INTEGRATED MULTI-PROCESSING SYSTEMS

Stéphane Rubini, Frank Singhoff
Lab-STICC, UBO, Brest, France

AADL standards meeting, April 27th, 2015

UBO université de bretagne occidentale
Lab-STICC

Integrated Multi-Processing Systems

- Multi-core and many-core
  - Freescale T4240: 12 Power ISA dual-threaded cores
  - Kalray MPPA-256: 256 VLIW processors
- MPSoC (Multi-Processing System On Chip): integrate processing units and devices (communication resources)
- Current trend to increase the computing performances
  - Effort to increase sequential processor speed is high
  - Adapted to SWAP requirements (Size, Weight and Power) of embedded systems.
- Multi-processing execution platforms seem to be well adapted to IMA requirements.
- But, barriers for using them in critical systems are:
  - Hardware interference channels: processing units are not fully isolated even if no software dependency exits.
  - Documentation: lack of precise specifications and descriptions of the internal of the chip.

Multi-processing: Architecture Classification

Based on the communication paradigm

1. Shared-memory, Unified Memory Access (UMA)

   Tightly coupled, communication through variables or shared memory segments

   →multi-cores

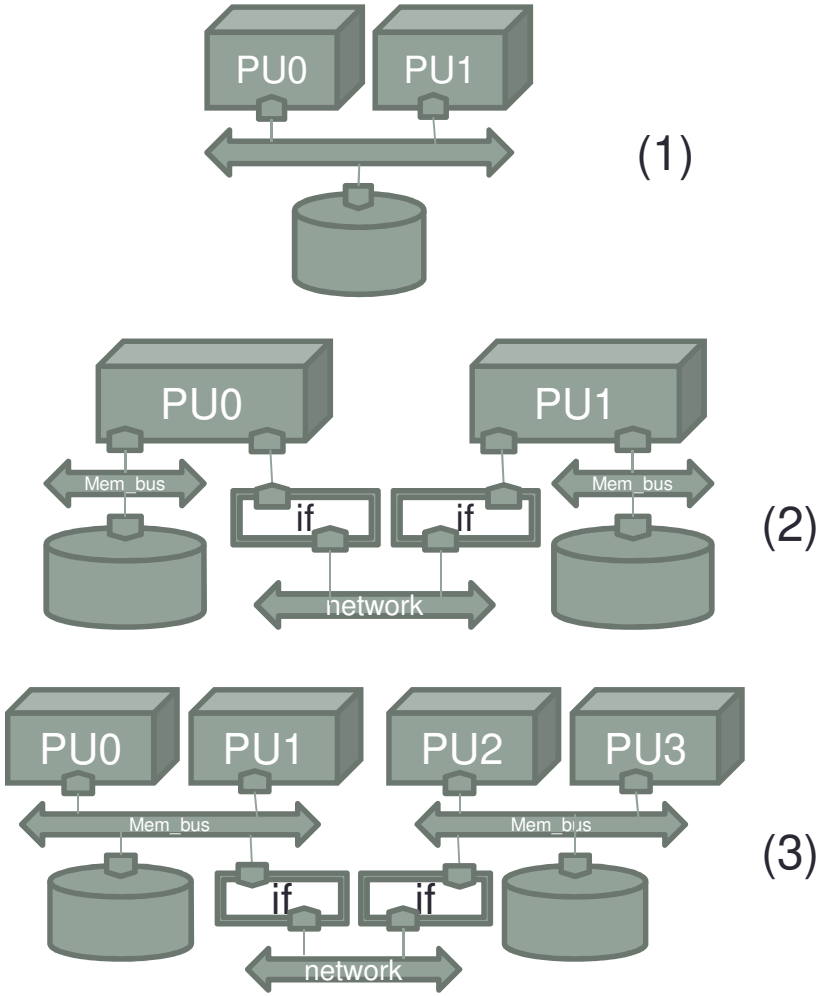
2. Distributed

   Loosely coupled, messages, communications through explicit input/output operations

3. Shared-memory, Non Unified Memory Access (NUMA)

   Communication through variables or shared memory segments

   → many-cores



Outline

- Enumeration and identification of the processing units
  - Processor characterization
- Software deployment
  - Thread scheduling
  - Processor groups
- Hardware interference channels
  - Memory hierarchy modeling
- TSP configurations
- Further works

Enumeration of the processing units

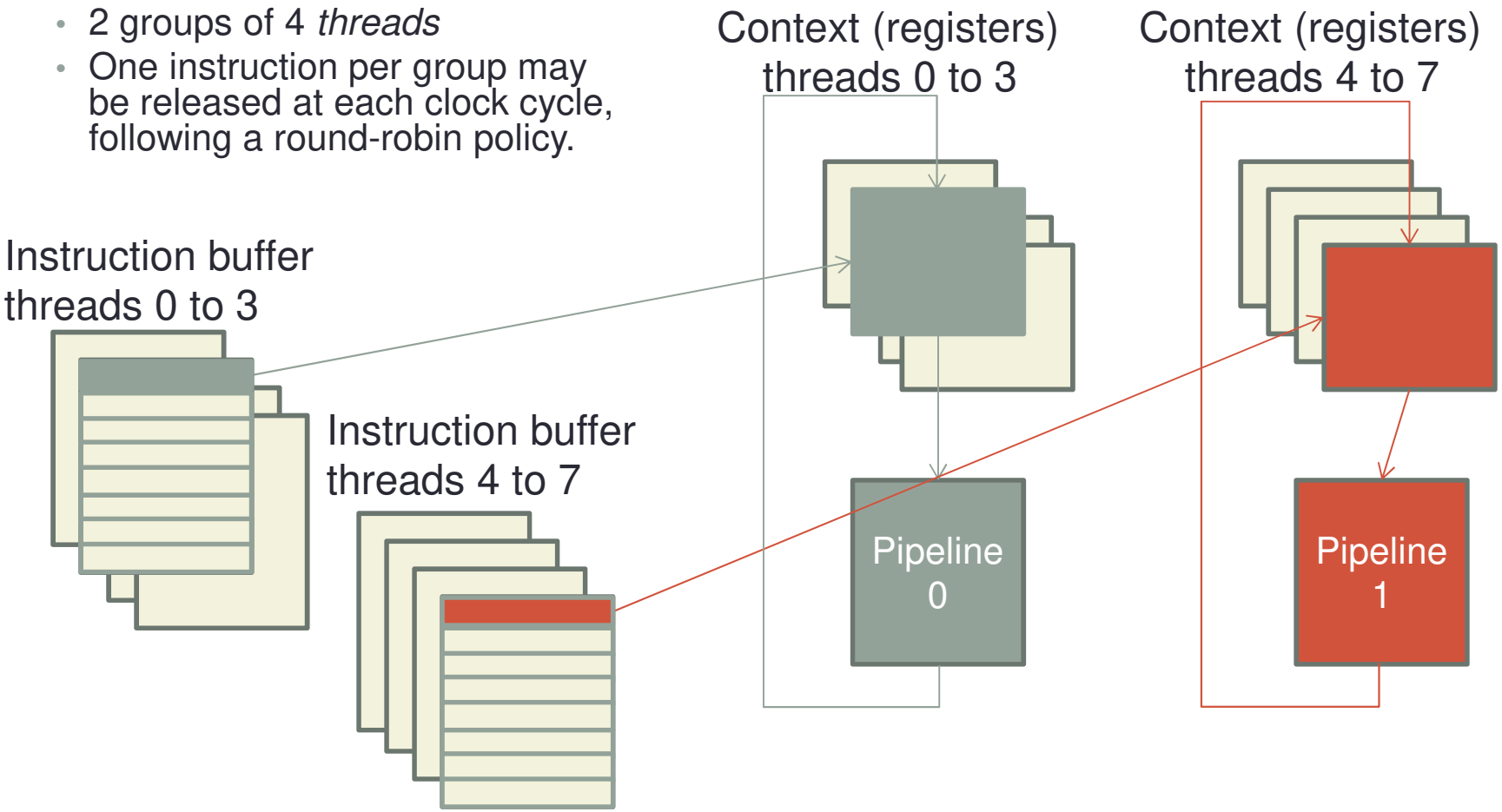
- From the AADL standard : “A processor is an abstraction of hardware and software that is responsible for scheduling and executing threads and virtual processors that are bound to it.”
- Additional semantics in multi-processing context
  - A processor is able to execute zero or one sequential flow of control at a time, and to store the architectural state associated to this execution. The execution flow of a processor is called later "physical thread".
  - 0 or 1 software thread or virtual processor is executed at a time by a physical thread. The scheduler or runtime of a processor is responsible to select the current software thread to be executed by the physical thread.

Multi-processing system modeling

- A physical thread, is modeled as an AADL processor. The hardware supporting the execution of the physical thread may be implemented as:
  - a sequential mono-threaded processor;
  - or a sequential mono-threaded core, part of a multi-core or many-core processor;
  - or a physical thread in a multi-threaded core or processor.
- A *system* component hierarchy captures the hardware structure of the execution platform. A system contains the processors and/or the processing sub-systems, and the set of hardware resources they share (bus, memory, device).

Example: Oracle OpenSPARC T2 (1)

- 8 multi-threaded cores
- 8 physical threads per core
  - 2 groups of 4 *threads*
  - One instruction per group may be released at each clock cycle, following a round-robin policy.



Example: Oracle OpenSPARC T2 (2)

- 64 physical threads
- 2-level system hierarchy
- The L1 caches are shared by the physical thread of a core

```
system smp end smp;
system implementation smp.t2
subcomponents
  cores : system core.t2 [8];
  L2_cache : memory cache;
end smp.t2;

system core end core;
system implementation core.t2
subcomponents
  physical_thread : processor processing_unit.sparc [8];
  L1_inst_cache : memory cache;
  L1_data_cache : memory cache;
end core.t2;

processor processing_unit
end processing_unit;
processor implementation processing_unit.sparc
end processing_unit.sparc;
```

Additional properties

- Physical thread implementation

  physical_thread_implementation : enumeration (physical_thread,
  processor_core,
  mono_core_processor,
  hardware_acellerator);

- Physical thread Identification ?

  Physical_Thread_Id: aadlinteger;

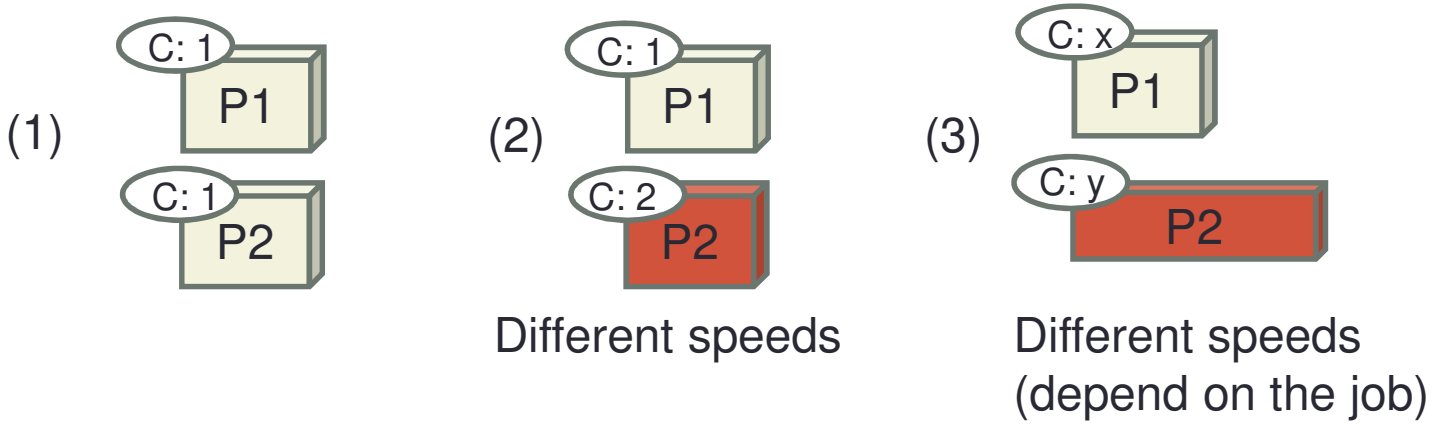
  Consistency rule: the identifier is unique for the set of the processor sub-components contained in a system component. (?)

Outline

- Enumeration and identification of the processing units
  - Processor characterization
- Software deployment
  - Thread scheduling
  - Processor groups
- Hardware interference channels
  - Memory hierarchy modeling
- TSP configurations
- Further works

Processor Characterization

- In multiprocessor scheduling analysis, a usual classification distinguishes 3 kinds of systems:
  1. identical processors: the capability and the speed of the processors are the same;
  2. uniform processors: they have the same capabilities (i.e. same ISA), but their execution speed are different;
  3. heterogeneous processors: they are not able to execute the same code, their ISA is different.



Processor Characterization: capabilities

- Capabilities
  - Binary code compatibility

    → new AADL property

    Instruction_Set_Architecture: aadlstring
    applies to (processor, virtual processor, thread, subprogram);
    - Consistency rule: if defined, the *Instruction_Set_Architecture* of threads or subprograms must be the same than the processor or the virtual processor used to execute them.

  - Memory space isolation

    → New AADL property
    - Define whether a processor may have its own private addressing space
    - Related to the capability of hardware to isolate thread's memory access flows (virtual memory context, address space numbers, ?)

    Implement_Runtime_Protection: aadlboolean applies to processor;
    Implement_Runtime_Protection: inherit aadlboolean
    applies to virtual_processor;
    - Consistency rule: two processor components cannot be bound to different processes, if *runtime_protection* is required by at least one of these processes and *Implement_Runtime_Protection* is false for at least one processor.

Processor characterization: speed

- Speed performances
  - Processor level

Clock_Period,: Time;
Clock _Period_Range: Time_Range
Reference_Processor: inherit classifier (processor);
Scaling_Factor : inherit aadlreal;

Intel Core 2
2.8 GHz,
SPECnt 2006 benchmark

Program	Dell motherboard	Intel motherboard	diff
gcc	12.1	17.5	30 %
mcf	25.5	25.4	0 %

What's the "Intel Core 2" scaling factor? 12, 17, 25?

New property proposal:

Peak_MIPS: addinteger;

(This is the worst mean to express the speed of a processor, but everyone knows the limit of that metric!)

DVFS example

The property *Scaling_Factor* quantifies the slow down due to a processor frequency change

```
system implementation dvfs.i
subcomponents
  soft      : process   as.i;
  cpu       : processor isa.i;
  freq_ctrl : process   monitor;
connections
  fs : port freq_ctrl.change_frequency -> cpu.change_frequency;
properties
  Actual_Processor_Binding => reference(cpu) applies to soft.t1;
  Reference_Processor => classifier(isa);
end dvfs.i;

process as end as;
process implementation as.i
subcomponents
  t1 : thread task { Compute_Execution_Time => 5 ms .. 10 ms;};
end as.i;

processor isa
features
  change_frequency : in event port;
end isa;
processor implementation isa.i
modes
  full_rate : initial mode;
  half_rate : mode;
  full_rate -[ change_freqency ]-> half_rate;
  half_rate -[ change_freqency  ]-> full_rate;
properties
  Scaling_Factor => 0.7 in modes (half_rate);
  Scaling_Factor => 1   in modes (full_rate);
end isa.i;
```

Processor Characterization: speed (2)

- Runtime/processor level:

  Thread_Swap_Execution_Time: Time_Range;

  Process_Swap_Execution_Time: Time_Range;

  Clock_Jitter: Time;

- Thread level : binding specific execution time (*in binding*)

  Compute_Execution_Time: Time_Range

- But, all these metrics are established for an “isolated processor”, not for the case of multi-processing systems.
  - Research topics: Interference-Sensitive WCET Analysis (ECRTS’14)

Outline

- Enumeration and identification of the processing units
  - Processor characterization
- Software deployment
  - Thread scheduling
  - Processor groups
- Hardware interference channels
  - Memory hierarchy modeling
- TSP configurations
- Further works

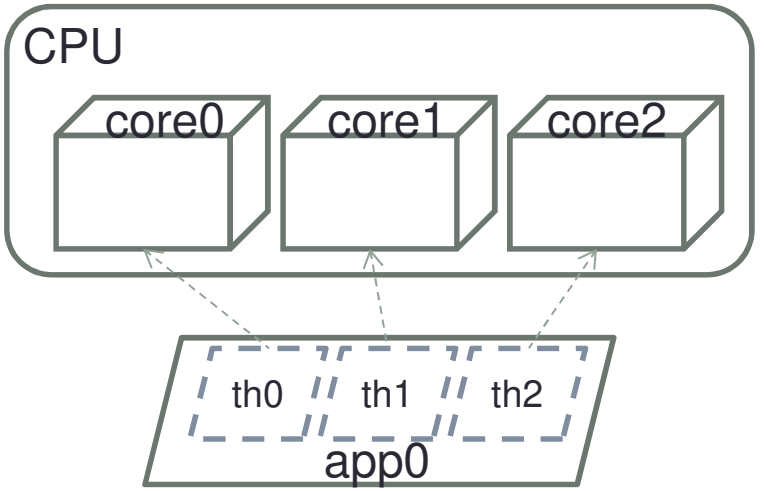
Software Deployment

- Asymmetrical versus Symmetrical
- Asymmetric Multiprocessing (AMP): each processor runs a separate OS/runtime.
  - The whole system looks like a distributed one,
  - Application codes create the communication channels.
- Symmetric Multiprocessing (SMP): A unique operating system controls several processors.
  - The SMP OS/runtime knows the parallel nature of the system.

AMP/SMP: scheduling point of view

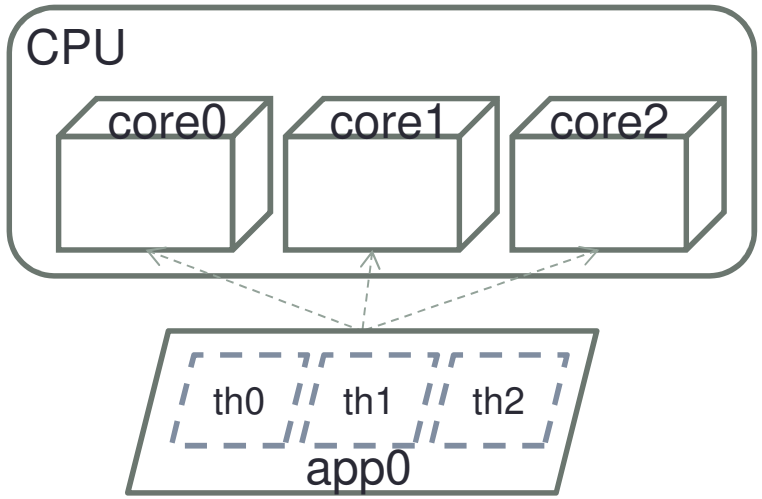
- SMP – partitioned scheduling

```
Actual_Processor_Binding => reference(cpu.core0)
        applies to app0.th0;
Actual_Processor_Binding => reference(cpu.core1)
        applies to app0.th1;
Actual_Processor_Binding => reference(cpu.core2)
        applies to app0.th2;
Scheduling_Protocol => RM
        applies to cpu.core0, cpu.core1, cpu.core2;
```



SMP – global scheduling

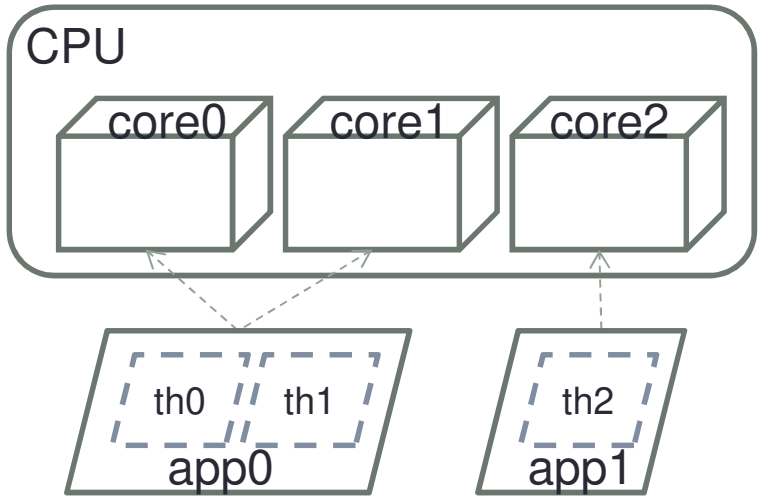
```
Actual_Processor_Binding =>
        reference(cpu.core0),
        reference(cpu.core1),
        reference(cpu.core2)
        applies to app0;
Scheduling_Protocol => RM
        applies to cpu.core0, cpu.core1, cpu.core2;
```



AMP/SMP: scheduling point of view

AMP(+SMP – global scheduling)

```
Actual_Processor_Binding =>
  reference(cpu.core0), reference(cpu.core1)
            applies to app0;
Actual_Processor_Binding => reference(cpu.core2)
            applies to app1;
Scheduling_Protocol => RM
            applies to cpu.core0, cpu.core1;
Scheduling_Protocol => HPF
            applies to cpu.core2;
```



Comments on SMP/AMP configurations

- An SMP OS has multiple instances of the same configuration data in the set of processors that it controls.
  - Ex: in global scheduling, Scheduling_Protocol is located in all processors.
- Need for
  - Either a consistency rule on some property values
  - or a mean to factorize “SMP system scope” properties.

Outline

- Enumeration and identification of the processing units
  - Processor characterization
- Software deployment
  - Thread scheduling
  - Processor groups
- Hardware interference channels
  - Memory hierarchy modeling
- TSP configurations
- Further works

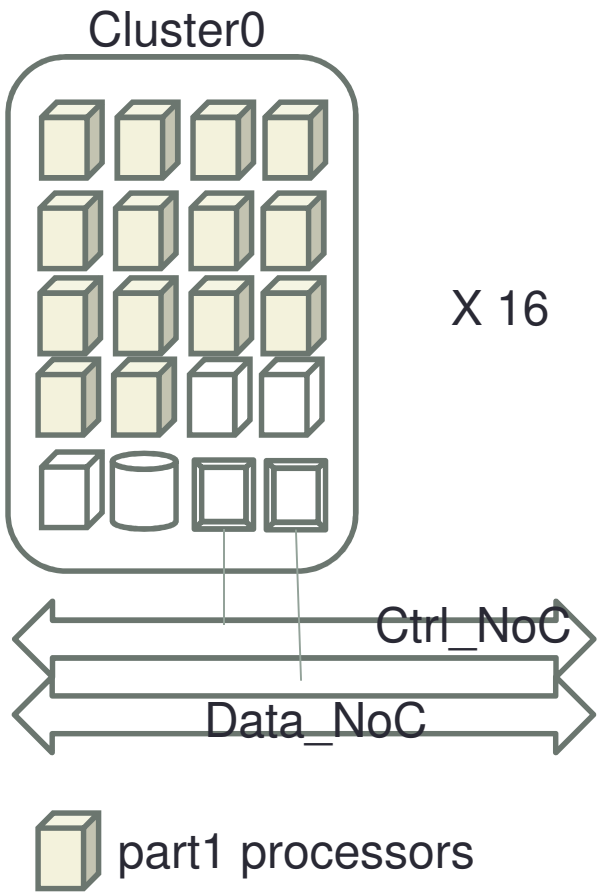
Processor Group

- Need for grouping processors.
  - Shared features between all the units of a “processor chip” for instance
  - Definition of sets of processors dedicated to an application
- Using the processor component as a “processor container”?
  - Do the container have its own execution unit? Its own state? If not, the solution leads to a contextual interpretation of the processor component.
  - What is the hyper-period of the system?
  - Risk of virtual processor/“sub-processor” confusions.
  - The current version of the standard explicitly describes this usage for the system component (semantic rule 3).
  - Systems-On-Chip integrate a lot of devices. “Chip” frontiers do not help to structure the model.

“processor group” component?

- A processor group represents an organizational component to logically group processors contained in systems.
- Property associations of processor groups are inheritable by the inner subcomponents.
- Usage
  - SMP operating systems: set of controlled execution units.
  - Global scheduling: set of a computing resources for the scheduler
  - Physical implementation modeling (cores within the same processor chip), physical threads in a same core
  - Definition of global properties (scheduling protocol, clock frequency, assign time)

Example: many-core Kalray MPPA-256



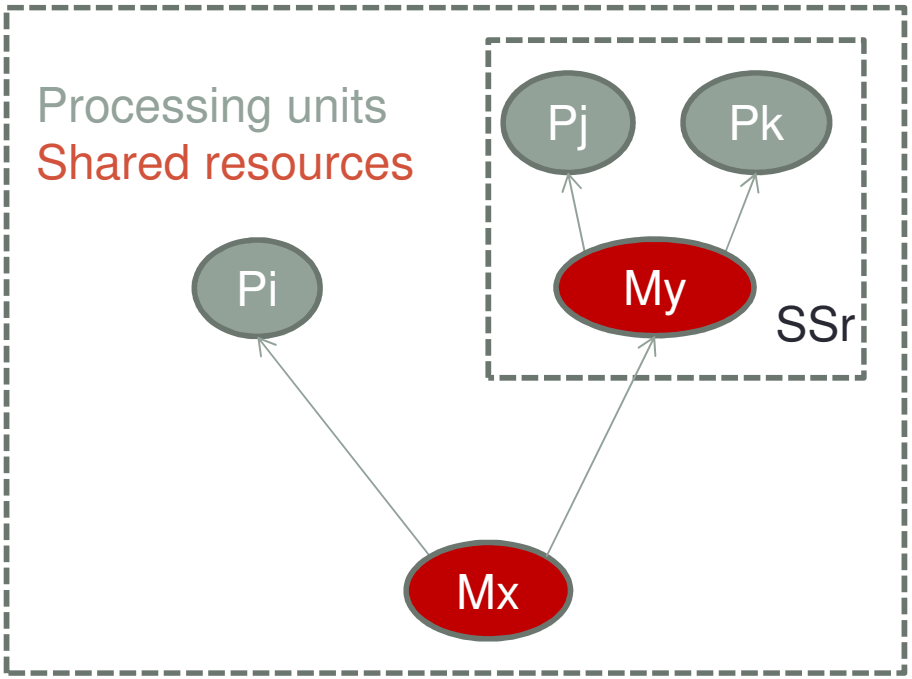

```
system implementation compute_cluster.mppa_256
subcomponents
  PEs : processor group PE_part.cluster;
  shared_memory : memory;
  app1 : process;
  ...
properties
  Actual_Memory_Binding => reference(shared_memory)
                                   applies to PEs;
  Actual_Processor_binding => reference(PEs.part1)
                                   applies to app1;
end compute_cluster.mppa_256;
processor group PE_part end PE_part;
processor group implementation PE_part.smp_os1
subcomponents
  PE0 : processor PE.mppa_256;
  ...
  PE13 : processor PE.mppa_256;
properties
   Scheduling_Protocol <= HPF;
end PE_part.smp_os1
processor group implementation PE_part.cluster
subcomponents
   part1: processor group PE_part.smp_os1;
   PE14 : processor PE.mppa_256;
   PE15 : processor PE.mppa_256;
properties
   Clock_Period => 2.5 ns;
end PE_part.cluster;
```

Outline

- Enumeration and identification of the processing units
  - Processor characterization
- Software deployment
  - Thread scheduling
  - Processor groups
- Hardware interference channels
  - Memory hierarchy modeling
- TSP configurations
- Further works

Hardware Interference Channels

- In multi-processing systems, the hardware interference channels come from the shared resources (memory, bus ,network, device).
- Memory hierarchy is the main interference channel in shared memory multi-core.
- Modeling guideline of memory hierarchy
  - The node's children represent the entities that interact directly within that node (shared resources).




```
system S
  Mx: memory mem.i1;
  Pi : processor pu.i1;
  SSr : system mpu.i1;
end S;
system SSr
  Pj : processor pu.i2;
  Pk : processor pu.i2;
  My : memory mem.i2;
end SSr;
```

Memory hierarchy modeling

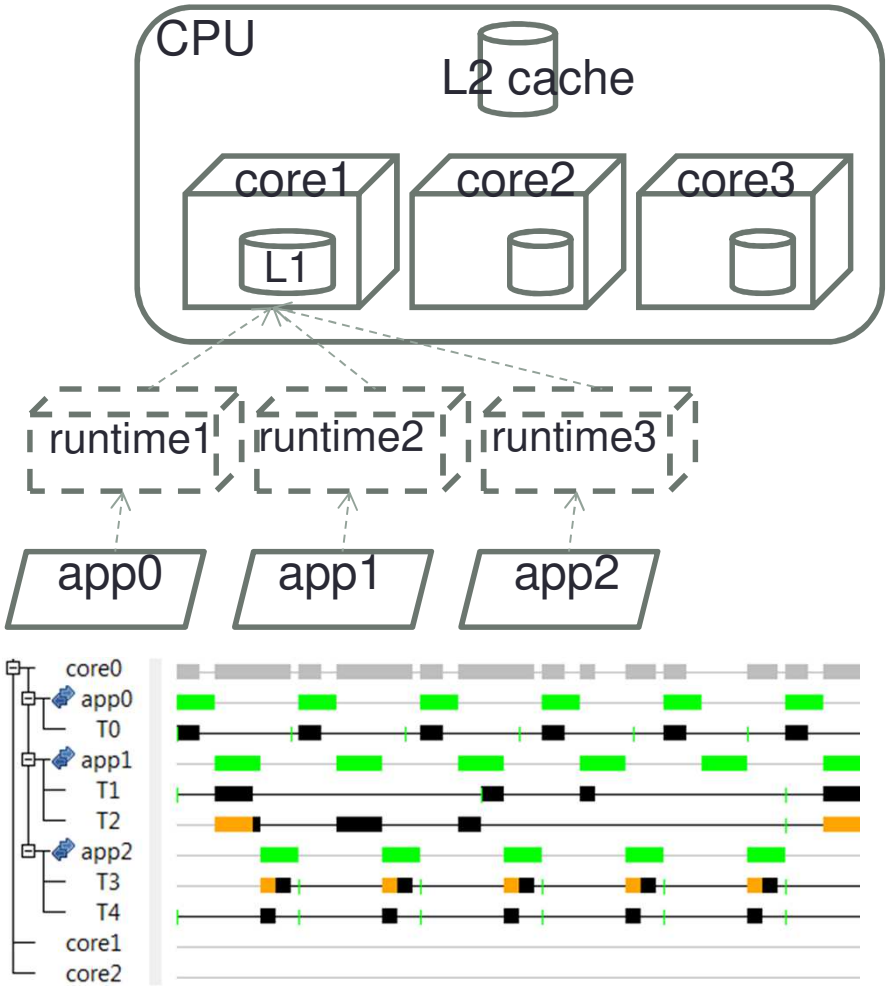
- The memory entities used by a same set of processing units are declared in a *system* component, as sub-components. If the memory entity is unique, a single *memory* component may be substituted to the *system* component.
- A *system* component groups, as sub-components, a memory system, and the processors and upper level memory systems which share the access to this memory system.
- For the sake of simplicity, levels of memory hierarchy connected to only one processor, i.e. processor's private caches, may be modeled as sub-components of this *processor* component.
- The memory system associated to the root node contains at least a *memory* component representing the main memory.
- TODO: integrate processor groups in the guideline. Are the shared memories inside or outside the processor groups?

Outline

- Enumeration and identification of the processing units
  - Processor characterization
- Software deployment
  - Thread scheduling
  - Processor groups
- Hardware interference channels
  - Memory hierarchy modeling
- TSP configurations
- Further works

TSP configurations

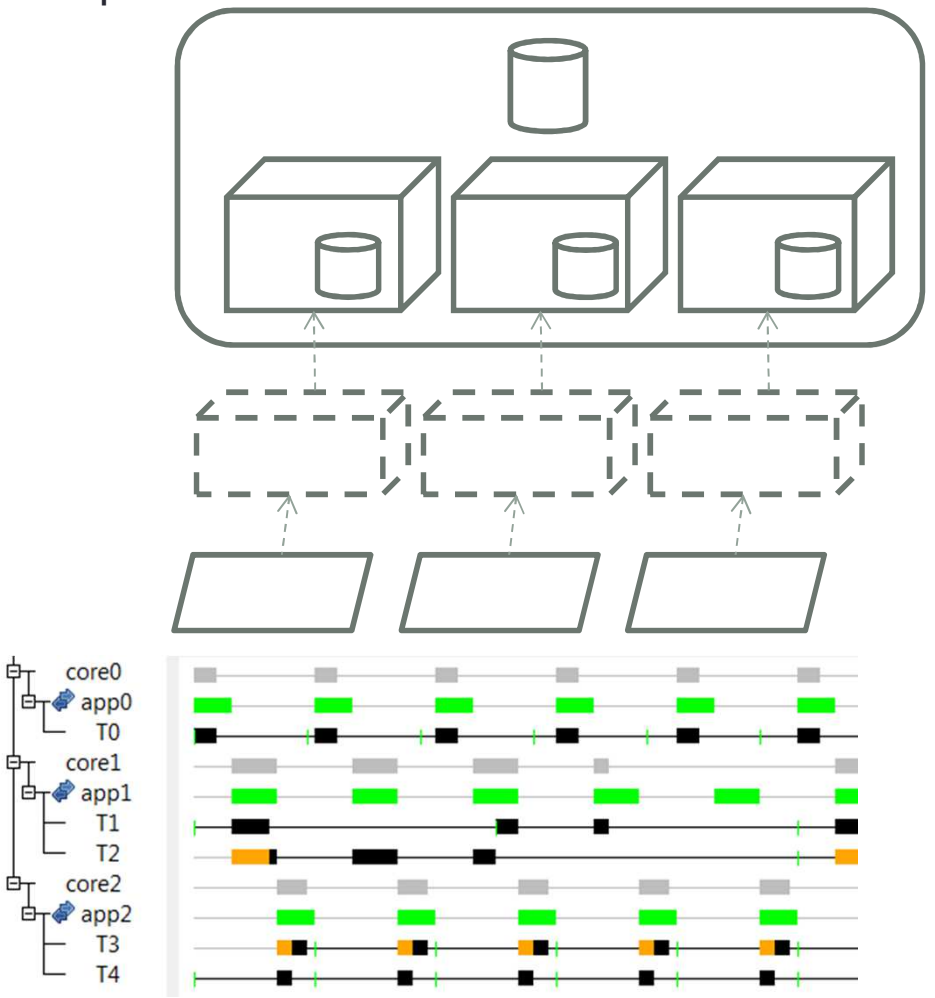
ARINC653 mono-processor « like » system



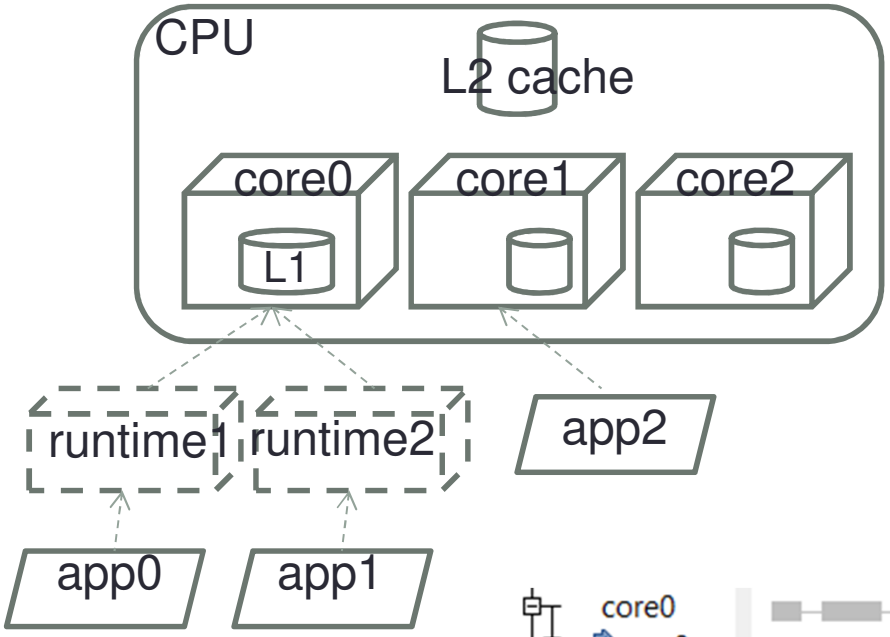
TSP on multi-processors

1-to-1 processor-partition binding

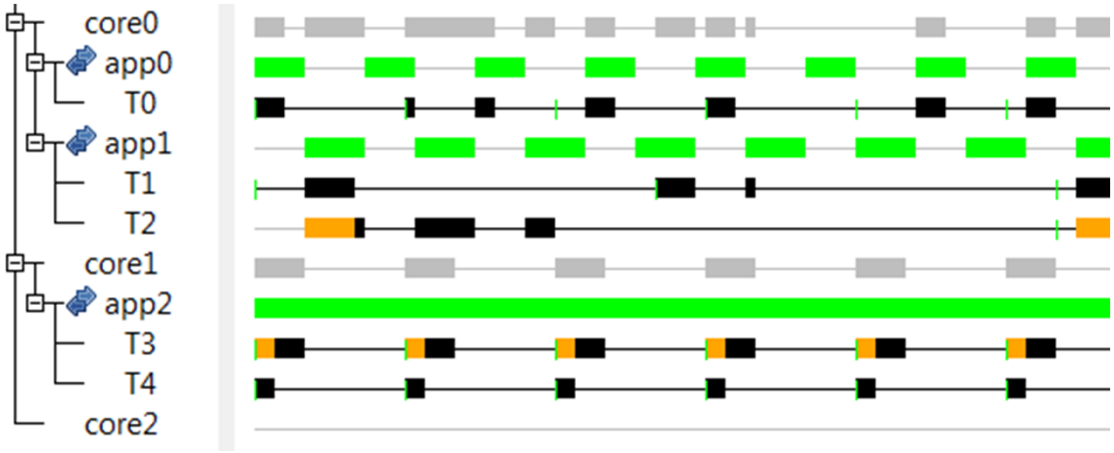
Private cache levels are also private for partitions



AMP configuration



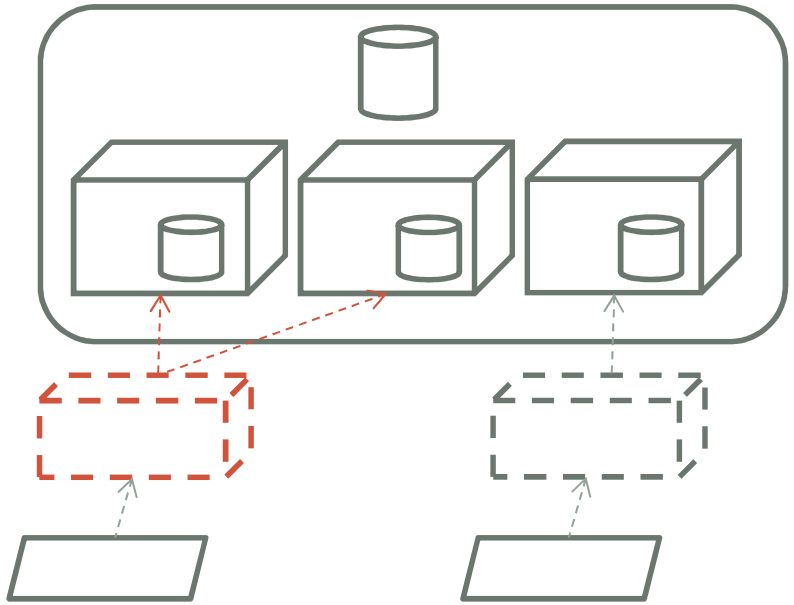
- An ARINC653 OS controls the core0
- Another mono-processor OS is used on core1
- Interferences between core1 and core0 ?



TSP configurations (2)

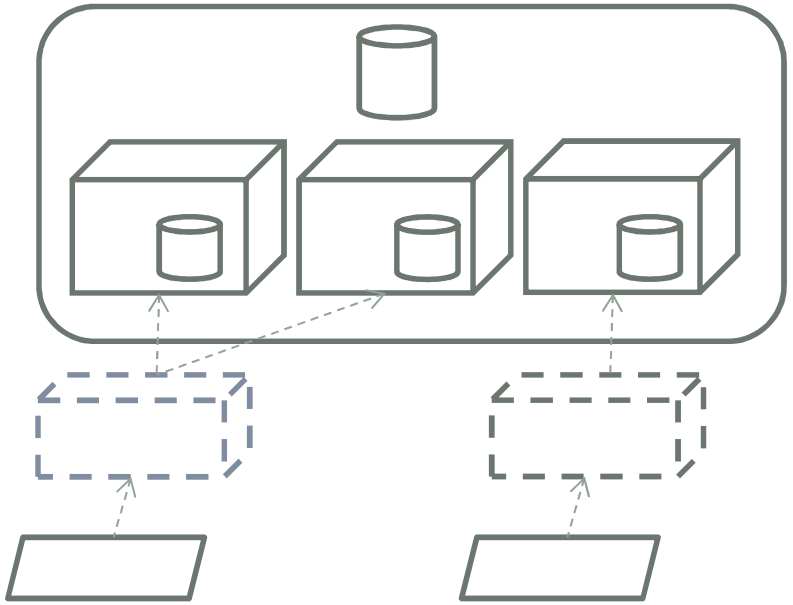
SMP partition

- Problem: A *virtual processor* does not represent several execution flows.
- How to model a “SMP runtime”?



Partition (runtime) migration

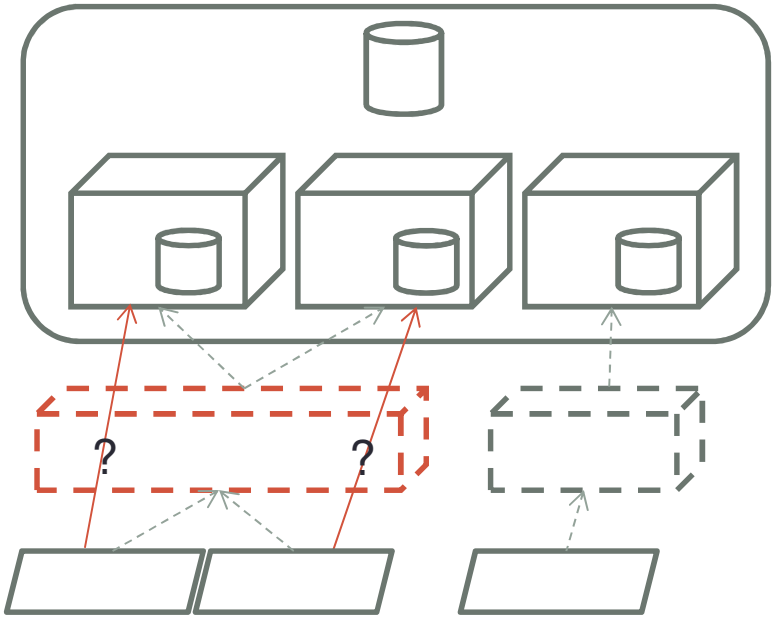
- How to distinguish SMP runtime from runtime migration?



TSP configurations (3)

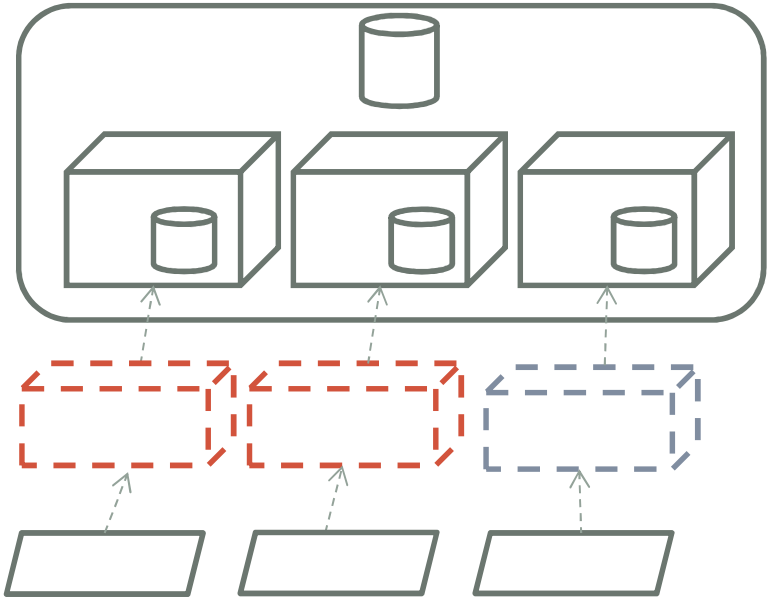
Intra-partition processor binding

- How to bind a TSP application to a physical processor?



Multiple runtimes in one partition

- Need to change *Module_Schedule* property



Outline

- Enumeration and identification of the processing units
  - Processor characterization
- Software deployment
  - Thread scheduling
  - Processor groups
- Hardware interference channels
  - Memory hierarchy modeling
- TSP configurations
- Further works

Further works

- Hardware platform modeling
  - Bus (mutli-core) or Network-On-Chip (many-core systems) characterization
  - Hardware hypervisor (freescale TOPAZ)?
  - NUMA access
  - What are the useful parameters at the system design level?
- Work (or not) on the “processor group” component ?