Modeling Shared-Memory Multiprocessor Systems with AADL

Stéphane Rubini[1], Pierre Dissaux[2], and Frank Singhoff[1]

[1] Université de Bretagne Occidentale, UMR 6285, Lab-STICC, Brest, France
[2] Ellidiss Technlogies, Brest, France
{stephane.rubini,frank.singhoff}@univ-brest.fr
pierre.dissaux@ellidiss.com

Abstract. Multiprocessor chips are now commonly supplied by IC manufacturers. Real-time applications based on this kind of execution platforms are difficult to develop for many reasons: complexity, design space, unpredictability, ... The allocation of the multiple hardware resources to the software entities could have a great impact on the final result.
Then, if the interest to represent the set of hardware computing resources inside an architectural model is obvious, we argue that other hardware elements like shared buses or memory hierarchies must be included in the models if analyze of the timing behavior is expected to be performed. This article gives guidelines to represent, with AADL, shared-memory multiprocessing systems and their memory hierarchies while keeping a high-level model of the hardware architecture.

Keywords: Multiprocessor, Architecture Description Language, Memory Hierarchy

Introduction

The growth potential for computing power supplied by general purpose monoprocessor (GPU) systems, is quite decreasing. New gains are related to complex hardware structures and to very advanced fabrication technologies. Especially in the embedded system context, the additional power consumption to paid for increasing the instruction rate is high.

Today, the main answer to that problem is to multiply the number of processing units in VLSI chips. Less aggressive internal core designs allow for increasing the hardware efficiency and reducing the thermal problems (hot spots).

Moreover, as the execution units are multiple, it becomes possible to specialize some of them for dedicated usages. As an example, System-On-Chips like TI OMAPs include multiples GPUs, a DSP and an image processing unit.

These Programmable Heterogeneous Multiprocessors (PHMs) are now execution platforms that the designers must consider when they develop new products. The focus should not be only on the individual processing units, but on the whole hardware system. An early knowledge of some non-functional details of the execution target might be a condition to lead a project to completion.

A meaningful performance analysis cannot be conducted without considering a realistic model of the platform, and the data flow. The challenge is to eliminate low-level complexity, while the functional or non-functional behaviors remain close to the reality of the execution environment. The complexity of both the hardware execution platform and the software application requires that designers develop high-level models of their systems.

At the same time, system design space exploration requires to apply separation-of-concerns principles and to distinguish the application model from the architecture platform one. This approach requires that a deployment step maps software entities onto the hardware resources, statically or dynamically like for instance in the case of multi-processor scheduling decisions.

The aim of the paper is to propose a level of abstraction of the hardware for complex shared-memory multi-processor platforms. The section 1 focuses on the modeling of the processing resources based only on the functional point of view. The usefulness of such models are tested for the allocation of software entities on processors through deployment processes. The section 2 develops these descriptions to include memory hierarchy models, as a major structuring entities by considering the information flow into the system. In the next section, we apply those modeling guidelines to a real board. Hence, examples of tools that work or could work from the modeling level that we propose are given. Finally, after the description of related works, we conclude on some uncovered features of the hardware platform that could enrich our modeling guidelines.

1 Resource allocations

When a hardware platform provides multiple execution units, the design process must include an allocation or a placement strategy. The system model defines what software entities are assigned to hardware resources. Such assignments may be done by the mean of a component-containment hierarchy[1]. However AADL addresses the placement of software on a hardware resources through property values. Such binding properties form a deployment or an allocation layer in the model (Fig. 1).

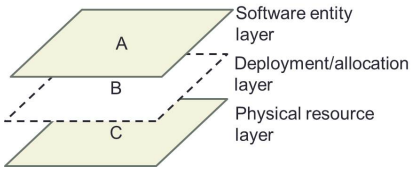

Fig. 1. Layered model

The next two paragraphs illustrate the usage of binding properties for allocating tasks and logical partitions to processors.

Processing unit modeling and allocation In multi-processing systems, the processing units may be implemented as a separated chip (a processor), or grouped into a single chip (a core). Finally, the data path of a core or a processor may be shared to execute concurrently several execution threads (physical threads)[2].

A way to model the system is to enumerate, at the same level, all the processing units without consideration of their actual implementation. The model of the Fig. 2 represents the available processing units of a PHM as arrays of AADL processors which group together those compatible with the same instruction set.

```
processor isa1 end isa1;
processor implementation isa1.proc end isa1.proc;
processor implementation isa1.core end isa1.core;
processor implementation isa1.phys_thread end isa1.phys_thread;
processor isa2 end isa2;
system mpHardSystem end mpHardSystem;
system implementation mpHardSystem.impl
subcomponents
  PU_isa1 : processor isa1[N];
  PU_isa2 : processor isa2[M];
end mpHardSystem.impl;
```

Fig. 2. Hardware layer (layer C of Fig. 1): set of processing units. N and M parameters are constant fixed in the model. `isa2` processing unit implementations are not exposed.

The ability of this model to express the allocation of the hardware execution resources to the software tasks is shown Fig. 3.

```
Allowed_Processor_Binding => (reference(exec.PU_isa1))
     applies to app.thrs;
Allowed_Processor_Binding => (reference(exec.PU_isa2))
     applies to app.dedicatedThr;
Actual_Processor_Binding => (reference(exec.PU_isa1[1]),
                             reference(exec.PU_isa1[4]))
     applies to app.thrs[2];
```

Fig. 3. Software placement (deployment layer B). `exec` is a component of type `mpHardSystem`. `app`, `thrs` and `dedicatedThr` are respectively a process, an array of threads and a singular thread. `thrs[2]` may be scheduled on processor units 1 and 4.

As the processing units of an array are homogeneous, all tasks, which are abstraction of executable codes for an ISA, can be allocated on all of them. The `Allowed_Processor_Binding` property expresses this in the model. Then,

thee `Actual_Processor_Binding` properties allocate threads onto one or a set of processors.

Process and partition allocations The same basic principle of allocation schemes can be applied to processes to define the visibility space of data in the context of partitioned systems. On the model of the Fig. 4, a process `part` may be used by two processing units following the operational mode: in normal mode, only `PU_isa1[0]` has the right to access it, whereas `PU_isa1[1]` is activated in an escape state.

```
system implementation main.redundant
subcomponents
 part : process partition.impl;
 exec : system  mpHardSystem.impl;
modes
 normal    : initial mode;
 redundant : mode;
properties
 Allowed_Processor_Binding => (
  reference(exec.procUnits[0]), reference(exec.PU_isa[1])
  ) applies to part;
 Actual_Processor_Binding => (reference(exec.PU_isa1[0]))
    in modes (normal) applies to part;
 Actual_Processor_Binding => (reference(exec.PU_isa1[1]))
    in modes (redundant) applies to part;
end main.redundant;
```

Fig. 4. Partition allocations (adapted from the OSATE github examples/core-examples/multi-core/simple.aadl)
.

In [3], similar allocation scheme is used to model that an AADL *virtual processor* can be hosted by several processing units. An implementation requirement is that the context can migrate from one processing unit to the other one.

Discussion These examples, i.e. thread allocations in the scheduling field and partition allocations in the Time-Space Partitioned field, are nearly similar, and the modeling approach is homogeneous. The deployment layer represents spatial allocations of the software entities. Temporal intervals where the resources are actually used are not directly specified; property values assigned to hardware resources may reference what are the rules to determine the dynamic sharing of the resource.

At the hardware layer level, the architecture models enumerate the processing units as a "flat" structure; they abstract multi-processing architectures as a set of available implementation-agnostic computing resources.

But a major problem with tightly coupled systems like shared-memory multiprocessors comes from the unexpressed sharing of resources at the level of memory hierarchy. Private cache memories locally separate the information flows near the processing units, but some parts of the memory hierarchy remain shared between processors.

The programming model of shared memory multiprocessor platforms allows communications of data or synchronizations between tasks through memory locations accesses. However, such a functional explicit sharing is generally secondary with respect to the sharing of the main memory banks and buses which provides all the memory words that the program execution requires.

For instance, [4] reports the effect of this resource sharing on a quad core system (Intel Xeon) for programs of the SPEC2006 benchmark: when a core executes a synthetic workload, another program running on the other core may experience an impressive slowdown (up to a slowdown of 200%).

So, an AADL model of shared memory multiprocessor systems must represent the resources shared by the processing units, especially if the software specification does not express this sharing as own.

2 Modeling of the memory hierarchy

The memory hierarchies can be complex subsystems, but they mostly expose, to the software, the interface of a simple memory[3]. Hence, designers must take attention to memory hierarchy as its non-functional characteristics are significant.

The guidelines we propose are based on the modeling of the memory hierarchy as a tree. The terminal nodes represent the processing units and the nodes abstract the different sub-parts of the memory system. A node of the memory hierarchy is characterized by the visibility of the memory words it contains; a node is shared by the same set of processing entities. Data exchanges within the memory hierarchy follow the paths of the tree.

The following rules may be applied to build the model:

- 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.

[3] We consider that the program or the operating system are aware of memory coherency problems.

When required for analyzing purposes, processors and composite memory components may detail their internal structure. AADL bus connection features can be used for that purpose. Information about the behavior of the each memory entities are given by AADL properties or AADL memory classifiers, for instance for distinguishing the main memory and the cache levels.

The Fig. 5 shows the modeling of the tree with a hierarchy of AADL components. The memories associated to the M_y sub-system are used or shared by P_j and P_k processors. M_x is the root memory system, including all the data that could be addressed in the physical address space. The M_x memory is shared by the P_i processor and M_y memory sub-system.

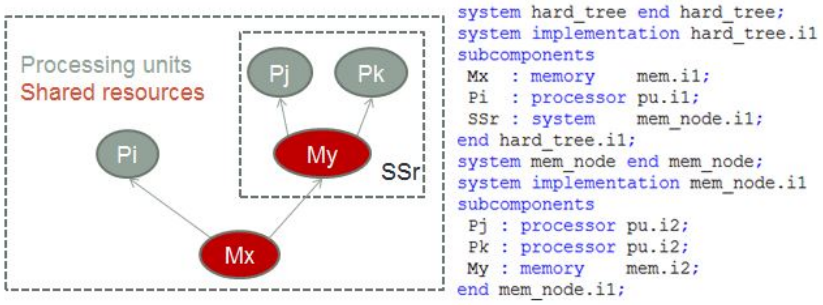

Fig. 5. "Sharing" tree and its AADL representation.

3 Example

As an example, we apply the guidelines that we propose on the processor of a VPX board from the company *InterfaceConcept*. The board VPX3a includes an Intel *core i7*, a FPGA Xilinx Kintex7 and an IO Bridge. The *core i7 2566LE* processor contains two cores, with an optional activation of multi-threading capabilities. Three levels of caches constitute the memory hierarchy; level 1 is composed of two separated caches for instructions and data, level 2 and 3 are unified ones. Only the level 3 of cache is shared between cores.

The Fig. 6 shows an high level model of the hardware architecture for the VPX3a board, with emphasis on the processing units and the memory hierarchy. We assume that the hyper-threading is enabled on the core number two.

The Fig. 7 and 8 show the structure of the memory hierarchy nodes. AADL bus connections have been used to model the internal hierarchy of the implementation `L1I_L1D_L2` of `mem_system`. A similar modeling method can be applied to the other memory systems.

Notice that the cache sharing may change during the design process, even on a given execution platform. The activation of the multi-threading on a core or

```
system implementation exec.IC_INT_VPX3a
subcomponents
  core1 : processor Intel64.core_single_thread;
  core2 : system    Intel64.core_dual_thread;
  mem   : memory    mem_system.L3_main;
end exec.IC_INT_VPx3a;
system implementation Intel64.core_dual_thread;
subcomponents
  pth1 : processor Intel64.physical_thread;
  pth2 : processor Intel64.physical_thread;
  mem  : memory    mem_system.L1I_L1D_L2;
end Intel64.core_dual_thread;
```

Fig. 6. Model of the processor core i7 2566LE. The root subsystem `mem` is used directly by the processor *core1* and communicates with the subsystem *core2*. *core2* contains two physical threads which share the caches L1 and L2.

```
memory implementation mem_system.L3_main
subcomponents
  L3cache : memory cache_system.L3;
  main    : memory shared_memory.main;
end mem_system.l3_main;
processor implementation Intel64.core_single_thread
subcomponents
  L1Icache : memory cache_system.L1I;
  L1Dcache : memory cache_system.L1D;
  L2cache  : memory cache_system.L2;
end Intel64.core_single_thread;
```

Fig. 7. Structured memory subsystems and processor's private memory resources. The L2 cache is private to a core when hyper-threading is not activated. The L3 cache and the main memory are shared by all the cores, and then are grouped into a unique node.

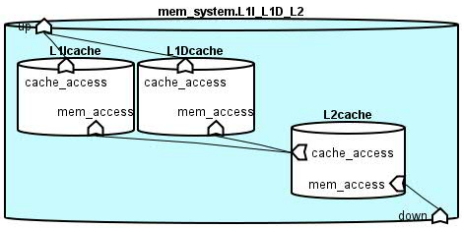

Fig. 8. Internal structure of the L1I_L1D_L2 composite memory system.

the operating system enabling of a level of cache are examples of modifications that a designer could do to explore their impact on the system performance. If such situations may exist, the component hierarchy can contain "potentially shared" nodes in the tree, currently used by only one processing unit.

4 Exploitation of the models

As shared-memory multiprocessor platforms become more complex and widely used, designers of real-time systems need to have access to analysis tools in order to assess their choices or dimension their systems. Such tools must work from an architectural model of the execution platform; we think that the modeling guidelines presented in this article set up a right level of abstraction for this purpose. These AADL models may be used directly as a tool entry or as a source for a model transform whether the tool uses a DSL language.

In the sequel, we give examples of tools which are or could able to handle models compliant with the guidelines proposed in this article.

Scheduling analysis *AADL Inspector* from Ellidiss Technologies is a lightweight framework allowing to apply different analysis on AADL models. One of the analysis tools available with *AADL Inspector* is Cheddar[5], a scheduling analysis tool which deals with global multiprocessor scheduling [6]. *AADL inspector* transforms the significant AADL entities (components and properties) into Cheddar-ADL components, in order to control the scheduling analysis [7]. Today, the model transform does not deal with caches, but Cheddar includes some algorithms to evaluate the cache related preemption cost when instruction caches are defined in an architecture model [8].

Resource allocations The placement of software entities on hardware resources may be complex if the number of tasks or virtual processors is high. The use of optimization methods are sometimes necessary to perform these allocations. For instance, Cheddar implements basic partitioning algorithms such as RM-Best-Fit [9] to automatically allocate tasks to processors.

From an AADL model, `Allowed_Processor_Binding` can be used to supply the set of candidate resources and allocation targets (e.g. processors). The results of partitioning algorithms can then be expressed on the analyzed AADL model by updating its `Actual_Processor_Binding` properties. *AADL Inspector* will integrate such a function in its future releases.

Timing analysis In [10], the timing analysis begins by considering the processor individually and isolated from the other ones. The tool conducts a first computation of WCET, completed by a cache analysis. A multilevel hierarchy of caches is considered at this stage; the hierarchy ends with the first level of shared cache.

Then, the analysis assumes that all the possible conflicts happen with the other processors for the shared cache usage. From this pessimistic analysis, the worst case response times of all the tasks are computed, and the interferences

previously considered are kept if the task's running times intersect. This process is recurrently performed upto a fix point is reached.

These examples of tools and analysis show the interest of modeling multi-processing platform and their memory hierarchy, as many analysis can be conducted from that kind of models.

5 Related works

We describe in this section other modeling approaches used to guide performance analysis on shared-memory multiprocessor systems.

In [1], Paul proposes a software-on-hardware performance modeling environment called MESH that founds the modeling and the simulation of single-chip multiprocessors on a layered approach. A software and a hardware layer respectively include the application threads and the hardware resources. An intermediate layer represents the dynamic mapping of logical threads onto physical resources through the scheduler decisions and associates thread's logical events to physical times. MESH models can include shared resources other than execution units; the simulation kernel applies time penalties of application threads when access contentions are detected.

In the context of shared-memory multiprocessors, [11] abstracts the essential features of the memory hierarchy in a formal model. A 3-tuple $\langle p_l, \alpha_l, \sigma_l \rangle$ characterizes a level l of memory caches, where p_l is the number of cores sharing the level l caches, α_l the number of level l caches, and σ_l the size of one level l cache. From the rules of a peddle game, the authors establish lower bounds on memory traffic between different levels of caches for computations represented as a directed acyclic graph. Next, they base optimizations on this metric to improve the implementation of some parallel algorithms.

The modeling of multicore execution platforms with *Simulink* has been proposed by [12] to optimize the partitioning of tasks in soft real-time systems. The *Simulink* model represents the cores, the core communication costs, and the task set. But, the approach does not take into account the memory hierarchy in the performance analysis; the behavior issued from the sharing of the caches cannot be inferred from this model.

Conclusion and future works

This article presents an approach for modeling shared-memory multiprocessor platforms. Basic entities that we have emphasized in this high-level model, are the processing units, whereas the way they are implemented, and the different levels of the memory hierarchy. We expect the architecture of the memory subsystem is a key feature to perform timing analysis on multi-processing platforms.

The memory hierarchy is not the only challenge in the timing analysis on multiprocessor platforms. Another topic is the multiplication of interconnection buses, with various characteristics in term of throughput and latency. Multiprocessor SoCs use the capabilities of integrated circuit to implement a lot of buses

to connect the different functions available on the chip. One goal devoted to the architectural models will be to document this structure, and guide tools and designers to identify potential bottlenecks.

Another problem is to deal with PHMs. SoCs mix different types of processors, with different speeds or capabilities. How AADL models can represent all these types of information must be investigated.

Acknowledgments This work is done in the context of the SMART project. SMART and Cheddar are supported by the *Conseil Régional de Bretagne*, *Bpifrance*, *Conseil Général du Finistère* and *BMO*. Cheddar is also supported by *Ellidiss Technologies*, *EGIDE/PESSOA n. 27380SA* and *Thales TCS*.

References

1. J. M. Paul, D. E. Thomas, and A. S. Cassidy, "High-level modeling and simulation of single-chip programmable heterogeneous multiprocessors," *ACM Trans. on Design Automation of Electronic Systems*, vol. 10, no. 3, pp. 431–461, 2005.
2. S. Rubini, C. Fotsing, P. Dissaux, F. Singhoff, and H. N. Tran, "Scheduling analysis from architectural models of embedded multi-processor systems," *ACM SIGBED Review*, vol. 11, no. 1, 2014.
3. J. Delange and P. Feiler, "Design and analysis of multi-core architectures for cyber-physical systems," in *Proceedings of the 7th European Congress Embedded Real Time Software and Systems (ERTSS)*, Toulouse, France, Feb. 2014.
4. V. Babka, "Cache sharing sensitivity of SPEC CPU2006," Distributed Systems Research Group, Department of Software Engineering, Tech. Rep., 2009.
5. F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a Flexible Real-Time Scheduling Framework," *ACM SIGAda Ada Letters*, vol. 24, no. 4, Dec. 2004.
6. R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011.
7. P. Dissaux, O. Marc, S. Rubini, C. Fotsing, V. Gaudel, F. Singhoff, and H. N. Tran, "The SMART project: Multi-agent scheduling simulation of real-time architectures," in *Proceedings of the 7th European Congress Embedded Real Time Software and System (ERTSS)*, Toulouse, France, Feb. 2014.
8. H. N. Tran, F. Singhoff, S. Rubini, and J. Boukhobza, "Instruction cache in hard real-time systems: modeling and integration in scheduling analysis tools with AADL," in *Proceedings of the 12th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC 14)*, Milan, Italy, August 2014.
9. Y. Oh and S. H. Son, "Tight performance bounds of heuristics for a real-time scheduling problem," *Technical Report CS93-24, University of Virginia.*, 1993.
10. S. Chattopadhyay, A. Roychoudhury, and T. Mitra, "Modeling shared cache and bus in multi-cores for timing analysis," in *Proceedings of the 13th ACM International Workshop on Software & Compilers for Embedded Systems*, 2010, pp. 6–15.
11. J. E. Savage and M. Zubair, "A unified model for multicore architectures," in *Proceedings of the 1st ACM International Forum on Next-generation multicore/-manycore technologies*, 2008, pp. 9–20.
12. J. Feljan, J. Carlson, and T. Seceleanu, "Towards a model-based approach for allocating tasks to multicore processors," in *Proceedings of the 38th EUROMICRO Conference on the Software Engineering and Advanced Applications (SEAA)*, Sep. 2012, pp. 117–124.