  int x,y;
  #pragma omp task depend(out:x,y)
  { f1(&x,&y); }
  #pragma omp task depend(in:x)
  { f2(x); }
  #pragma omp target map(to:y) depend(in:y)
  { f3(y); }
}}
```

OpenMP
Direct Acyclic Graph (DAG)



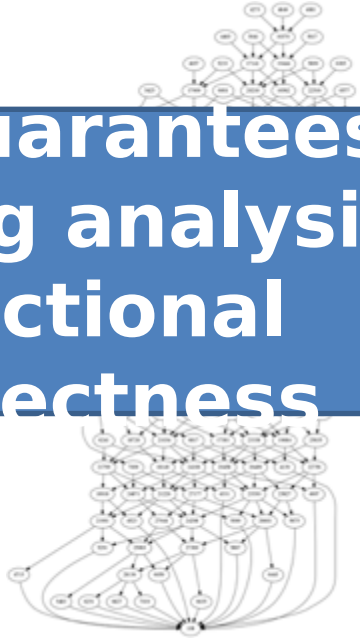
Platform Independent OpenMP-DAGs

Pedestrian detection (automotive)

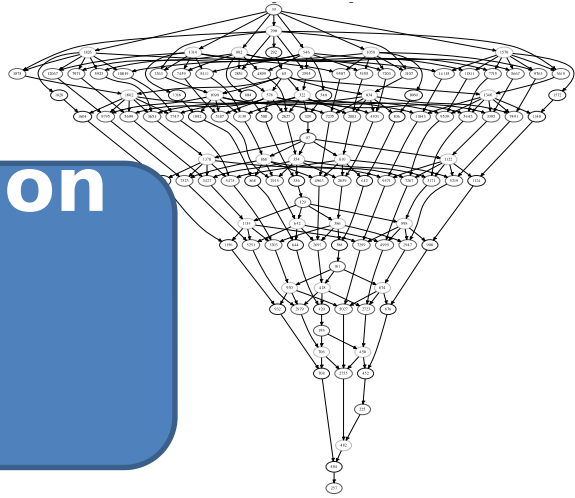


Infra-red sensor pre-processing (space)

3D Path Planning (avionics)



Cholesky

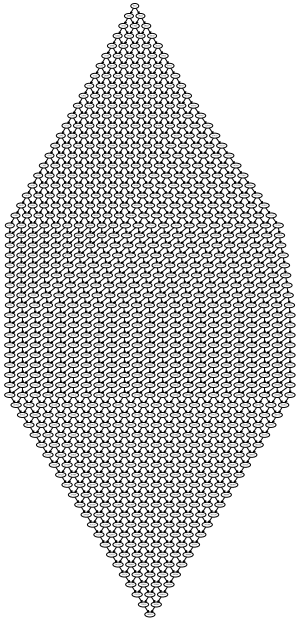


Provide guarantees on
1.Timing analysis
2.Functional correctness

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

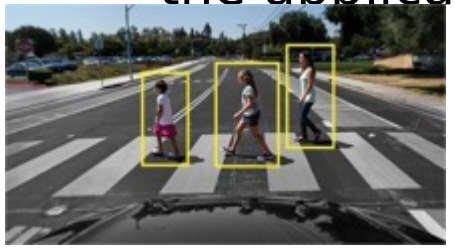
Timing Analysis

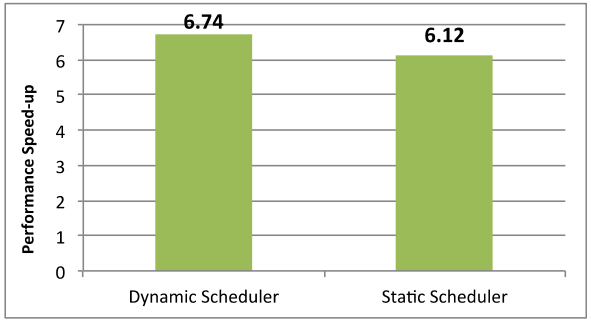
edestrian detector (automotive)

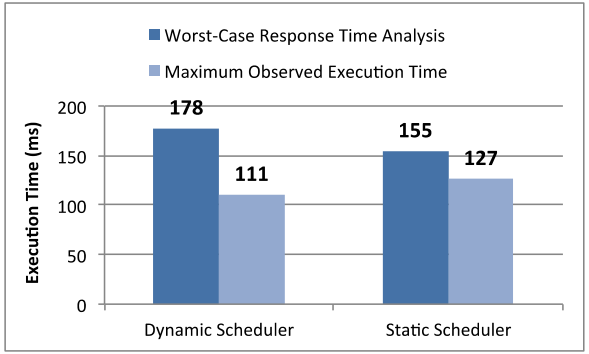


1. Dynamic and static allocation strategies of processor resources
2. Upper-bound the response time of the application

$$R_k^{ub} \leftarrow len(G_k) + \frac{1}{m}(vol(G_k) - len(G_k)) + I_k^p + I_k^{lp}$$





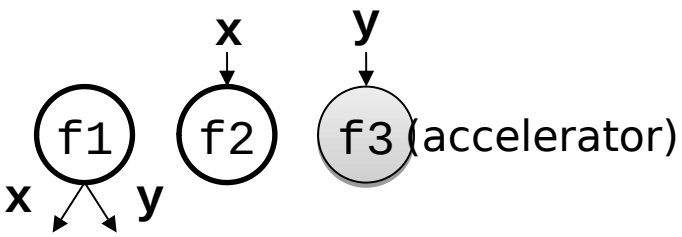


Platform dependent! (Intel 8-core processor)

Functional Correctness

- Race conditions
  - Incorrect data scoping definition and usage of synchronization mechanisms

```
#pragma omp parallel {
#pragma omp single{
  int x,y;
  #pragma omp task depend(out:x,y)
  { f1(&x,&y); }
  #pragma omp task depend(in:x)
  { f2(x); }
  #pragma omp target map(to:y) depend(in:y)
  { f3(y); }
}}
```

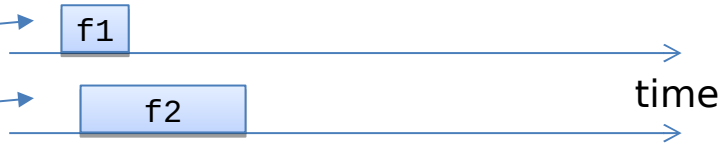


f2 and f3 may use incorrect values of x and y!

Real-time Control Loop (RTC)

- HPC Parallel programming models are time-agnostic

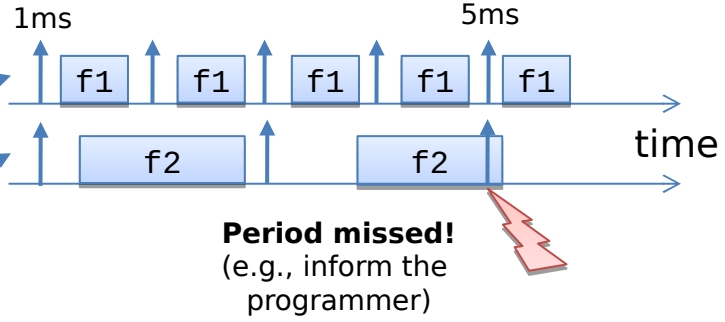
```
#pragma omp parallel {
#pragma omp single{
  int x,y;
  #pragma omp task
  { f1(&x); }
  #pragma omp task
  { f2(&y); }
}}
```



Real-time Control Loop (RTC)

- Parallel programming models are time-agnostic

```
#pragma omp parallel {
#pragma omp single{
  int x,y;
  while(1) {
    if (period == 1ms){
      #pragma omp task
      { f1(&x); }
    }
    if period == 2ms)
      #pragma omp task
      { f2(&y); }
    }
  }
}}
```




```
#pragma omp parallel {
#pragma omp single{
  int x,y;
  #pragma omp task period(1ms)
  { f1(&x); }
  #pragma omp task period(2ms)
  { f2(&y); }
}}
```

Conclusions

1. Parallel programming models are fundamental for productivity in terms of programmability, portability and performance
   - They do not provide timing guarantees and functional correctness
2. OpenMP is a very convenient model for converging productivity with timing guarantees and functional correctness
   - Extracting a representation of the parallel execution (OpenMP-DAG)
   - Introducing a RTC loop within the parallel framework
3. **We are working with the language standardization committee to enhance OpenMP**

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

HPC Parallel Programming Models for Real-Time Control Systems

Eduardo Quiñones
{eduardo.quinones@bsc.es}

5th workshop on Real-time Control for Adaptative Optics
Paris, 25 October 2018