Modeling and scheduling analysis of multi-processor/multi-core/many-core architectures with AADL

Frank Singhoff+, Stéphane Rubini+, Hai Nam Tran+ (speaker), Pierre Dissaux*, Jérôme Legrand*

*** Ellidiss Technologies**
+ Lab-STICC UMR CNRS 6285, University of Brest

Introduction

- **Real-time scheduling analysis**
  - Models of basic functions (or tasks).
  - Models of hardware components.
  - Analytical methods, verify the feasibility/schedulability of a system.

- **Scheduling analysis for uniprocessor**
  - Simplified task models (e.g. Liu & Layland periodic task model) to model computational interferences on the processor (*processing units*)

Introduction

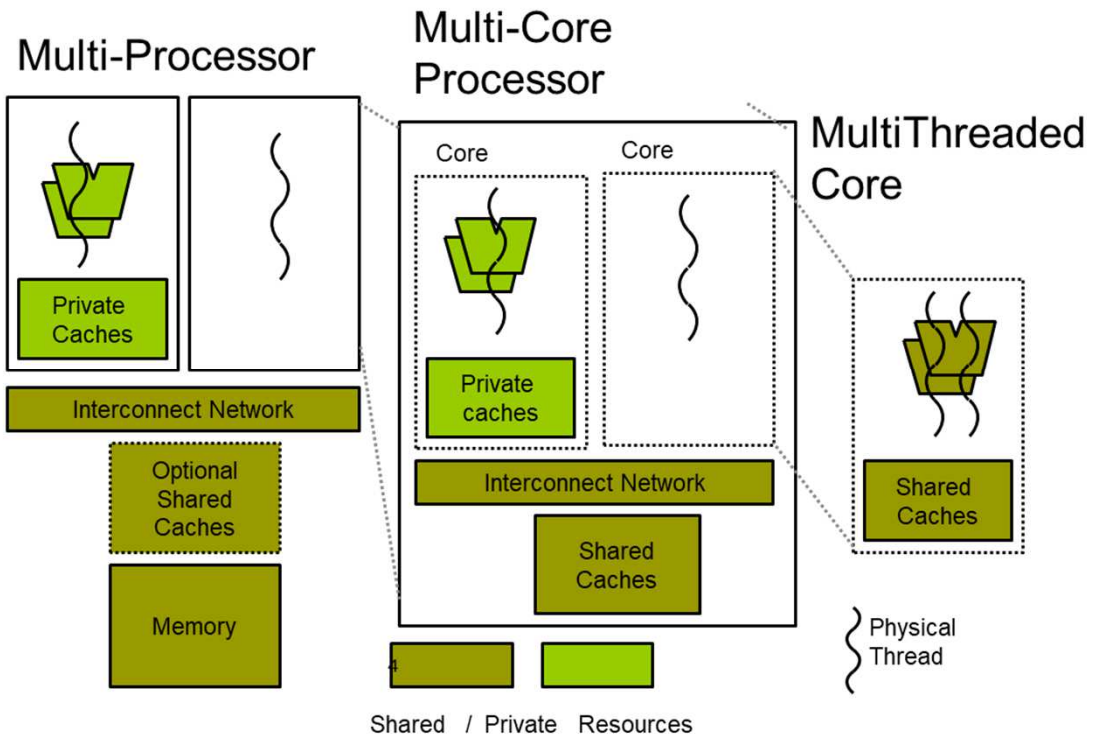
- **Scheduling analysis for multi-core/many-core**
  - Require modeling/verification of computational interferences + interferences due to other hardware resources - *memory units* and *interconnect units*
- Implementation effort of scheduling analysis for multi-core/many-core into Cheddar since 2012.
- **How to do it with AADL V2 ? What are missing? What should we include in V3?**

Agenda

1. Introduction
2. **Rational**
3. Modeling examples/guidelines
   A. Processing units
   B. Memory units
   C. Interconnect units
4. Summary/Conclusion

Rational: Processing units

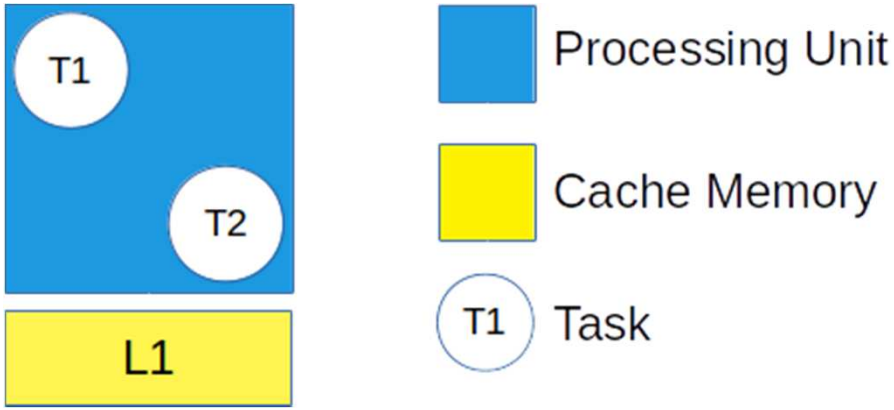
- Processors, cores or physical threads may be seen as processing units by the scheduler
- Asymmetric/Symmetric Multiprocessing (AMP/SMP)



Rational: Memory units

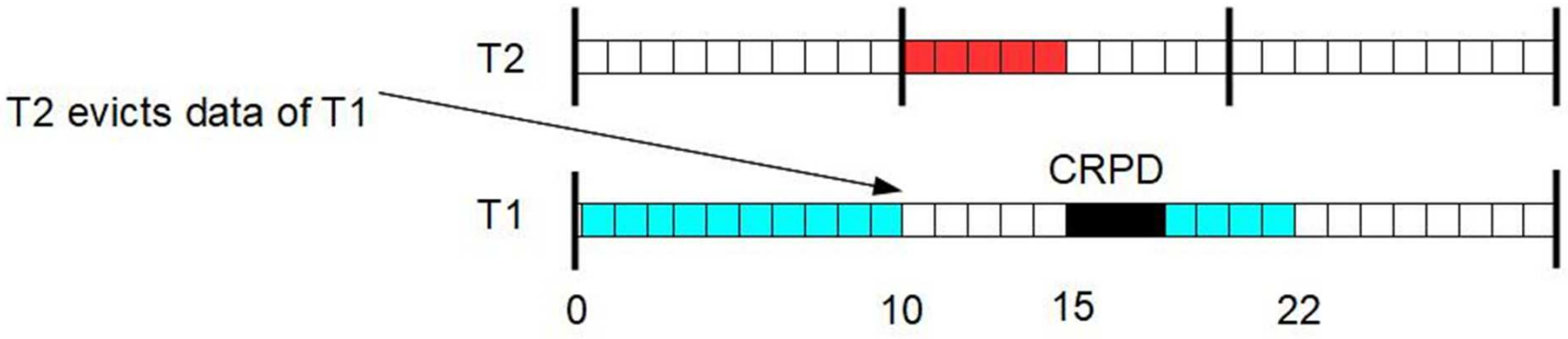
- Various memory units : cache, scratchpad, DDR3
- Processing units may contain memory units, and respectively

Rational: An example with cache



- Scheduling analysis = mastering interferences … software and hardware
- T1 and T2 are functionally independent but are not actually independent due to the hardware resources.
- Interferences due to the shared processing unit and memory unit

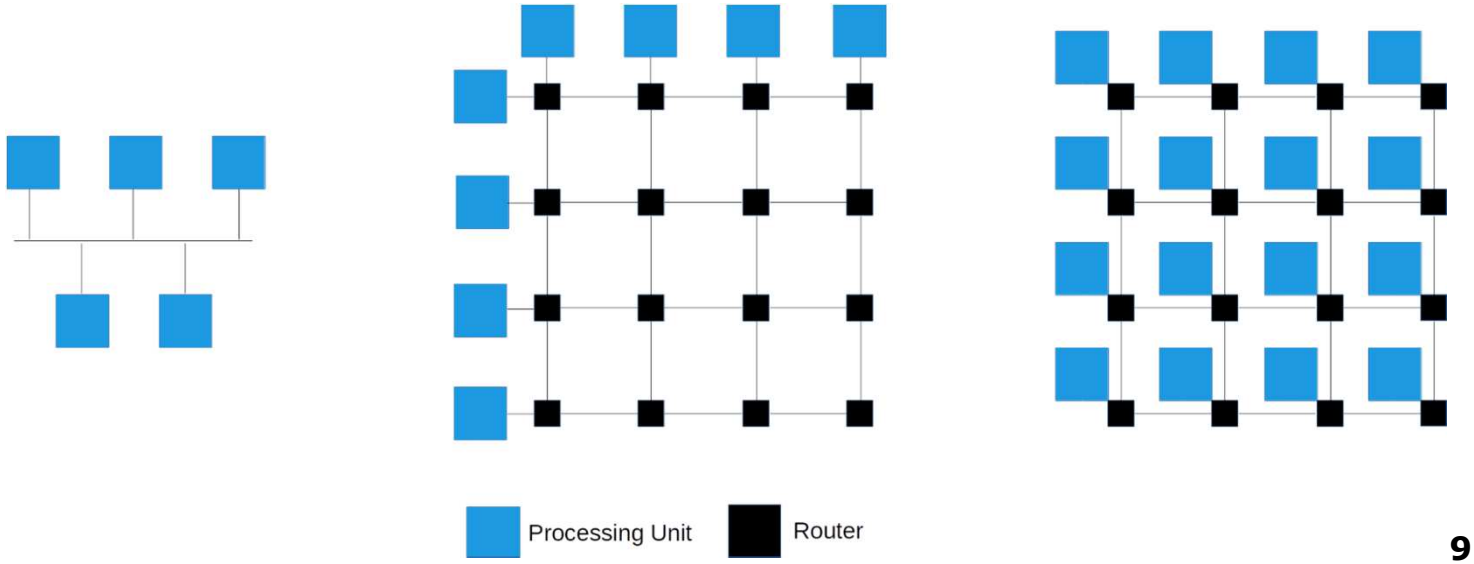
Rational: An example with cache



- Cache related preemption delay (CRPD): the additional time to refill the cache with memory blocks evicted by preemption.
  - CRPD is a high preemption cost, up to 44% of the WCET of task (Pellizzoni et al., 2007).
  - Variability of CRPD makes schedulability difficult

Rational: Interconnect units

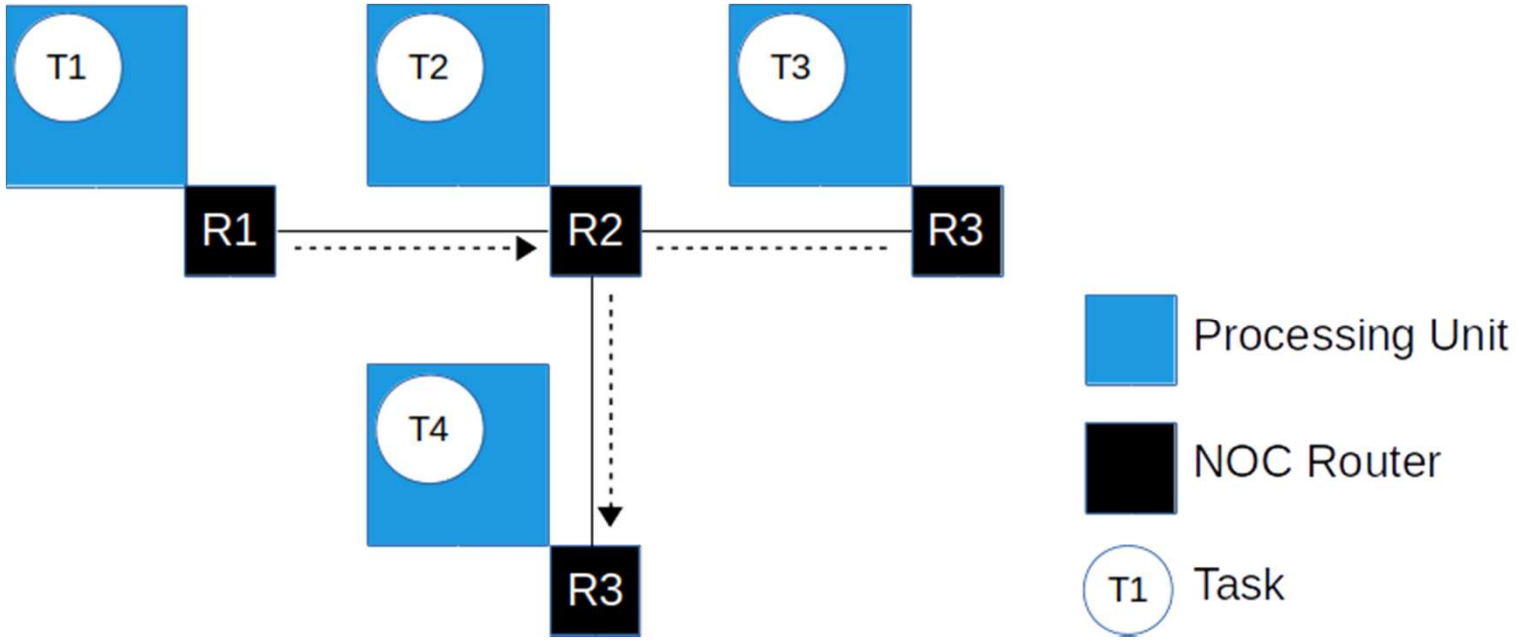
- Various interconnect technologies between processing/memory units
  - Bus, crossbar, **Network on Chip (NoC)**
- Various levels of shared resources & sharing protocols to model for scheduling analysis



Rational: An example with NoC

- T1 sends data to T2, T3 sends data to T4

→ Interferences on Router R2



Rational (summary)

- Various kinds of **processing units**: multi-core, multi-threading, multi-processor
- Various **memory units**: cache, scratchpad, DDR3
- Various **interconnect units** with different technologies between processing/memory units
- Need specific AADL properties for scheduling analysis

Rational (summary)

- We do not need to model hardware in the detail, but only:
  - Entities providing or using resources
  - Sharing/arbitration protocols leading to hardware interferences and to delays impacting schedulability

Agenda

1. Introduction
2. Rational
3. **Modeling examples/guidelines**
   - A. **Processing units**
   - B. **Memory units**
   - C. **Interconnect units**
4. Summary/Conclusion

Modeling guidelines

- **Entity to model**: processing units, memory units, interconnect units
- How to model
  1. Model system components with semantic defining the kind of modeled SoC component

```
Supported_Soc_Type : type enumeration (
        SoC_Processing_Unit,
        SoC_Memory_Unit,
        Soc_Interconnection_Unit);
System_Soc_Type : Supported_SoC_Type applies to (system);
```

  2. Required AADL properties for analysis => **Cheddar_Multicore_Properties**

Library processing/memory units

```
processor uni_core
properties
  Scheduling_Protocol=>(POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL);
end uni_core;

system implementation dual_core.impl
  subcomponents
       core1 : processor uni_core;
       core2 : processor uni_core;
  properties
       aadlv3::System_Soc_Type => SoC_Processing_Unit;
       Cheddar_Multicore_Properties::SoC_Interconnection_Type => Crossbar;
end dual_core.impl;

system implementation quad_core.impl
  subcomponents
       cores : processor uni_core [4];
  properties
       aadlv3::System_Soc_Type => SoC_Processing_Unit;
       Cheddar_Multicore_Properties::SoC_Interconnection_Type => Crossbar;
end quad_core.impl;
```

Library processing/memory units

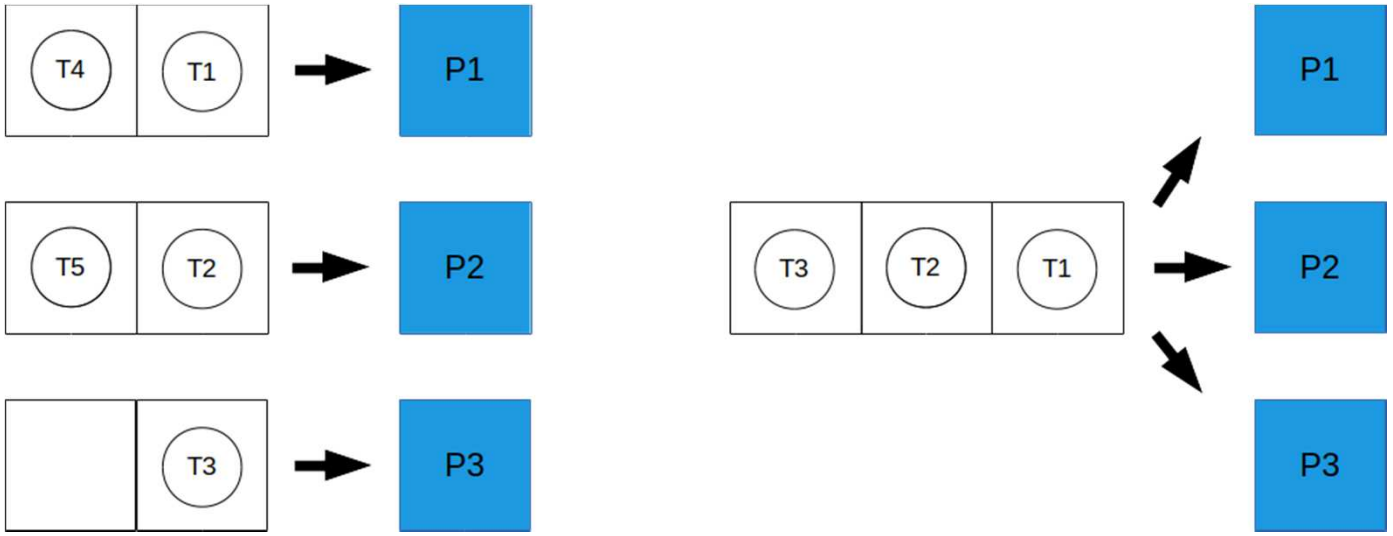
```
system implementation DRAM_chip.impl
    subcomponents
        banks : memory memory_units::Bank_Memory [7];
        controller : system DRAM_controller.impl;
    properties
        aadlv3::System_Soc_Type => Soc_Memory_Unit;
end DRAM_chip.impl;
```

```
system implementation DRAM_controller.impl
   subcomponents
       memory_requests : process DRAM_request_demand.impl;
       DRAM_controller_scheduler : processor multicore_crossbar_units::uni_core;
   properties
       aadlv3::System_Soc_Type => Soc_Processing_Unit;
       Actual_Processor_Binding => (reference (DRAM_controller_scheduler)) applies to memory_requests;
end DRAM_controller.impl;
```

Process to verify the proposals

1. **Design AADL model component/library units from the multi-core survey (Maiza et al., 2019)**: verify we can model them AADL V2/V3 and define required properties for analysis
2. **Prototype in OSATE 2 Cheddar plugin**: verify we can produce a Cheddar analysis model.
3. **Run schedulability analysis**: implement required/missing analysis methods into Cheddar and verify we have all required properties to run them succesfully

Processing units: examples with partitioned vs global scheduling



- **Partitioned scheduling:** first assign off-line each task on a processing unit ; each processing unit schedules its own task set.
- **Global scheduling:** choose the next task to run on any available processing unit (or preempt if all busy).

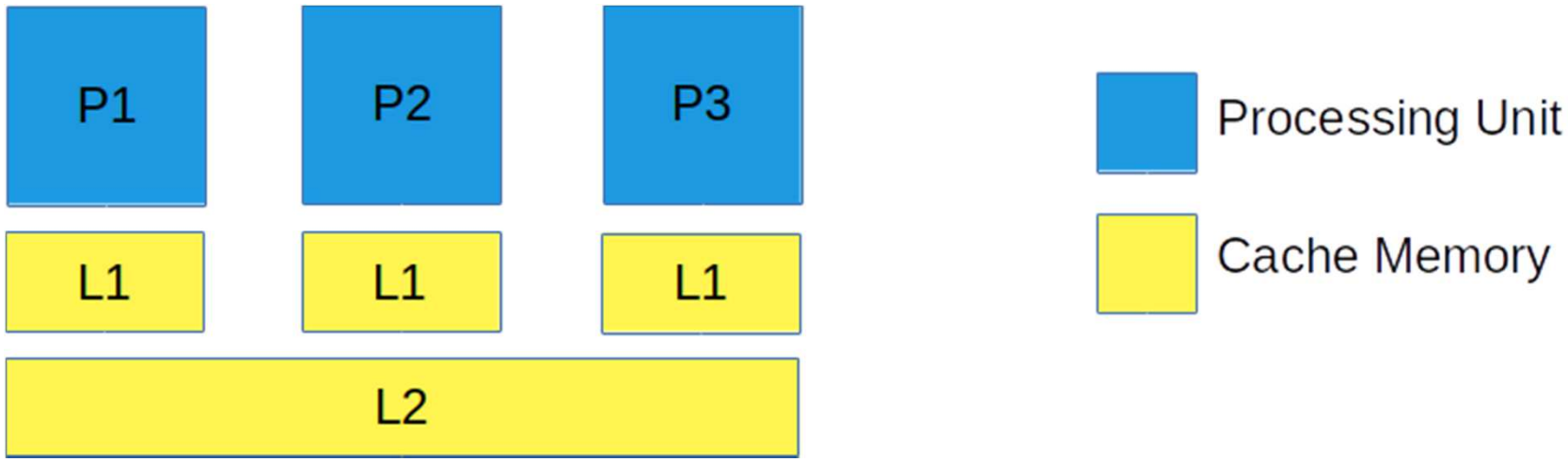
Partitioned scheduling (both processing and memory units)

```
system implementation dual_core.impl
  subcomponents
       core1 : processor uni_core;
       core2 : processor uni_core;
  properties
       aadlv3::System_Soc_Type => SoC_Processing_Unit;
       Cheddar_Multicore_Properties::SoC_Interconnection_Type => Crossbar;
end dual_core.impl;

system implementation processor_with_memory_latency.partitionned
subcomponents
    cores : system  multicore_crossbar_units::dual_core.impl;
    bank1 : system  memory_units::DRAM_chip.impl
        {Cheddar_Multicore_Properties::Private_Access_Latency => 2 us .. 10 us;};
    bank2 : system memory_units::DRAM_chip.impl
        {Cheddar_Multicore_Properties::Private_Access_Latency => 3 us .. 20 us;};
    soft  : process Software.impl;
properties
    aadlv3::System_Soc_Type => SoC_Processing_Unit;
    actual_processor_binding => (reference(cores.core1)) applies to soft.T1;
    actual_processor_binding => (reference(cores.core2)) applies to soft.T2;
    actual_memory_binding => (reference(cores.core1)) applies to bank1;
    actual_memory_binding => (reference(cores.core2)) applies to bank2;
end processor_with_memory_latency.partitionned;
```

Global scheduling (both processing and memory units)

```
system implementation dual_core.impl
  subcomponents
       core1 : processor uni_core;
       core2 : processor uni_core;
  properties
       aadlv3::System_Soc_Type => SoC_Processing_Unit;
       Cheddar_Multicore_Properties::SoC_Interconnection_Type => Crossbar;
end dual_core.impl;

system implementation processor_with_memory_latency.global
subcomponents
    cores : system  multicore_crossbar_units::dual_core.impl;
    bank1 : system memory_units::DRAM_chip.impl
        {Cheddar_Multicore_Properties::Private_Access_Latency => 2 us .. 10 us;};
    bank2 : system memory_units::DRAM_chip.impl
        {Cheddar_Multicore_Properties::Private_Access_Latency => 3 us .. 20 us;};
    soft  : process Software.impl;
properties
    actual_processor_binding => (reference(cores)) applies to soft;
    actual_memory_binding => (reference(cores)) applies to bank1;
    actual_memory_binding => (reference(cores)) applies to bank2;
    aadlv3::System_Soc_Type => SoC_Processing_Unit;
end processor_with_memory_latency.global;
```

Memory units: an example with cache



Memory units: an example with cache

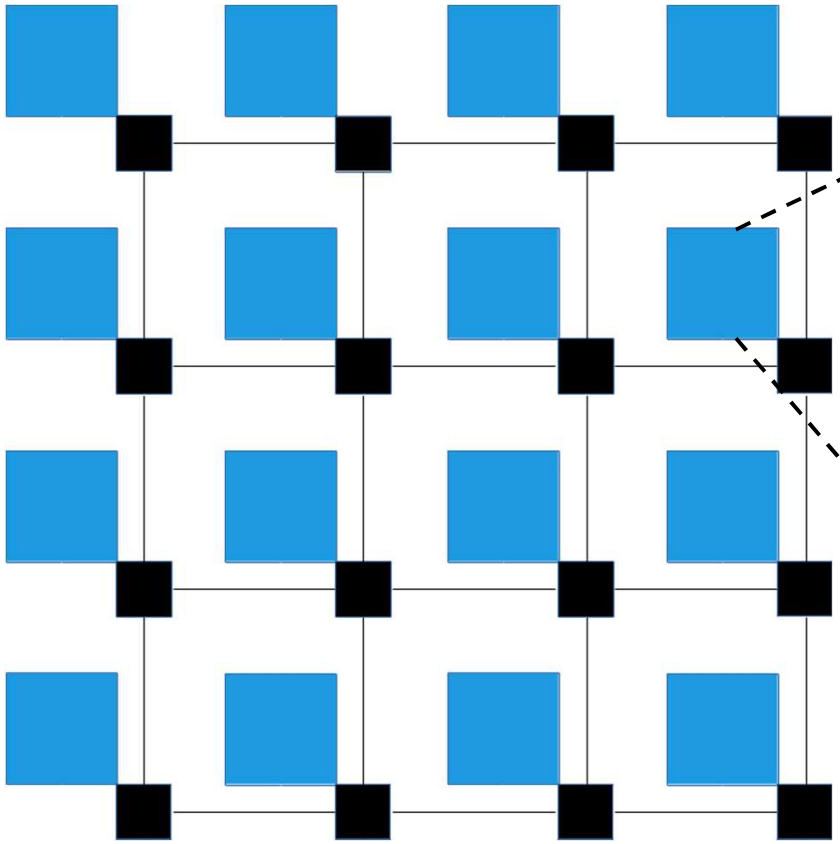
```
processor implementation core.impl
subcomponents
  L1_Instruction_Cache : memory memory_units::Cache.impl {
    Memory_Size => 1024Bytes;
    Cheddar_Multicore_Properties::Line_Size => 16Bytes;
    Cheddar_Multicore_Properties::Cache_Type => Instruction_Cache;
    Cheddar_Multicore_Properties::Cache_Level => 1;
    Cheddar_Multicore_Properties::Associativity => 1;
    Cheddar_Multicore_Properties::Cache_Size => 1024;
    Cheddar_Multicore_Properties::Block_Reload_Time => 1 us .. 2 us;
  };
properties
  Scheduling_Protocol=>(POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL);
end core.impl;
```

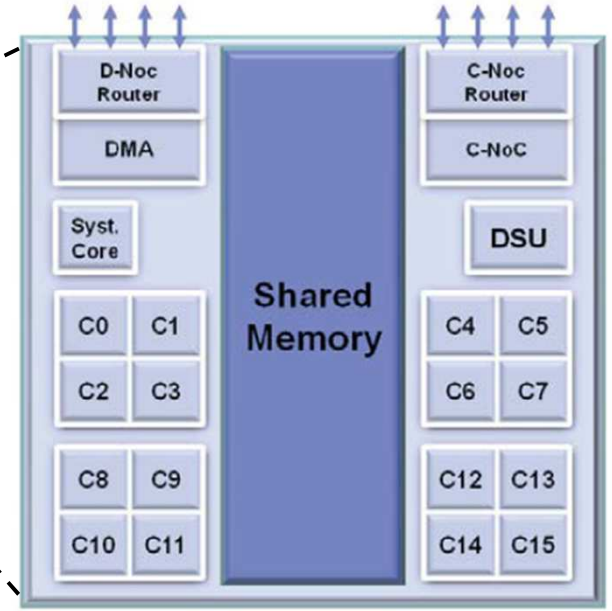
Memory units: an example with cache

```
system implementation crossbar_multicore.impl
subcomponents
    core1          : processor core.impl;
    core2          : processor core.impl;
    core3          : processor core.impl;
    L2_data_cache : memory memory_units::Cache.impl;
properties
    aadlv3::System_Soc_Type => SoC_Processing_Unit;
    Cheddar_Multicore_Properties::SoC_Interconnection_Type => Crossbar;
end crossbar_multicore.impl;
```

Interconnect units: an example with a many-core architecture + NoC

- Kalray MPPA-256 architecture*





Kalray MPPA-256 computing tile (Adapted from Djemal., 2014)

*: The NoC is illustrated as a 2D-Mesh for simplicity

Interconnect units: an example with a many-core architecture + NoC

```
system implementation cluster.shared
   subcomponents
       pe : system  kalray_core.impl [16];
       rm : processor multicore_crossbar_units::uni_core;
       shared_memory : memory memory_units::Bank_Memory.impl [16];
    properties
       Allowed_Memory_binding => (reference (pe)) applies to
           shared_memory;
       aadlv3::System_Soc_Type => Soc_Interconnection_Unit;
       Cheddar_Multicore_Properties::SoC_Interconnection_Type => Crossbar;
end cluster.shared;
```

Interconnect units: an example with a many-core architecture + NoC

```
system implementation mppa256.impl
   subcomponents
       clusters : system cluster.shared [4][4];
       io_subsystems : system io_subsystem.impl [4];
   properties
       aadlv3::System_Soc_Type => Soc_Interconnection_Unit;
       Cheddar_Multicore_Properties::SoC_Interconnection_Type => Torus_NoC;
       Cheddar_Multicore_Properties::Routing_Protocol => Programmable_Routing;
       Cheddar_Multicore_Properties::Switching_Protocol => Wormwole;
       Cheddar_Multicore_Properties::Arbitration_Type => Round_Robin_With_Bandwidth_Reservation;
end mppa256.impl;
```

Conclusion

- **Multi-processors:** several shared resources & entities, not only the processing units
- **Modeling interferences between software and hardware, and also between hardware and hardware** are needed for schedulability analysis

- **Status:**
  - Current proposal seems convenient to model various multi-core/many-core architectures that can be analyzed with Cheddar, while uniprocessor modelling/analysis rules are kept.
  - **Cheddar multi-core/many-core analysis features:** global/partitioned scheduling, cache CRPD, NoC worst case latencies, memory worst case latencies
  - **First set of AADL models for muticore archi. verified by Cheddar:** http://beru.univ-brest.fr/svn/CHEDDAR/trunk/project_examples/tests/osate_projects/
  - **Cheddar OSATE 2 plugin for multi-core:** not in the update site, but available on request

- **Few open questions:**
  - Cache units: CFG, BA or separate model + properties?.
  - Missing models to have a higher coverage of the multi-core survey (Maiza et al., 2019)
  - TSP/ARINC 653 + multi-core examples (Frank's visit to SEI next Feb.)