Architectural patterns for Asymmetric Multiprocessing Devices on Embedded Systems

PEDRO IGNACIO MARTOS, National University of La Plata

In embedded systems there is a variant of Multicore System on Chip devices (MSoC devices) where not all the computing elements (processors) are equal. The differences on the processors in these devices are ranging from different hardware architectures using the same instruction set to completely different processors working together inside the same device. These SoCs are called “Asymmetric Multi Processing Devices” (AMP Devices).

In this work we present four architectural patterns (three patterns and one antipattern) focused on this kind of devices with application in low power, performance, and real time software requirements.

General Terms: Embedded Systems Patterns

Additional Key Words and Phrases: Asymmetric Multiprocessing Patterns

1. INTRODUCTION

The Embedded Systems (E.S), as opposed to general purpose systems, are systems developed for a very specific purpose. In some cases the final user of the system can configure them (even program them), but the system is not intended to change its purpose. These systems are called “embedded” because they are part of a bigger system in a device with a specific functionality. [1]

An Asymmetric Multicore Processor (AMP), is a processor where the computing elements (“cores”) have different characteristics, they can vary from the same processor running with different clock speeds, up to completely different processor architectures (i.e. 64 bit cores with 32 bit cores in the same processor).

These processors are studied because of the advantages that they offer. Their performance is over the expected average. Studies show that the AMP configurations with two complex processors and two simple processors have better performance that the average of four complex processors and four simple processors over a wide variety of computing loads (application server, database server, web server, scientific computing, video compression, massive software compilation) [2]:

$$Avg\_Perf(2\ Complex\ Cores + 2\ Simple\ cores) > \frac{Avg\ Perf(4\ Complex\ Cores) + Avg\ Perf(4\ Simple\ Cores)}{2}$$

Thus, they are a very attractive computing platform for servers and workstation in the middle/low range. Besides that, the AMP architectures show an optimal performance in the (performance)/(power) and (performance)/(silicon area) ratios [3], so they are also attractive for embedded systems implementations.

2. A PATTERN LANGUAGE FOR ASYMMETRIC MULTIPROCESSING

The AMP Embedded Systems have architectural design patterns that take advantage of their very specific characteristics, and those patterns are related to the application of the E.S. that is under development. Also a pattern language is a collection of patterns related and applicable in a specific domain. The patterns generally are a small nugget of information that is not going to “resolve the world” or be a “silver bullet” for any problem, they resolve problems within a certain context and after resolution they leave the system in a new context, where there are new problems to be resolved [4]

Author's address: Universidad de Buenos Aires, Facultad de Ingeniería, Departamento de Electrónica, Paseo Colon 850 piso 1° Ala Sur; email: pimartos@gmail.com / pmartos@fi.uba.ar

This work presents a pattern language for the domain of AMP systems, the patterns in this language are organized in a hierarchy, rooted at “Asymmetric Multiprocessor”, a high-level pattern about the use of AMP processors. That pattern is specialized by the patterns “MiniMe”, intended for low power systems; the pattern “Optimized Execution”, where the software is optimized for the ISA of the processor in which is going to be executed; and the anti-pattern “Level Down” , where the software is compiled without optimizations to be able to run on any processor of the AMP system. The pattern “Optimized Execution” has a further specialization: “Dedicated Processor”, where some processors in the AMP system are tailored for very specific tasks, so the software must cope with that particularity

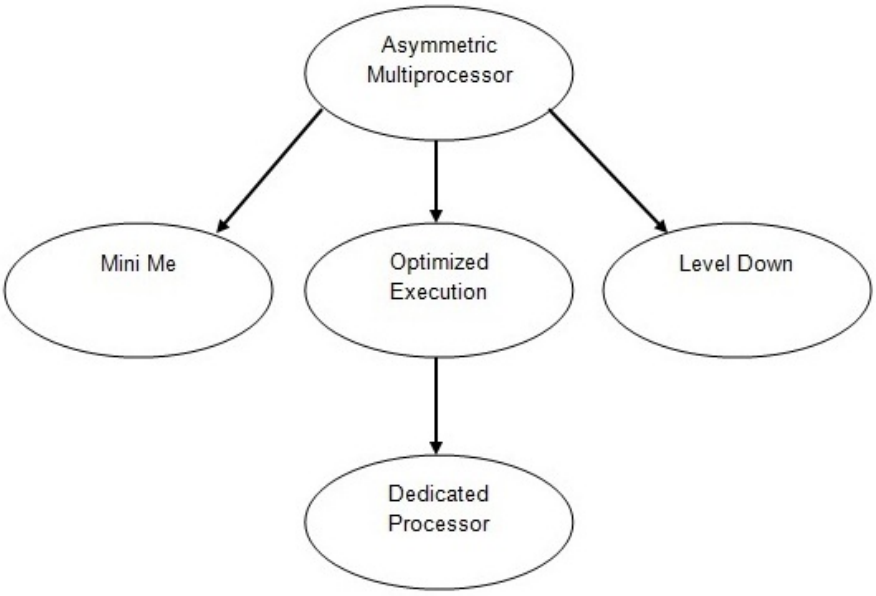

Fig.1 Asymmetric Multiprocessing Pattern Language Hierarchy Tree.

3. PATTERN “ASYMMETRIC MULTIPROCESSING” (A.K.A. POOR MAN’S SUPERCOMPUTER)

3.1 Context

Some systems require achieving two competing goals at the same time: high performance and low energy consumption. For those systems traditional symmetric multiprocessing (SMP) attains the first goal, but fails the second one.

3.2 Problem:

Symmetric multiprocessing with homogeneous cores can be done using complex or simple processors, but the use of only one type of core is inefficient for the diverse performance requirements of these particular systems

3.3 Forces:

- The Complex processors, who have speculative execution and out-of-order instruction sequence execution, are better for computation intensive applications
- The Simple processors, who don’t have speculative execution and execute the instruction sequences as they appear in the executable image (in-order execution), are better for low power applications
- For irregular control flow software execution, speculative and out-of-order execution performs poorly. For this type of computation, complex cores perform no better, and sometimes worse, than simple cores.

3.4 Solution:

With asymmetric multiprocessing the processor dynamically switches between big (complex, out-of-order) and little (simple, in-order) cores according to workload phases. There are two types of app diversity that provides AMPs many opportunities to switch. The first type of diversity is inter-application diversity. Some apps, such as video and web browsers, are compute-intensive and require big cores; some apps, such as a calendar, require minimal computation and can be performed on little cores. Loosely coupled big and little cores that share memory are sufficiently responsive to inter-application diversity. The second type of diversity exists inside individual apps: intra-application diversity. For example, a mobile app normally contains more than one "activity window," allowing users to perform distinct types of actions such as scrolling, reading and typing. Different actions correspond to different patterns of computation. It's observed that user input often triggers a short burst of processor activity, executing a few billion instructions in a few seconds. Input-triggered computation exhibits irregular control flow which provides opportunities for tightly coupled AMPs that share a cache hierarchy and switch quickly between execution modes. [5]

3.5 Remarks:

The AMP Processors can be implemented in different ways, some are hardware related:

- AMP with respect to the clock speed: All the cores in the AMP are equal, but some run with a slower clock speed to reduce the power consumption. This AMP is tailored for systems that need a complex core for software execution but also need a low power functionality that is transparent to the software

- AMP in hardware: There are cores of different kind inside the same AMP (i.e. 64 bit cores and 32 bit cores). These AMPs are appropriate to reduce the power consumption and meet real time requirements

Others are Operating System related:
- AMP with respect to the Operating System (O.S.): All the cores in the AMP are equal, but some run a different operating system. They are used where there are real time requirements, so one group of cores run a high level OS like Microsoft Windows or Linux for tasks that don't have real time requirements; and other group of cores run a Real Time Operating System (RTOS) for tasks that have real time requirements

And others are Instruction Set Architecture (ISA) related:
- AMP with the same ISA: The different cores have the same ISA, so the same software can be executed in any core. In these AMPs the software is agnostic about the core where it's running, so the same software can run in a high performance or in a low power environment.

- AMP with different ISA: Each type of core must run its own version of the software (the software must be compiled for the specific core's ISA). These AMPs are well suited when the hardware is very application specific (i.e. GPU for video processing or a DSP for signal processing applications)

The alias name of this pattern comes from the fact that when we use an AMP processor we can't expend the energy that a complex processor would require, but we need the best performance achievable for the system taking into account the low power limitation.

4. PATTERN "MINI ME"

4.1 Context:

In a battery power embedded system (i.e. the cell phone), there are long periods of low activity interrupted by short periods of very intensive activity that can't be predicted in advance, Some tasks need to run continuously (for example the ones related to the cell network management). These tasks must be executed in both periods: when the cell phone is idle and when the user is using the cell phone with a specific purpose (phone call, game play, document reading, etc)

4.2 Problem:

It's necessary to preserve the battery power of the system and there are tasks that must be executed in periods of very intensive activities as well as low activities.

4.3 Forces:

- Low power cores are better for power saving in low activity periods
- High performance cores are better for application performance in high activity periods
- There are tasks that must run in both kind of periods

4.4 Solution:

Implement the system using an AMP processor with high performance cores and low power cores; because both kind of cores have the same ISA, for the point of view of the software that must be executed in both periods, it is indifferent in which core is running at any time.

4.5 Remarks:

A commercial AMP processor that applies this pattern is the Samsung Exynos 5 Octa [6], which has 4 high performance cores (ARM Cortex A15 [7]) and 4 low power cores (ARM Cortex A7 [8]) inside the AMP. Both kind of cores have the same processor technology and ISA (ARM V7-A) [9]. Other commercial processor that applies this pattern is the AllWinner A80 [10]. AllWinner processors are very popular in tablets and set-top boxes that run Android OS. ARM Company calls this pattern “The big.LITTLE Technology” [11]

The pattern’s name comes from “Dr. Evil” character from “Austin Powers” movie; who has a clone of himself, but with 1/8 of his height, he calls the clone “Mini Me” [12]. The low power cores are the “Mini Me” of the high performance cores.

5. PATTERN “OPTIMIZED EXCUTION”

5.1 Context:

In an embedded system based on an AMP processor where the computing units have different ISAs, and one ISA, called “A”, is a subset of the other ISA, called “B”; the software compiled for the “ISA A” cores can run in “ISA B” cores but not in the other way. Also there is no task creation in runtime; the tasks, their resources and the core where they are going to run can be defined at compile time.

5.2 Problem:

For tasks with strict real time requirements, when there are different ISA’s available and one is a subset of the other, it’s necessary to take advantage of the specific instructions of the bigger ISA to improve the execution performance of the task.

5.3 Forces:

- Using the bigger ISA improves the execution performance of a task, but forces the use of a specific core [13]; and this must be done at compile time.
- Using the smaller ISA gives the freedom that the task can be configured to run in any core; and the core assignment can be done at run time.

5.4 Solution:

For each task with strict real time requirements, the compiler options are configured to generate code for the biggest ISA and the task is configured to run in a core with that ISA, so these tasks runs optimally and it’s easier to reach their real time requirements.

5.5 Remarks:

This pattern generates an optimal core resources usage, but is necessary to assign each task to a specific core in compile time. Some examples of compiler options for specific Processor Architectures are:
- GCC (GNU Compiler): -march=<Architecture Name>
- ARMCC (ARM Compiler): --cpu=<Architecture Name>
- CL (Microsoft Visual Studio): /favor < Architecture Name >
- ICC (Intel Compiler): -m<Architecture Name> or /Arch: <Architecture Name>

The name “Optimized Execution” comes from the fact that in this pattern we are looking for software running in an optimal way based on the core where is running.

6. ANTIPATTERN “LEVEL DOWN”

6.1 Context:

In an embedded system based on an AMP processor where the computing units have different ISAs, and one ISA, called “A”, is a subset of the other ISA, called “B”; the software compiled for the “ISA A” cores can run in “ISA B” cores but not in the other way.

6.2 Problem:

For software with some performance requirements, when there are different ISA’s available and one is a subset of the other, if the task uses the smaller ISA (common to all cores in the system), a suboptimal code is generated.

6.3 Forces:

- Using the bigger ISA improves the execution performance of the software, but forces the use of a specific core or ISA.
- Using the smaller (and more generic) ISA gives the freedom that the software can run in any of the cores that support this ISA without recompiling it or having a specific binary image for each type of core, so we have only one binary image to keep track.

6.4 (Bad) Solution:

The compiler options for the software are configured so the executable code has the ISA common to both type of cores, and suboptimal code is generated. So it could be difficult to reach real time requirements; or could be a wasted core resource.

6.5 Remarks:

This antipattern can be shown in libraries that have an intensive use of floating point operations (FP), like graphics libraries, video codecs and similar, who were compiled with FP support on software (SoftFP). In this case, the FP operations are emulated with operation using other kind of data types; but these libraries could run in cores with a hardware FP unit, which is not utilized because the FP operation is emulated in software. When the library is compiled with hardware FP support (HardFP), there is an increase in the library’s performance in cores with a hardware FP unit.

The same situation arises when the library is compiled for the common ISA for both types of cores. For example, the compiler options are tailored for the ARMv6a ISA, but the library is running in an ARMv7a core. When the library is compiled for the ARMv7a ISA, there is an increase of the library’s performance. In a video codec, we can see an increase in the frames/second rate that can be processed by the library

The name “Level Down” comes from the fact that when the software is compiled for the common ISA, the code is “mediocre” for the more powerful cores

7. PATTERN “DEDICATED PROCESSOR”

7.1 Context:

There is an AMP embedded system that runs a high level OS (Windows/Linux). Some tasks have hard real time requirements, some have soft real time requirements; and other tasks don’t have real time requirements at all.

7.2 Problem:

The real time requirements of all tasks must be reached without wasted core resources or risk of not reaching them because the core where the task is running is overloaded.

7.3 Forces:

- Inside the AMP, each type of core’s ISA is not compatible with the ISA of the other cores.
- Each type of core is optimized for a different kind of task.

7.4 Solution:

Each core must be used for a specific purpose. In this way, it is possible to reach the different real time requirements without wasted core resources for suboptimal core usage and without risk of not reaching the hard real time requirements because a core is used beyond its computing capacity

7.5 Remarks:

A commercial AMP that applies this pattern is the Sitara AM5728 from Texas Instruments [14], which has Two ARM Cortex-A15 (ISA ARMv7a) cores, Two ARM Cortex-M4 (ISA ARMv6m) cores, Two C66x (Floating Point Digital Signal Processor) cores, Two graphical processing units (GPU SGX544) and Four generic 32 bit cores (PRU), totalizing twelve cores with five different processor architectures.

The Cortex-A15 cores are intended for use with a high level OS like Linux; The Cortex-M4 cores are designed for tasks with hard real time requirements; The C66x cores are optimal for tasks with heavy FP usage; The SGX544 GPU cores are intended for 2D & 3D hardware accelerated graphics; And the generic 32 bit cores are optimal for tasks with soft real time requirements, like industrial communications (PROFIBUS or CAN)

This AMP is use on the Beagleboard X15 board [15]. The Beagleboard systems are very popular in the open source world for embedded systems implementations

The name “Dedicated Processor” comes from the fact that in this pattern we are looking for that each type of task runs in the core most optimized for that type of task.

ACKNOWLEDGES

- To Alejandra Garrido, who introduced me into the world of software patterns.
- To Robert Hanmer for his time and efforts shepherding this work.

REFERENCES

[1]: *“Embedded Systems Design” 2nd Ed.* Steve Heath. Elsevier, (2002)
[2]: *“The impact of performance asymmetry in emerging multicore architectures”*. Balakrishnan, Rajwar, Upton, Lai. *2005. Proceedings of the 32nd International Symposium on Computer Architecture (ISCA '05)*. IEEE. (2005)
[3]: *“Maximizing power efficiency with asymmetric multicore systems”* Fedorova, Saez, Shelepov, Prieto. *Communications of the ACM Volume 52 Issue 12* (2009).
[4]: *“Patterns for fault tolerant software”*, Robert S. Hanmer. Wiley Series in Software Design Patterns. John Wiley & Sons (2007)
[5]: *“Evaluating Asymmetric Multiprocessing for Mobile Applications”* Songchun Fan and Benjamin Lee. The IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS’16), IEEE. (2016)
[6]: https://en.wikipedia.org/wiki/Exynos (retrieved July 2016)
[7]: https://en.wikipedia.org/wiki/ARM_Cortex-A15 (retrieved July 2016)
[8]: https://en.wikipedia.org/wiki/ARM_Cortex-A7 (retrieved July 2016)
[9]:https://wiki.ubuntu.com/Specs/M/ARMGeneralArchitectureOverview?action=AttachFile&do=get&target=ARMv7+Overview+a02.pdf (retrieved July 2016)
[10]: https://en.wikipedia.org/wiki/Allwinner_Technology#A8x_family (retrieved July 2016)
[11]: https://en.wikipedia.org/wiki/Mini-Me (retrieved July 2016)
[12]: https://www.arm.com/products/processors/technologies/biglittleprocessing.php (retrieved August 2016)
[13]: *“A survey of multicore processors”*, Blake, G., Dreslinski, R. G., & Mudge, T. IEEE Signal Processing Magazine, 26(6), 26-37. (2009).
[14]: http://www.ti.com/product/AM5728 (retrieved July 2016)
[15]: https://en.wikipedia.org/wiki/BeagleBoard#BeagleBoard_X15 (retrieved July 2016)