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**

**Real-time Embedded Computing**

**Converging of system requirements**

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

- Autonomous driving
- Adaptive Optics
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 comparison

<table>
<thead>
<tr>
<th>Type</th>
<th>Language</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Centric</td>
<td>Intel® TBB</td>
<td>- Highly tunable</td>
<td>- Portability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- High-level (task concept)</td>
<td>- Mapping thread/core not part of the model</td>
</tr>
<tr>
<td></td>
<td>NVIDIA® CUDA</td>
<td>- Highly tunable</td>
<td>- Low level (explicit data management)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Wrappers for many languages</td>
<td>- Restricted to NVIDIA GPUs</td>
</tr>
<tr>
<td>Application Centric</td>
<td>OpenCL</td>
<td>- Automatic vectorization</td>
<td>- Low level (explicit data management)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Executes in host and accelerator</td>
<td>- Full rewriting</td>
</tr>
<tr>
<td></td>
<td>Pthreads</td>
<td>- Full execution control (thread concept)</td>
<td>- Low level (reductions, work distribution, synchronization, etc. by hand)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Dynamic creation/destruction of threads</td>
<td></td>
</tr>
</tbody>
</table>
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

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

OpenMP version

```c
#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
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 analysis 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
• **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

```c
#pragma omp data	int
#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); }
}
```
Platform Independent OpenMP-DAGs

Pedestrian detector
Infra-red sensor pre-processing
3D Path Planning
Cholesky

Provide guarantees on:
1. Timing analysis
2. Functional correctness
Timing Analysis

1. Dynamic and static allocation strategies of processor resources

2. Upper-bound the response time of the application

Platform dependent! (Intel 8-core processor)
Functional Correctness

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

```c
#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!
• HPC Parallel programming models are time-agnostic

```c
#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

```c
#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); }
}
```

Period missed!
(e.g., inform the programmer)

```c
#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
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