ISBN: 978-987-46297-1-5 E-Book: 978-987-46297-2-2 Printed IEEE CATALOG: CFP1746V-PRT IEEE XPLORE: CFP1746V-ART

Software Patterns for Asymmetric Multiprocessing Devices on Embedded Systems: a performance assessment

Pedro Ignacio Martos
GPSIC & LSE – Facultad de Ingeniería
Universidad de Buenos Aires
Ciudad Autónoma de Buenos Aires, Argentina
pmartos@fi.uba.ar / pimartos@gmail.com

Alejandra Garrido
LIFIA – Facultad de Informática
Universidad Nacional de La Plata & CONICET
La Plata, Prov.de Buenos Aires, Argentina
garrido@lifia.info.unlp.edu.ar

***Abstract*—In embedded systems there is a variant of Multicore System on Chip devices (MSoC devices) where not all the computing elements (processor cores) are equal. The differences in the cores of these devices range 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 order to help developers to take advantage of the possibilities that these devices may offer in the context of embedded systems, software design patterns have been defined, describing software architectural solutions with known uses. However, there are still no experimental results showing the benefits of these solutions. In this work we measure the performance of a design pattern called Mini Me, applied on an AMP device configuration, and compare it against two Symmetric Multiprocessing Device (SMP Device) configurations. The evaluations show a better than expected computing performance of the AMP Configuration using the design pattern Mini Me.**

Keywords— Embedded Systems Patterns; Asymmetric Multiprocessing Patterns.

I. INTRODUCTION

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 they have been shown to provide improved computing performance [2], and also in the (performance)/(power) and (performance)/(silicon area) ratios [3], so they are attractive for E.S. implementations. However, designing an E.S. that can take advantage of AMPs' improved performance is hard. For example, performance asymmetry may adversely affect behavior of many workloads on commercial servers and make them less scalable [2].

With the goal of helping developers identify good architectural decisions when implementing software on AMPs and take advantage of their very specific characteristics, we have been working on the specification of a pattern language for AMP Embedded Systems [4]. A pattern language is a collection of interconnected design patterns or good practices in a specific domain [5]. Each design pattern in the language identifies a recurrent solution to a problem in a specific context [6]. Thus, a pattern conveys a small nugget of design and architectural knowledge to solve a problem within a certain context and after resolution it leaves the system in a new context, where there are new problems to be resolved by the other patterns in the language [7].

In this work, we apply one of that software patterns, called "Mini-Me", on an AMP configuration, to evaluate its performance against a traditional Symmetric Multiprocessing (SMP) configuration. In a few words, the pattern Mini-Me proposes to address the low power vs. high performance requirements using an AMP with high performance cores and low power cores with the same ISA. Thus, the low power cores become a "mini" version of the high performance cores.

In the work by Balakrishnan et al., authors demonstrate that using an AMP configuration causes a performance gain with generic workloads of multithreaded commercial applications [2]. In their experiment, they approximate performance asymmetry by varying the individual processor frequencies in a multicore system, i.e., varying the speed of clocks in each processor. Similarly, the purpose of our study is to evaluate if an AMP still gives higher performance when using a "Mini-Me" configuration running an E.S. Thus, we intend to answer these 2 questions:

Q1. Is performance still "better than expected" in an AMP configuration where asymmetry comes from processors with different hardware? (as opposed to processors with the same hardware but different clock speeds)

ISBN: 978-987-46297-1-5 E-Book: 978-987-46297-2-2 Printed IEEE CATALOG: CFP1746V-PRT IEEE XPLORE: CFP1746V-ART

Q2. Is improved performance sustained when the workload is specific to an E.S.? (as opposed to being a generic workload)

To answer these questions we use a computing platform for E.S. containing 4 complex cores and 4 simple cores, which have the advantage that processors can be turned on/off dynamically, thus facilitating the configuration of a symmetric or asymmetric platforms. Moreover, we use a well-known E.S. multiprocessor benchmark suite for our study: ParMiBench [8].

The rest of this paper is structured as follows: Section 2 presents related work. Section 3 describes the benchmark and details of the performance assessment. Section 4 shows the experimental results and Section 5 presents the conclusions and future work. Finally, we include an Appendix with the complete description of the pattern Mini-Me and its template for the sake of self-containment.

II. Related Work

The AMP processors are studied as general computing elements because of the advantages that they offer, in particular, their computing performance is over the expected average. Studies show that the AMP configurations with two complex processors and two simple processors have better performance than the average of four complex processors and four simple processors over a wide variety of general computing loads (application server, database server, web server, scientific computing, video compression, massive software compilation) [2, 3]. In Fig.1 (extracted from [2]) we can see the performance comparison between SMP and AMP architectures for the different general computing loads. In that work, the AMP architecture was made by reduction of the system clock of some processors, so nf/ms scale means n fast cores and m slow cores running at 1/scale the speed of fast cores (all the cores are equal except for their clock speed). The total computing power of a system of these characteristics is (n+m/scale). Symmetric configurations were 4f-0s; 0f-4s/4and 0f-4s/8 and asymmetric configurations were 3f-1s/4; 3f-1s/8; 2f-2s/4; 2f-2s/8; 1f-3s/4 and 1f-3s/8

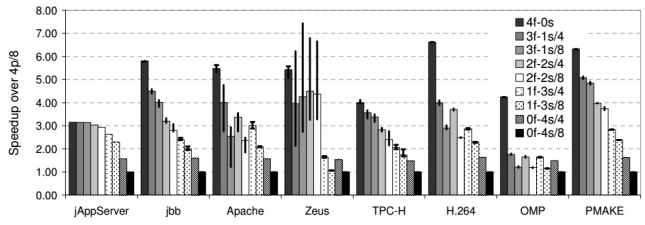

Fig. 1. Performance scalability for different SMP and AMP configurations.

Those results are valid for a general purpose computing device using standard workloads. However, in order to get results representative of the E.S. domain, we need to use a benchmark with E.S. oriented tasks. One such benchmark is ParMiBench [8], which is a multicore evolution of the MiBench [9] benchmark, a well-known suite used to evaluate uniprocessor E.S. performance.

As we explained in the introduction, a shortcoming of the work of Balakrishnan et al. for our purpose is that they approximate performance asymmetry by varying clock speed in each individual processor of a multicore system. While that can give a good approximation, we aim at evaluating performance when the hardware of the processors is different.

Furthermore, we have been working on the specification of a pattern language for AMP Embedded Systems [4]. This pattern language contains architectural software patterns, i.e., patterns at the level of architectural design, which are intended to help developers take advantage of the specific characteristics of asymmetric platforms in the design of E.S. Other works on pattern languages related to this domain are Hanmer's patterns for fault tolerant software [7] and White's patterns for embedded systems [10]. One of the patterns in the language of architectural patterns for AMP E.S. is "Mini-Me" [4], which is used in the context of a battery powered E.S., with long periods of low activity interrupted by short periods of very intensive activity. This pattern addresses the problem of preserving battery power when there are tasks that must be executed in both periods. Our intention in this work is to evaluate the performance of the architecture proposed by Mini-Me in the context of E.S.

III. Performance Assessment

The hardware platform used in our experiment was an Odroid XU4 Single Board Computer (SBC) [11], using a Samsung Exynos 5422 octacore Processor, which has four complex cores (ARM Cortex A15@2GHz) and four simple cores (ARM Cortex A7@1.4GHz). Both type of cores share the same Instruction Set Architecture (ISA), so the software is not aware about the kind of core where it is running, thus making the platform well suited to test the Mini-Me Software Pattern implementation. The Operating System was a vanilla Ubuntu Mate 16.04 tailored for that specific hardware platform obtained from the hardware platform web site. No special configuration / optimizations were made during the tests except to turn on/off the processors to implement the SMP and AMP configurations [12].

The ParMiBench [8] [13] [14] test suite has four E.S. subdomains: Automotive, Networks, Office and Security:

The **Automotive** tests are "BasicMath", which performs simple mathematical calculations, e.g., cubic function solving, angle conversions from degrees to radians, and integer square root. The input data set used for benchmarking is a fixed set of constants; and "Susan", which is an image recognition application for recognizing corners and edges. The input data is a complex picture.

The **Network** category represents embedded processors in network devices like switches and routers. The work done by these embedded processors involves shortest path calculations, tree and table lookups, and data input/output. The algorithms used to demonstrate the networking category are finding a shortest path in a graph and creating and searching a "Patricia tree" data structure. The "Dijkstra" benchmark constructs a large graph in an adjacency matrix representation and then calculates the shortest path between every pair of nodes using repeated applications of Dijkstra's algorithm. Dijkstra's algorithm is a well known solution to the shortest path problem and completes in $O(n^2)$ time. The "Patricia" test implements a Patricia tree: a data structure used in place of full trees with

ISBN: 978-987-46297-1-5 E-Book: 978-987-46297-2-2 Printed IEEE CATALOG: CFP1746V-PRT IEEE XPLORE: CFP1746V-ART

very sparse leaf nodes. Branches with only a single leaf are collapsed upwards in the tree to reduce traversal time at the expense of code complexity. Often, Patricia trees are used to represent routing tables in network applications. The input data for this benchmark is a list of IP traffic from a highly active web server for a 2 hour period. The IP numbers are disguised.

The **Office** category includes text manipulation algorithms to represent office machinery like printers and scanners with OCR recognition. The "StringSearch" benchmark finds a specific word in a number of given phrases by employing case sensitive or insensitive comparison algorithms.

The **Security** test is a SHA hash algorithm that produces a 160-bit message digest for a given input. It is often used in the secure exchange of cryptographic keys and for generating digital signatures. It is also used in the well-known MD4 and MD5 hashing functions. The input data set is a large ASCII text file of an article found online.

We executed the ParMiBench suite in three different configurations (two SMP and one AMP). The SMP configurations were 4c0s and 0c4s; and the AMP configuration was 2c2s. A XcYs configuration means X complex processors (ARM Cortex A15) and Y simple processors (ARM Cortex A7). It must be noted that in this particular SoC (Exynos 5422) the hardware interrupt controller is hardwired to CPU0 (a simple Cortex A7 processor). Thus, in practice it is impossible to disable CPU0 at all (because this also would disable all system hardware interrupts); so the configuration 4c0s is in fact a 4c1s with the benchmark running in the four complex processors. To assess the influence of that simple core, we also run the benchmark in the 4c4s full SMP configuration, and we verified that changing from 4c1s to 4c4s has only a very modest 5% in performance increase. Thus, we can conclude that the influence of 3 simple cores when the benchmark runs in the 4 complex cores has very little impact in the benchmark results. Therefore, the influence of 1 simple core will be less than that and it is valid to consider the configuration 4c1s equal to 4c0s for the benchmark results.

For each system configuration (SMP and AMP) we obtained the total execution time, and to take into account execution time variances from the operating system, each test was executed 10 times. For each test in each configuration we obtained the shortest, the largest and the average execution times; and also the total benchmark execution time (10 runs of each test).

IV. Experimental Results

Tables 1 to 6 show the minimum, maximum, average and total execution time for each test of the benchmark over the three configurations. The tables have two extra columns: one with the expected performance and the final with the improvement over the expected performance of the AMP configuration. The expected performance function *Exp.Perf(2c2s)* is the average between the SMP complex configuration (4c0s) performance, *Perf*(4c0s) and the SMP simple configuration (0c4s) performance *Perf*(0c4s), as defined in Equation 1:

$$Exp.Perf(2c2s) = [Perf(4c0s) + Perf(0c4s)] / 2 \quad (1)$$

Where *Perf* is the performance (execution time) of the particular configuration. Meanwhile, the improvement over the expected performance *AMP_Impr()* function is the ratio between the expected performance of the 2c2s AMP configuration and the real performance of that configuration, as defined in Equation 2:

$$AMP\_Impr() = Exp.Perf(2c2s) / Perf(2c2s) \quad (2)$$

Table 1: BASICMATH

	4c0s	2c2s	0c4s	Exp.Perf	AMP Imp.
Min.	140.45	180.13	227.37	183.91	1.02
Max.	141.46	182.72	230.74	186.10	1.02
Avg.	140.79	181.25	229.66	185.23	1.02
Total	1407.90	1812.54	2296.64	1852.27	1.02

Table 2: SUSAN

	4c0s	2c2s	0c4s	Exp.Perf	AMP Imp.
Min.	0.0060	0.0060	0.0097	0.0079	1.31
Max.	0.0071	0.0090	0.0115	0.0093	1.04
Avg.	0.0062	0.0066	0.0104	0.0083	1.25
Total	0.0620	0.0664	0.1038	0.0829	1.25

Table 3: DIJKSTRA

	4c0s	2c2s	0c4s	Exp.Perf	AMP Imp.
Min.	0.0181	0.0179	0.0293	0.0237	1.33
Max.	0.0210	0.0219	0.0335	0.0273	1.25
Avg.	0.0193	0.0190	0.0310	0.0252	1.33
Total	0.1928	0.1928	0.3104	0.2516	1.30

Table 4: PATRICIA

	4c0s	2c2s	0c4s	Exp.Perf	AMP Imp.
Min.	0.0060	0.0059	0.0096	0.0078	1.33
Max.	0.0071	0.0080	0.0129	0.0100	1.25
Avg.	0.0063	0.0064	0.0101	0.0082	1.29
Total	0.0628	0.0635	0.1013	0.0821	1.29

Table 5: STRINGSEARCH

	4c0s	2c2s	0c4s	Exp.Perf	AMP Imp.
Min.	213.77	255.52	599.49	406.63	1.59
Max.	216.51	259.57	608.80	412.65	1.59
Avg.	215.34	257.42	602.88	409.11	1.59
Total	2153.40	2574.25	6028.76	4091.08	1.59

Table 6: SHA

	4c0s	2c2s	0c4s	Exp.Perf	AMP Imp.
Min.	0.0623	0.0721	0.1354	0.0989	1.37
Max.	0.0765	0.0818	0.1650	0.1207	1.48
Avg.	0.0669	0.0759	0.1482	0.1075	1.42
Total	0.6026	0.7592	1.4819	1.0422	1.37

ISBN: 978-987-46297-1-5 E-Book: 978-987-46297-2-2 Printed IEEE CATALOG: CFP1746V-PRT IEEE XPLORE: CFP1746V-ART

In Table 7 we show the total execution time of the ParMiBench benchmark suite for the three configurations, the expected performance and the AMP configuration improvement over that expected value.

TABLE 7: PARMIBENCH TOTAL EXECUTION TIME

	4c0s	2c2s	0c4s	Exp.Perf	AMP Imp.
Tot.Ex.Time	3562.22	4387.87	8327.40	5944.81	1.35

In Figure 2 we show the average AMP configuration improvement for each type of test.

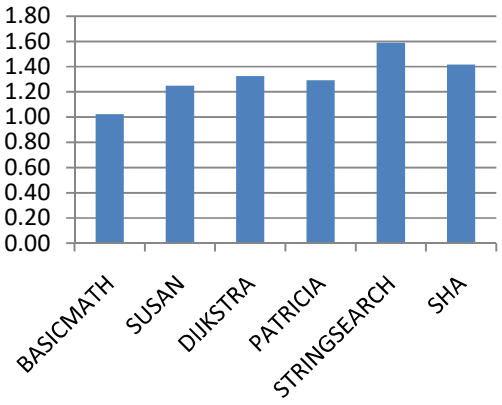

Fig. 2. AMP Configuration Improvement for each type of test

Analyzing the results in Tables 1 to 7 and in Figure 2 we can see that the BASICMATH test shows the expected performance improvement (the average between both SMP configurations), so for E.S. applications that make intense use of math, the use of the Mini-Me architectural pattern doesn't affect adversely the performance of the system. For the other tests, we found a performance improvement from 1.25 to 1.6 times over the expected performance; with an average of 1.4 times.

The reason for these improvements can be explained by the hardware differences between both processors [15]: Cortex A15 is a high end, triple-issue, out-of-order processor core which also implements virtualization instructions, hardware-accelerated integer division, and 40-bit virtual memory addressing extensions, so it has a very good single-threaded peak performance [16]. Figure 3 from [15] shows the Cortex A15 instruction pipeline.

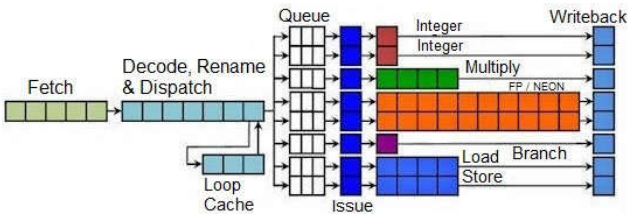

Fig. 3. ARM Cortex A15 Instruction Pipeline

On the other side, Cortex A7 processor is an in order, partial dual issue machine. The dual integer pipelines are eight stages long; the Cortex A7 combines full ALU (labeled "integer" in Figure 4 below) and partial ALU (labeled "dual-issue") structures, thereby enabling dual issue instruction execution for some integer operations. However, both conventional multiplication and NEON SIMD operations are single issue only. These architectural differences between both processors, and taking into account power and die area, generate a roughly equivalence of one Cortex A15 to four Cortex A7 [16]. Figure 4 from [15] shows the Cortex A7 instruction pipeline.

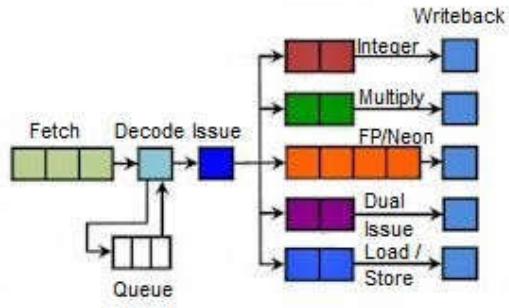

Fig. 4. ARM Cortex A7 Instruction Pipeline

Therefore, a SoC that implements two Cortex A15 and two Cortex A7 processors will have a better than expected performance because the Cortex A15 processors will have a high single thread peak performance and the Cortex A7 will produce a high multithreading peak performance.

V. CONCLUSIONS AND FUTURE WORK

We advocate for the definition and use of design patterns, which carry the knowledge of experts in a specific domain, thus helping developers to understand the problems that they may encounter in the domain, and the possible solutions with their weighted consequences. Moreover, in the particular context of E.S., it is very important to have real performance measurements of the solutions proposed by each pattern, and this is, to our knowledge, the first work on that direction.

In this work we measured the performance of the design pattern called Mini Me, using a benchmark for E.S. oriented tasks. Our evaluation was guided by two research questions. The first question: "Is performance still "better than expected" in an AMP configuration where asymmetry comes from processors with different hardware?" has been proved to be true by our experiment, as we still have a "better than expected" performance in a AMP system. The second question: "Is improved performance sustained when the workload is specific to an E.S.?" has been also found to be true, as results show that we can expect a performance improvement of about 1.4 times on average over the expected performance of the AMP system.

Future work includes running experiments on other design patterns in the pattern language for AMP-based embedded systems and extending, as well as discovering other patterns in the language.

ISBN: 978-987-46297-1-5 E-Book: 978-987-46297-2-2 Printed IEEE CATALOG: CFP1746V-PRT IEEE XPLORE: CFP1746V-ART

REFERENCES

[1] Steve Heath, Embedded Systems Design, 2nd Ed. Elsevier, (2002)

[2] Balakrishnan, Rajwar, Upton, Lai, The impact of performance asymmetry in emerging multicore architectures, Proceedings of the 32nd International Symposium on Computer Architecture (ISCA '05). IEEE. (2005)

[3] Fedorova, Saez,Shelepov, Prieto, Maximizing power efficiency with asymmetric multicore systems, Communications of the ACM, Vol 52 Issue 12 (2009)

[4] P. Martos, Architectural Patterns for Asymmetric Multiprocessing Devices on Embedded Systems, Proceedings of the 11th Latin American Conference on Pattern Languages of Programs (SugarLoaf PLoP '16). Hillside Group (2016)

[5] R. Hanmer, Pattern-Oriented Software Architecture For Dummies, John Wiley & Sons, 2013

[6] Gamma, Help, Johnson, Vlissides. Design Patterns. Addison-Wesley, 1995.

[7] Robert S. Hanmer, Patterns for fault tolerant software, Wiley Series in Software Design Patterns. John Wiley & Sons (2007)

[8] S.M.Z.Iqbal, Y.Liang, H.Grahn.,"ParMiBench – An Open-source Benchmark for Embedded Multiprocessor Systems", IEEE Computer Architecture Letters Vol 9 Issue 2. IEEE (2010)

[9] M.R.Guthaus, J.S.Ringenberg, T.Austin, T.Mudge, R.B.Brown, MiBench: a Free, commercially representative embedded benchmark suite",.Proc. of the IEEE International Workshop on Workload Characterization (WWC-4), IEEE (2001)

[10] E. White, "Making Embedded Systems: Design Patterns for Great Software", O'Reilly Media (2011)

[11] "ODROID-XU4", http://www.hardkernel.com/main/products/prdt_info.php, (retrieved May,2017)

[12] https://forum.odroid.com/viewtopic.php?f=93&t=16525 (retrieved May,2017)

[13] https://sites.google.com/site/parmibench/ (retrieved May,2017)

[14] https://github.com/cota/parmibench (retrieved May,2017)

[15] https://www.bdti.com/InsideDSP/2011/11/17/ARM (retrieved June,2017)

[16] http://www.eetimes.com/author.asp?section_id=36&doc_id=1318968 (retrieved June, 2017)

APPENDIX A. PATTERN "MINI-ME"

This section presents the description of the pattern evaluated in this work, to make the paper self-contained.

First, it is important to note that the patterns in the language for AMP-based E.S. are described with a specific template, as follows:

- Context: summarizes situations in which you may find the pattern useful
- Problem: provides a brief summary of the problem which is addressed by the pattern
- Forces: describe the conflicts of interest occurring in the problem
- Solution: describes how the pattern offers a solution that balances the forces in the problem
- Consequences: this section describes how the use of the pattern affects the system.
- Hardware implications: The application of the pattern has some hardware implications that should be taken into account.
- Portability: In E.S, it is common to have to port the software to other hardware platforms; this section describes how the pattern affects the portability.
- Overall strengths and weaknesses: summarizes pros and cons of the pattern.
- Related patterns and alternative solutions: discusses alternative solutions and/or related patterns that could be of interest.

PATTERN "MINI-ME"

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, gameplay, document reading, etc).

Problem: When running on a battery powered embedded system, 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.

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.
- When different cores have the same ISA, the software is easier to develop and maintain (software upgrades).

Solution: Implement the system using an AMP processor with high performance cores and low power cores with the same ISA, because the way to preserve battery power is using a processor with less power requirements. Also, because both kind of cores have the same ISA, for the point of view of the software that must be executed in both periods (high activity and power saving), it is indifferent in which core is running at any time. When both type of cores share the same ISA, the low power cores became a smaller version of the high performance cores, so the low power cores are "mini" high performance cores.

Consequences: Because both type of cores share the same ISA, the operating system has no restriction about the core where a process/task can run. But when the process/task run in the low power core, there is a performance penalty, so the OS scheduler should take this into account for the process/task execution priority and processor time allocation.-

Hardware implications: Some SMP multicores have a configuration register for the system clock of each core. By configuring different values for that register, we can convert a

ISBN: 978-987-46297-1-5 E-Book: 978-987-46297-2-2 Printed IEEE CATALOG: CFP1746V-PRT IEEE XPLORE: CFP1746V-ART

SMP system in a AMP system where all cores have the same ISA.

Portability: As both type of cores share the same ISA, the software can run on any core without modification.

Overall Strengths and Weaknesses:

+ The AMP with cores having the same ISA has a minimum impact on the software that runs on it and the system has less power consumption.

- There is a performance penalty when the process/task runs on the low power cores and the operating system could not be aware of that.

Related patterns and alternate solutions: This pattern is well suited when the processes/tasks don't have real time requirements, because they run sometimes in a high performance core and sometimes in a low power core, so the execution predictability and performance could be very difficult to establish. For this type of processes/task, the "Dedicated Processor" or "Optimized Execution" patterns would be better. This pattern is a specialization of the "Asymmetric Multiprocessing" pattern.

Known Uses: A commercial AMP processor that applies this pattern is the Samsung Exynos 5 Octa, which has 4 high performance cores (ARM Cortex A15) and 4 low power cores (ARM Cortex A7) inside the AMP. Both kind of cores have the same processor technology and ISA (ARM V7-A). Other commercial processor that applies this pattern is the AllWinner A80. AllWinner processors are very popular in tablets and set-topboxes that run Android OS. ARM Company calls this pattern "The big.LITTLE Technology". 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". The low power cores are the "Mini Me" of the high performance cores.