SOMA: An OpenMP Toolchain For Multicore Partitioning

E. Ruffaldi, G. Dabisias, F. Brizzi, G. Buttazzo

Scuola Superiore Sant'Anna

Pisa,Italy

ACM/SIGAPP Symposium on Applied Computing
April 6, 2016

Context and Motivations

Real-time systems are moving towards multicore architectures. The majority of multithread libraries target high performance systems.

- **Real-time** applications need **strict timing** guarantees and **predictability**.

Vs

- **High performance** systems try to achieve a lower computation time in a **best effort manner**.

There is no actual automatic tool which has the advantages of HPC with timing constrains.

Objectives

Starting from a **parallel C++** code, we aim to create:

- a way to **visualize** task concurrency and code structure as graphs.
- A **scheduling** algorithm, supporting multicore architectures and guaranteeing real-time constraints.
- A **run time support** for the program execution which guarantees the scheduling order of tasks.

State of the Art

StarPu[1]

- Parallelization tool over heterogenous resources.
- Scheduler.
- Drawback: no timing guarantee.

RT-OpenMP[2]

- Real-time OpenMP
- Drawback: mainly theoretical.

OMPSS[3](Barcelona Supercomputing Center)

- Asynchronous parallelism and data-dependency.
- Drawback: difficult to be extended.

[1] C. Augonnet, et al.. Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. Concurrency and Computation: Practice and Experience, 2011.

[2] D. Ferry, et al.. A real-time scheduling service for parallel tasks. In Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013.

[3] A. Duran et al. Ompss: a proposal for programming heterogeneous multi-core architectures. Parallel Processing Letters,2011.

Design Choices

Requirements

- Specification of the parallel tasks' structure.
- Specification of the real-time parameters.
- Tool to instrument the code.

Design Choices

Requirements

- Specification of the parallel tasks' structure.
- Specification of the real-time parameters.
- Tool to instrument the code.

OpenMP

- Standard in High Performance Computing.
- Minimal code overhead.

Clang

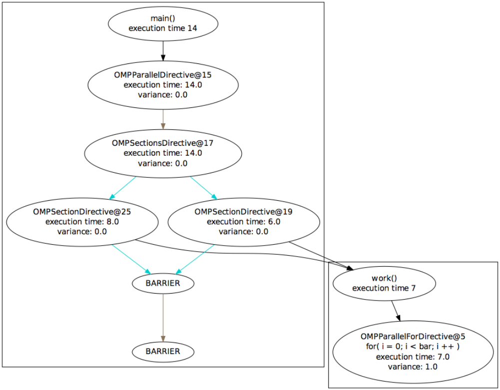
- Provides code analysis and source to source translation capabilities through **AST** traversal.
- Patched to support custom OpenMP pragmas: deadline and period.

Both are open source and supported by several vendors.

Basic Example

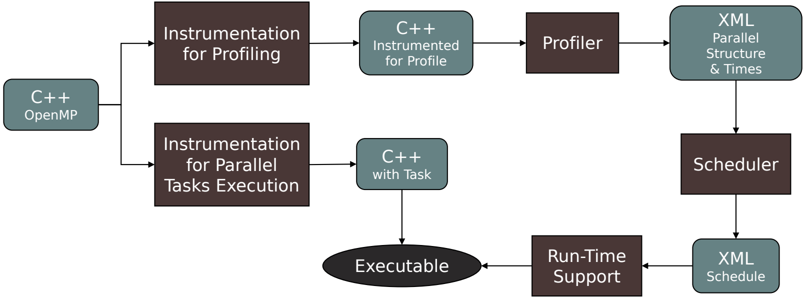
```
void work(int bar)
{
  #pragma omp parallel for
  for(int i = 0; i < bar; ++i)
  {
    //do stuff
  }
};
int main()
{
  int bar;
  #pragma omp parallel private
      (bar)
  {
  #pragma omp sections
  {
    #pragma omp section
    {
      //do stuff (bar)
      work(bar);
    }
    #pragma omp section
    {
      //do stuff (bar)
      work(bar);
    }
  } //implicit barrier
  } //implicit barrier
}
```

Parallel code structure



General Design

SOMA: Static OpenMP Multicore Allocator



Instrumentation for Profiling

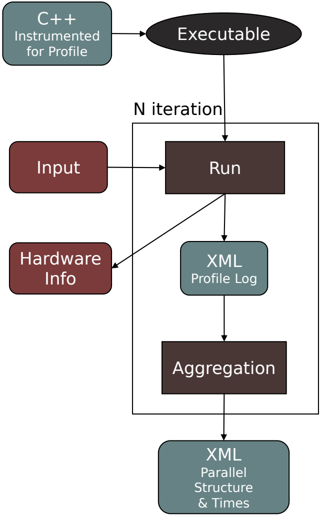
Custom profiler to time OpenMP code blocks and functions.

- Extracted information: **execution time**, **children execution time**, **caller identifier**, **for loop counter**.
- Output as XML file.

```
...
//#pragma omp parallel for
if( ProfileTracker profile_tracker = ProfileTrackParams(3, 5, bar - 0))
for (int i = 0; i < bar; ++i)
{
     //do stuff
}
...
//#pragma omp section
if( ProfileTracker profile_tracker = ProfileTrackParams(12, 25))
{
    //do stuff (bar)
    work(bar);
}
...
```

Profiling

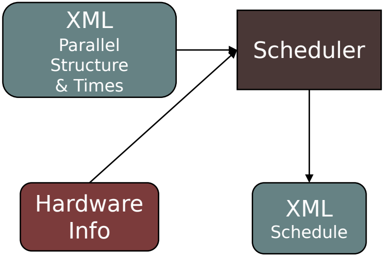
- The profiled code is executed N times and statistics are obtained.
- Profile statistics can be associated to different input arguments.



Scheduler

The input is the profiling XML with the tasks' deadline and period.

- The problem is *NP*-**complete**
  - all possible schedules have to be checked,
  - high computational load.
- It is possible to set a **fixed** amount of **computation time**.
- Scheduler **parallel version**: better results in a fixed amount of time.

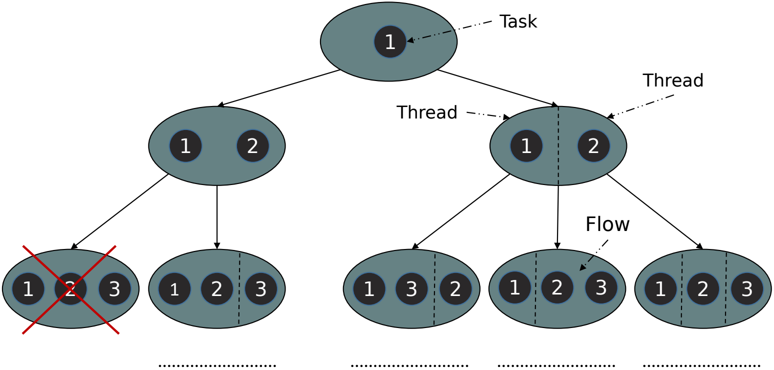


Output as XML file with the instructions for the real-time execution.

Scheduler: Algorithm

The scheduler assigns each task to a **flow** using a tree. Each flow will be allocated to a different virtual processor (thread).

- The algorithm splits each pragma *for* block.
- When a leaf is reached (complete schedule), the algorithm checks if the current solution is better then the previous one.



Scheduler: Feasibility

The produced schedule does not account for **precedence relations**.

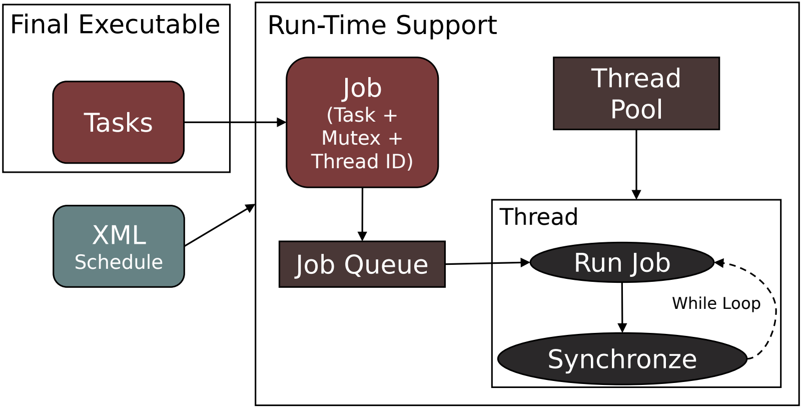
- Checking feasibility: modified version of Chetto&Chetto (1990).
- For each task we set :
  - the deadline starting from the last one;
  - the arrival time starting from the first and accounting for precedence relations.
- If all deadline are positive and each arrival time is less then the corresponding deadline the schedule is produced.

Instrumentation for Real-Time Execution

Pragma block $\longrightarrow$ Custom task.

- Pragma code block is embedded in a **function call**.
  - Nested function declaration not allowed in C++.
  - Declare the function in a **scoped class**.
- Out of scope variables are caught.
- The nested pragma structure is not changed.
- Each *for* statement is rewritten in order to allow it to be split.

Real-Time Execution



Test Objectives

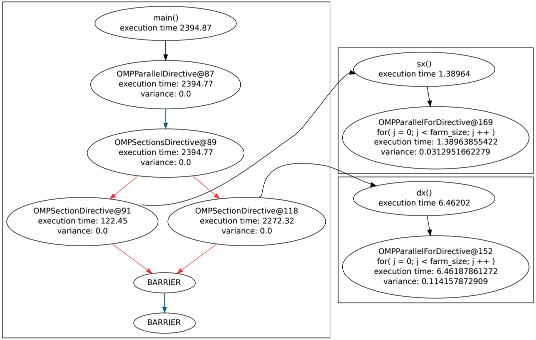
System framework evaluation

- Evaluate the instrumented program's **correctness**.
- Compare the OpenMP and SOMA completion time for **performance** evaluation.
- Measure framework's **overhead**.
- Check system's **predictability**.

Test Case

Face recognition algorithm in OpenCV using Multiscale Cascade Detector (Viola Jones algorithm).

- Input are two **stereo camera** videos.
- Frames are dispatched in blocks of N frames.



Results

- Test on an Intel i7@3.2 GHz with 6 cores and HT running Linux Kernel 3.8.0.
- Statistics are calculated over 5 executions.
- Tested with three different scheduler configurations: 4, 6 and 12 cores.
- Video properties:
  - 2 people in each.
  - 1 minute length.
  - 24 FPS.
  - Resolutions : 640x360, 1280x720, 1920x1080

Results: Execution Times

	Sequential	OpenMP		SOMA	
	$T_{seq}[s]$	$T_c(n)[s]$	$\epsilon(n) = \frac{T_{seq}}{nT_c(n)}$	$T_c(n)[s]$	$\epsilon(n) = \frac{T_{seq}}{nT_c(n)}$
480p(4)	750	195	0.96	195	0.96
720p(4)	3525	921	0.96	921	0.96
1080p(4)	8645	2271	0.95	2270	0.95
480p(6)	-	133	0.94	134	0.93
720p(6)	-	627	0.94	629	0.93
1080p(6)	-	1536	0.94	1539	0.94
480p(12)	-	98	0.64	92	0.68
720p(12)	-	427	0.69	426	0.69
1080p(12)	-	1043	0.69	1035	0.70

Results: Mean Service Time

Mean service time (gap between the delivery of a parsed image) in seconds.

- **SOMA variance < OpenMP variance**

	Sequential	OpenMP		SOMA	
	mean T_s	*mean* T_s	*mean var*	*mean* T_s	*mean var*
480p(4)	0.2823	0.2966	0.0014	0.2919	0.0004
720p(4)	1.3263	1.3955	0.0087	1.3884	0.0009
1080p(4)	3.2524	3.4399	0.0101	3.4369	0.0075
480p(6)	-	0.3038	0.0016	0.3023	0.0006
720p(6)	-	1.4241	0.0111	1.4206	0.0064
1080p(6)	-	3.4906	0.0238	3.4983	0.0197
480p(12)	-	0.4223	0.1421	0.4148	0.0044
720p(12)	-	1.9426	0.0862	1.9228	0.1334
1080p(12)	-	4.7394	0.3956	4.6915	0.6277

Results - Comments

All the results of the framework are **comparable** with OpenMP's.

- Almost **same performance**.
- SOMA has a lower service time variance $\rightarrow$ **more predictable**.
- **Low overhead** as OpenMP.

The framework achieved the two main requested properties to work with real-time applications.

- More realistic test cases will be tested.

Future Steps

Creation of **custom pragmas** and clauses.

- Too many pragmas
- No possibility to specify real-time constrains

Better **scheduler heuristics**.

- Save time by early pruning.

Implement a **probabilistic profiling** step.

- Some functions may not be called.

Add the possibility to extend the concept to **heterogeneous computing**.

Thank you!

- Questions?

Email:
fi.brizzi@sssup.it