MODELING OF MULTIPROCESSOR HARDWARE PLATFORMS FOR SCHEDULING ANALYSIS

Stéphane Rubini, Christian Fotsing, Frank Singhoff, Hai Nam Tran
Lab-STICC, University of Western Britany (UBO)
Contact: Stephane.Rubini@univ-brest.fr

Pierre Dissaux
ELLIDISS Technologies

Cheddar and SMART projects

AADL Comity, Toulouse, February 3rd 2014
Basic view of multi-processing scheduling

- Partitioned/Global scheduling

- Identical (a), uniform heterogeneous (b), or unrelated heterogeneous (c) processing units

(a) P1
    - C: 1

(b) P1
    - C: 1
    - C: 2

(c) P1
    - C: x
    - C: y

Different speeds

Different speeds (depend on the job)
Outline

• Scheduling analysis of multi-processing systems
  • Multi-processing implementations (shared memory)
  • Scheduling analysis concerns

• AADL modeling
  • Multi-processing systems
  • Homogeneous/Heterogeneous processors

• Cheddar and AADLinspector status
Multi-processing implementations and task scheduling

Processors, cores or physical threads may be seen as “standard” processing resources by the scheduler.

But the shared resources make the difference.
Scheduling Analysis Framework

Shared resources:

- Proc 1
- Proc 2
- Cache
- Memory
- Bus

Data

- S:10kb

Thread

- C:10ms

Scheduling analysis

Scheduling policy

Additional execution time

Proc 1: T1, T2
Proc 2: T1, T2
Task capacities

The capacity may depend on:

1. the execution unit (processor or memory speed) \(\rightarrow\) WCET analysis techniques, scheduling analysis
2. the sharing of resources (cache, bus) \(\rightarrow\) scheduling analysis
3. and the memory mapping. \(\rightarrow\) WCET analysis techniques with cache, scheduling analysis
Effective capacity

WCET analysis

P1

P2

C

M

Deployment (memory map, partition)

Scheduling analysis

Thread

Code

C: x ms

Speed A

Speed B

B1

Static

Dynamic

???
Outline

• Scheduling analysis of multi-processing systems
  • Multi-processing implementations (shared memory)
  • Scheduling analysis concerns

• AADL modeling
  • Multi-processing systems and shared resources
  • Homogeneous/Heterogeneous processors

• Cheddar and AADLinspector tools status
AADL Modeling of Multiprocessor Systems

- From scheduling analysis point of view, how to model for analyzing partitioned scheduling or global scheduling, on identical, or heterogeneous processors with a “realistic” behavior, i.e. considering implicit interferences between system entities?
Partitioned Scheduling

```
SYSTEM IMPLEMENTATION product.impl
SUBCOMPONENTS
  hard : SYSTEM soc_leon4::soc.asic_leon4;
  bank0 : MEMORY ram.ddr2;
  bank2 : MEMORY ram.ddr2;
  soft : PROCESS edgeDetection.impl;
PROPERTIES
  actual_processor_binding => (REFERENCE(hard.Proc_System.Core1)) APPLIES TO soft.getLine;
  actual_processor_binding => (REFERENCE(hard.Proc_System.Core2)) APPLIES TO soft.sharp;
  actual_processor_binding => (REFERENCE(hard.Proc_System.Core2)) APPLIES TO soft.edge;
  Scheduling_Protocol => (Rate_Monotonic_Protocol) applies to hard.core1;
  Scheduling_Protocol => (Rate_Monotonic_Protocol) applies to hard.core2;
END product.impl;
```
Task partitioning

Original AADL model

```
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
APPLIES TO soft.getLine;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
APPLIES TO soft.sharp;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
APPLIES TO soft.edge;
```

AADL model complemented by partitions

```
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
APPLIES TO soft.getLine;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
APPLIES TO soft.sharp;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
APPLIES TO soft.edge;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1))
APPLIES TO soft.getLine;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1))
APPLIES TO soft.sharp;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1))
APPLIES TO soft.edge;
```

Partitioning tools
Global Scheduling

Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),
REFERENCE(hard.Proc_System.Core2))
APPLIES TO soft.getLine;

Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),
REFERENCE(hard.Proc_System.Core2))
APPLIES TO soft.sharp;

Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),
REFERENCE(hard.Proc_System.Core2))
APPLIES TO soft.edge;

Scheduling_Protocol => Rate_Monotonic_Protocol applies to hard.Core1;
Scheduling_Protocol => Rate_Monotonic_Protocol applies to hard.Core2;

Actual_Processor_Binding has the inherit attribute; the binding may be applied at the container level

Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),
REFERENCE(hard.Proc_System.Core2)) applies to soft;
AADL consistency rules (AADLv2, p221)

From the AADL standard:

- (C2) In the case of dynamic process loading, the actual binding may change at runtime. In the case of tightly coupled multi-processor configurations, such as dual core processors, the actual thread binding may change between members of an actual binding set of processors as these processors service a common set of thread ready queues.

- (C5) A thread must be bound to a one or more processors. If it is bound to multiple processors, the processors share a ready queue, i.e., the thread executes on one processor at a time.
Specify the “global Scheduling_Protocol”

• The Scheduling_Protocol property can be applied to the component types processor or virtual processor.

• For global scheduling, the protocol must be the same for all the scheduled processors:
  ➔ append a consistency rule,
  ➔ or, allow the Scheduling_Property to be defined in a component system, and to be inherit by the processors of this system,
  ➔ or schedule on virtual processors, which are subcomponents of a processor where the scheduling protocol is defined.
Uniform processors

- Heterogeneous uniform processors: same capabilities, but different speeds.
- Effective capacity or variable processor speed?

« Binding related » execution times

```cpp
getline : THREAD t1 {
    period => 50us;
    Dispatch_Protocol => Periodic;
    Compute_Execution_Time => 2us..4us
    in binding (Processor_Core::Core.freq_500MHz);
    Compute_Execution_Time => 1us..3us
    in binding (Processor_Core::Core.freq_1GHz);
}
```

Or different processor speeds

```cpp
processor implementation core.freq_500MHz
properties
    reference_processor => classifier(core);
    scaling_factor => 0.75;
end core.freq_500MHz;
processor implementation core.freq_1GHz
properties
    reference_processor => classifier(core);
    scaling_factor => 1;
end core.freq_1GHz;
```
Cache sharing (architecture)

- Concurrent accesses to the shared cache → impact the WCET, even if cache partitioning techniques are used.
- Need to know the shared resources → AADL component hierarchy

WCET including cache intrinsic effect

Additional execution time due to concurrent accesses to the cache

```
memory cache | end cache;
processor core end core;
processor implementation core.impl
subcomponents
  cachel1 : memory cache;
end core.impl;

system dual_core end dual_core;
system implementation dual_core.impl
subcomponents
  c1 : processor core.impl;
c2 : processor core.impl;
cachel2 : memory cache;
end dual_core.impl;
```
Cache sharing (contents)

- Potential storage conflicts when tasks shared a cache.
- Instruction caches:
  - Relative memory locations of the task code
    - AADL properties: Base_address, source_code_size, memory_size
  - Code representation: explicit or abstract (CFG)
    - AADL: Source_Text, or reference to an external CFG representation
    - Cheddar: BasicBlock
  - Coding rules of CFG with BA?
- Data caches:
  - Shared data: private cache invalidation on writing (?)
    - AADL data components

WCET including cache intrinsic effect

T1 on P1

T2 on P2

Additional execution time due to concurrent accesses to the cache

Additional execution time due to storage conflicts
Core sharing (physical multi-thread)

- Architecture:
  - the threads share the first level cache;
  - fast context switching
- The `Scaling_Factor` value depends on the number of physical threads and on programs (operations, pipeline stalls).
- The “processors” inherit the properties `Scaling_Factor` and `Reference_Processor`.
- Context switching time may be quantified by the `Thread_swap_execution_time` AADL property.
Conclusion

• Modeling guideline
  • Multi-processor, multi-core or multi-thread are hardware implementation issues, but do not change the basics of multi-processing scheduling:
    ➔ use the processor entity to model the different kind of processing unit
    ➔ Include in processor private resources (i.e. caches, scratchpad memory)
  • Shared hardware resources between processing units must be appear as its own in the model (heavy impact on overall system performances) : bus, memory, cache
    ➔ Component hierarchy can represent at the same level the entities that interact directly.
SMART and Cheddar project status

- **Cheddar tool:**
  - Partitioned and global scheduling (without hardware interferences)
  - Tasks partitioning: basic algorithms (*-fit), studies about a framework to express and integrate optimization heuristics
  - Cache Preemption Related Delay analysis

- **AADLInspector tool:** multi-processor support in development.