Using Amdahl's Law for Performance Analysis of Many-Core SoC Architectures Based on Functionally Asymmetric Processors

Hao Shen and Frédéric Pétrot

System Level Synthesis Group, TIMA Laboratory
CNRS/Grenoble INP/UJF
46, Avenue Félix Viallet, 38031, Grenoble, France
{hao.shen,frederic.petrot}@imag.fr

Abstract. Amdahl's law is a fundamental tool for understanding the evolution of performance as a function of parallelism. Following a recent trend on the timing and power analysis of general purpose many-core chips using this law, we carry out an analysis aiming at many-core SoCs integrating processors sharing the same core instruction set but each potentially having additional extensions. For SoCs targeting well defined classes of applications, higher performances can be achieved by adding application specific extensions either through the addition of instructions in the core instruction set or through coprocessors leading to architectures with functionally asymmetric processors. This kind of architectures is becoming technically viable and advocated by several groups, but the theoretical study of their properties is yet to be performed: this is precisely our goal in this paper. We use Amdahl's law to prove the performance advantage of using extensions for many-core SoCs and shows that the many-core architecture based on functionally asymmetric processors can achieve the same performance as the symmetric one but at a lower cost.

1 Introduction

Following Moore's law, processor frequency doubled every 18 to 24 months until the mid 2000's. Due to the ever increasing core design complexity of the high performance processors (run-time data dependency analysis, speculative execution, branch prediction, ...) and the power consumption caused by the very high frequency, researchers started to look at other strategies to continue to increase system performances. The first solution is to optimize the instruction set for certain application classes. The generalization of the SIMD extensions, which first appeared in the general purpose high performance processors in the early 90's[1], to all processors including the embedded ones[2] is an evidence of this trend.

The second solution, straightforward from the point of view of the hardware designer, is to integrate several cores onto the same silicon die. Due to power dissipation issues, the integrated cores should feature a high performance per Watt ratio and an overall current consumption acceptable for the application. Following this trend, the ITRS predicts that a many-core era will arrive soon[3].

In the general and high performance computing domains, there is a clear trend towards *Chip Multiprocessor (CMP)* architectures. The processors integrated in these

M. Berekovic et al. (Eds.): ARCS 2011, LNCS 6566, pp. 38–49, 2011.

CMP architectures are symmetric in both performance and function. Amdahl's law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used [4]. Based on Amdahl's law, several works have defined performance asymmetric architectures which accelerate the sequential execution by using fast cores and the parallel execution by a massive usage of small cores having lower performance but better MIPS/Watt ratio than the fast ones. Therefore both the cost over performance ratio [5,6] and the power consumption [7] of these architectures are lower than the corresponding symmetric ones.

For embedded systems, the domain of applications is fixed and application specific optimizations must be used for the *Multiple Processors System-on-Chip* **(MPSoC)** to fit into the power and performance budget (no heat sink, no fan, battery operated, etc.). Extended instruction sets, either within the processor pipeline [8][9] or as a coprocessor [10] are two well accepted solutions which can improve the overall system performance and greatly lower its cost. Unfortunately, only part of the application can benefit from these extensions. For example, the edge detection instructions of the VIS extension of SPARC is useful only for texture mapping algorithms [11]. How to define an architecture which can benefit from these extensions for performance while minimizing the idle time of the associated resources is an open question in the many-core era.

The rest of this paper is organized as follows: section 2 discusses the related works about both performance asymmetric and functionally asymmetric architectures for embedded systems. In section 3, we use Amdahl's law to analyze the performance improvement with extensions and show the cost advantage with functionally asymmetric architecture. At last, section 4 concludes this paper and presents future works.

2 Related Works

Intrinsically all applications have sequential phases which limit the speedup that can be obtained by parallel execution on CMP architectures. Such a CMP architecture is presented in Fig. 1(d) where all cores are identical. An application that has both sequential and parallel phases executes at the same speed as shown with identical lines width in Fig. 1(a). The performance asymmetric many-core idea has been proposed to improve the overall performance [5] and save the system power consumption [7] under resource constraints. Fig. 1(e) shows such a performance asymmetric many-core architecture which includes a huge core and several small cores. The key reason for performance improvement is that such architectures can accelerate the sequential phase execution by using the fast/huge cores and the parallel phase execution by using a large number of lower performance small cores [6]. In Fig. 1(b), the sequential phase speedup with the fast core is shown with a wider line. All these researchers make the assumption that all processors have to implement the exact same instruction set with different performance and cost. It is thus possible to run, synchronize and migrate application threads with the same operating system.

In the embedded system domain, designers can benefit from extended instructions and coprocessors to improve the single processor performance for a specific application. The ARM NEON is a SIMD extension for multimedia and signal processing algorithms which can achieve 60-150% performance improvement on complex video codec algorithms [10]. The Tensilica [8] solution is very flexible and allows to make trade-offs between processor performance and cost of the extended extension. According to [8],

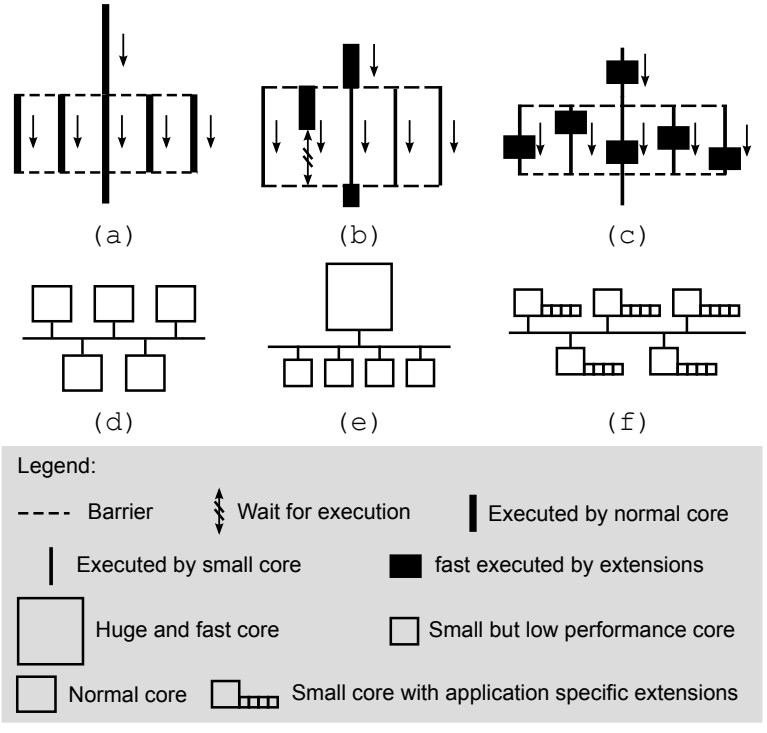

Fig. 1. (a)(d). Symmetric Many-Core Architecture and Parallel Application. (b)(e). Performance Asymmetric Many-Core Architecture and Parallel Application. (c)(f). Symmetric Many-Core Architecture with Extensions and Parallel Application.

with a cost of 4,500 gates (almost 18% of the 25k gates core), the implementation of the DES cryptographic algorithm reaches a 50x speedup. This approach can be extended to other applications such as digital filtering, Viterbi decoding, Motion estimation and so on, and the reported speedups range form 4 to 53 times. In the general computation domain, the authors [5] assume that if a computing resource of size 1 has a performance of 1, then when we scale up the computing resource to size r, it has only performance of $\sqrt{r}$. As introduced by the previous examples, compared with this general purpose processing resource/performance relationship, embedded systems can make a much better usage of the available resources. Fig. 1(f) introduces an architecture in which each core has an extension. The drawback is that only part of the application can be accelerated by these new resources (eg. extended instructions and coprocessors) making them useless for other parts of the execution. In Fig. 1(c), extensions that can partially speedup both the sequential and parallel phases with still wider lines. In the symmetric architecture shown in Fig. 1(f), as wide lines only occupy small parts of the whole execution process, the extensions have a low occupation rate and resources are wasted.

To increase the occupation rate of the extensions, several research works [12,13,14] propose a shared memory functionally asymmetric architecture composed of processors sharing a core instruction set but having each potentially different extensions. Fig. 2(a)

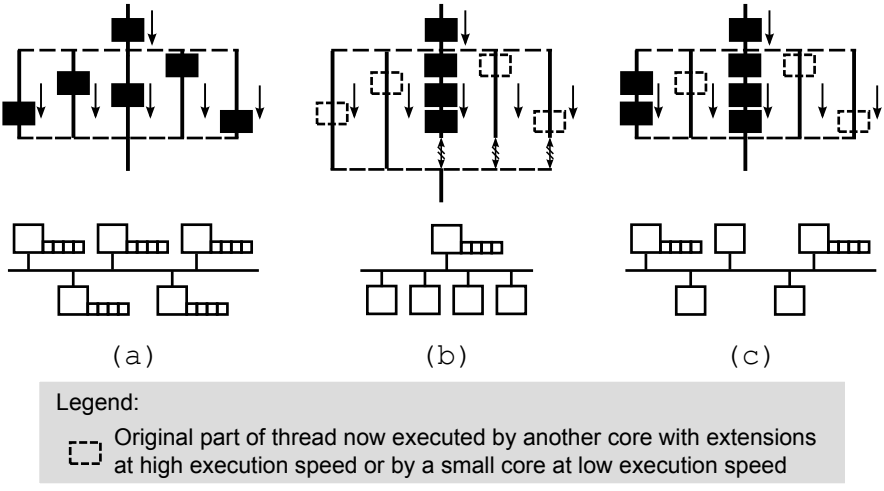

Fig. 2. (a). Symmetric Architecture VS. (b)(c). Functional Asymmetric Architecture.

presents the traditional symmetric architecture while Fig. 2(b) and Fig. 2(c) show the functionally asymmetric architectures in which some cores can avoid using extensions to save chip area and power consumption. This kind of architectures can improve the overall system performance by accelerating some critical parts of a parallel application while still providing very good flexibility.

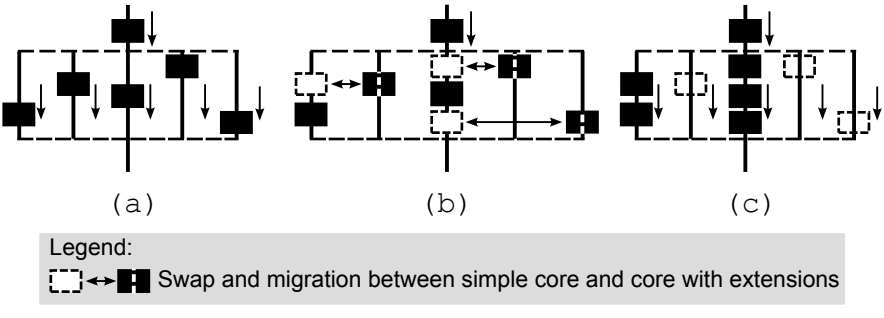

Fig. 3. Functional asymmetric architectures, (a). Traditional parallel execution of multiple threads. (b). With migration process capabilities, extension related executions are swapped to the cores with extensions. (c). The final execution status.

Fig. 3 shows that the execution of parallel application should be modified to adapt to the architecture changes. Task scheduling algorithms and migration frameworks proposed by [12][13] can fully use this kind of functionally asymmetric architectures by migrating threads between basic cores and cores with extensions presented in Fig. 2. The thread migration case shown in Fig. 3 is the ideal one where there are no timing conflicts between each two threads benefiting from the extensions for execution.

Unlike this optimal case, Fig. 4 shows an unfortunate one where all parallel threads are identical and conflicts accesses to the extensions cause a noticeable delay for the whole application. The Fig. 3 case and the Fig. 4 one are two extreme situations which will be discussed in the following theoretical analysis. Following the coarse grain analysis possible with Amdahl's law, the theoretical analysis using our abstract model ignores the context switches and cache flushes costs. Experimental results in which these overheads are accounted for can be found in [12] and [13].

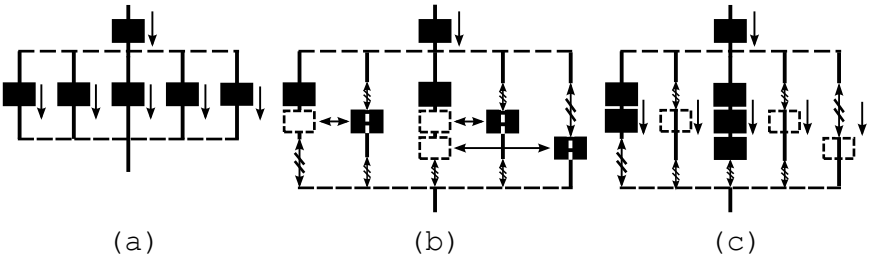

Fig. 4. (a) All executions that make use of the extensions occur at the same time for all threads. (b) Thread migration can solve the execution problem with some timing cost. (c) The final execution status.

To the best of our knowledge, this paper is the first work which analyzes and compares the performance limitation of both functionally asymmetric many-core architectures and traditional symmetric ones through an innovative usage of Amdahl's law.

3 Amdahl's Law and Functionally Asymmetric Many-Core Architecture

Amdahl's law is used to find the maximum expected improvement to the whole system when only part of the system is improved [4]. It is often used in parallel computing to predict the theoretical maximum speedup achievable by many-core architectures (see Fig. 1(d)). Using a many-core architecture targets to increase the *Thread Level Parallelism* **(TLP)**. The maximum theoretical speedup with N cores is given by Eq. 1. In this equation, f_{TLP} represents the fraction of the total program which can be accelerated by N parallel processors.

$$Speedup_{TLP}(f_{TLP}, N) = \frac{1}{(1 - f_{TLP}) + \frac{f_{TLP}}{N}} \tag{1}$$

Fig. 5 (a) shows the maximum speedup with different percentages of application parallelism rates (f_{TLP}). The figure shows the limitation of performance in the many-core era. This limitation can also be explained by using another parameter called *Occupation Rate* ρ. Occupation rate is commonly used in queueing theory to represent the fraction of time a server is working. Eq. 2 defines the relationship between ρ, the *arrival rate* λ, the *mean service time* $E(B)$ and the *number of servers* c. In our case, Eq. 2 can be interpreted as the time percentage a core or an extension is working during the whole

execution. In Fig. 1(a), among the five cores, only one core has a 100% occupation rate while others have low occupation rate due to lack of parallelism.

$$\begin{aligned}\rho &= \lambda E(B)/c \\ &= \frac{Total\ Time\ All\ Components\ Used}{(Time\ Whole\ Application\ Execution)(Processor\ Number)}\end{aligned} \tag{2}$$

$$\begin{aligned}\rho &= \frac{\frac{Whole\ Application}{Processor\ Number}}{Sequential\ Time + Parallel\ Time} \\ &= \frac{\frac{1}{N}}{(1 - f_{TLP}) + \frac{f_{TLP}}{N}} = \frac{1}{N \cdot (1 - f_{TLP}) + f_{TLP}}\end{aligned} \tag{3}$$

Eq. 3 refines the occupation rate parameter for the many-core TLP case. In this equation, as one application is distributed over N processors, the average used time for each one is $\frac{1}{N}$. Fig. 5(b) presents the decrease rate of the occupation rate when the core number scales up. The core occupation rate decreases really fast even for an application parallelism of 95%. When the number of cores reaches 1024, the occupation rate is close to zero. This figure indicates the symmetric architecture has huge drawback of cost efficiency and power efficiency in the many-core era because of the application sequential part (insufficient parallelism).

3.1 Extensible Processor and Functionally Asymmetric Many-Core Architecture

To avoid wasting power and area by using symmetric many-core, *Application Specific Integrated Processor* **(ASIP)** is a well accepted solution that has been introduced to improve the single core performance for domain specific applications. This extension method can also be explained by Amdahl's law for single core performance improvement by increasing *Instruction Level Parallelism* **(ILP)**. Here, ILP means the extended instructions and coprocessors can execute the equivalent of multiple original core instructions in one cycle. The extensions of embedded processor usually implement some sort of *single instruction, multiple data* **(SIMD)** instructions to increase ILP. For this ILP case, Amdahl's law can be expressed as shown in Eq. 4 where f_{ILP} represents the fraction of the total program can be sped up by S times with extended instructions/coprocessors.

$$Speedup_{ILP}(f_{ILP}, S) = \frac{1}{(1 - f_{ILP}) + \frac{f_{ILP}}{S}} \tag{4}$$

As Eq. 3, Eq. 5 refines Eq. 2 for the occupation rate in the ILP case. Because Eq. 5 focuses on the workload of extended instructions and coprocessors, the average time used by these extensions is defined as $\frac{f_{ILP}}{S}$.

$$\rho = \frac{\frac{f_{ILP}}{S}}{(1 - f_{ILP}) + \frac{f_{ILP}}{S}} \tag{5}$$

Based on these two equations, we also have the figure for both speedup and occupation rate in Fig. 5 (c) and (d). Identically to the increase in processor number, increasing

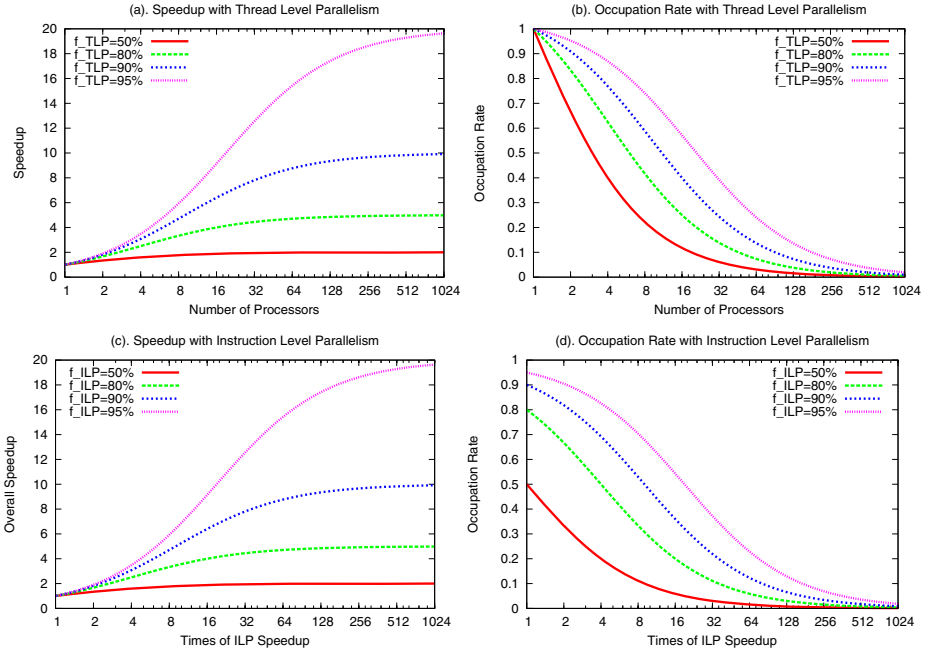

Fig. 5. Amdahl's law: speedup *VS.* occupation rate for TLP and ILP in the many-core era

the ILP can not avoid the drawback of low occupation rate even when the part of the application that can be sped up by the extensions reaches 95%.

3.2 Combination of TLP and ILP

To achieve higher performances in the embedded system domain, it is possible to combine both TLP (increasing the processor number) and ILP (adding application specific extensions) together.

We use the functionally asymmetric many-core architecture of Fig. 2(b), that has a core with extended instructions and coprocessors for better sequential performance. Beside this extended core, it has a group of small cores without any extensions to improve parallel performance. The parallel application can be executed as shown Fig. 2(b) and the speed up is given in Eq. 6.

$$
\begin{aligned}
&Speedup_{ILP\_TLP\_Sequential}(f_{ILP}, f_{TLP}, S, N) \\
&= \frac{1}{\frac{1-f_{TLP}}{Speedup_{ILP}(f_{ILP},S)} + \frac{f_{TLP}}{N-1+Speedup_{ILP}(f_{ILP},S)}} \\
&= \frac{1}{(1-f_{TLP}) \cdot ((1-f_{ILP}) + \frac{f_{ILP}}{S}) + \frac{f_{TLP}}{N-1+\frac{1}{(1-f_{ILP})+\frac{f_{ILP}}{S}}}}
\end{aligned}
\tag{6}
$$

In this equation, the sequential part $(1 - f_{TLP})$ is accelerated by extensions with $Speedup_{ILP}$ and the parallel part is executed by $N - 1$ small cores and the extended

core with the same $Speedup_{ILP}$. In this model, we make the assumption that all ILP which can be accelerated with extended instructions and coprocessors (f_{ILP} and S) are independent of the sequential and parallel execution (independent with f_{TLP} and N). Fig. 6 (a)(b)(c) shows the final speed up with functionally asymmetric many-core architecture compared with the architecture without ILP accelerations (red line in Fig. 6). We can also notice that it leads to higher performance compared with the TLP only and ILP only solutions in Fig. 5.

Though Fig. 6 already shows a good speed up by using a single extended core, we may want to speed up the parallel part of the application with extended instructions and coprocessors also. This leads to the architecture which includes symmetric many processors with the same extended instructions and coprocessors (Fig. 2(a)). Eq. 7 models the performance of this kind of architecture.

$$\begin{aligned} &Speedup_{ILP\_TLP}(f_{ILP}, f_{TLP}, S, N) \\ &= Speedup_{ILP}(f_{ILP}, S) \cdot Speedup_{TLP}(f_{TLP}, N) \\ &= \frac{1}{((1-f_{ILP}) + \frac{f_{ILP}}{S})((1-f_{TLP}) + \frac{f_{TLP}}{N})} \end{aligned} \tag{7}$$

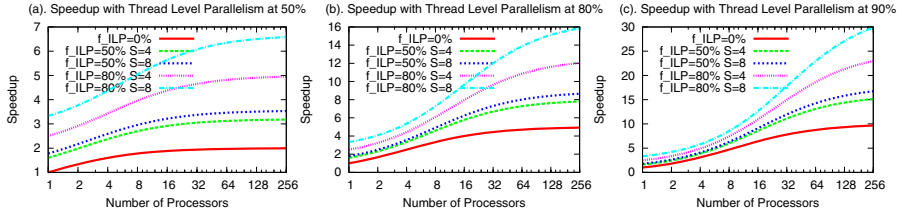

Fig. 6. Amdahl's law: speedup for functionally asymmetric many-core architecture with both TLP and ILP (1 extended core and $N-1$ simple cores)

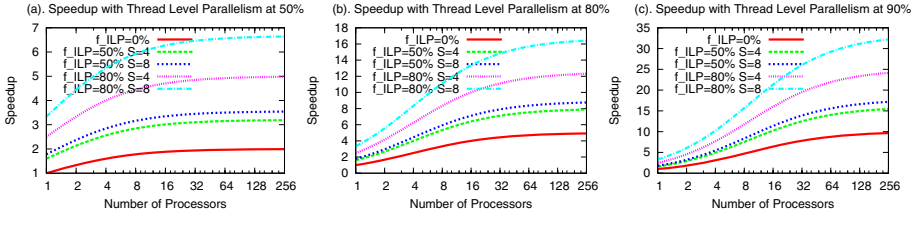

Fig. 7. Amdahl's law: speedup for symmetric MPSoC with both TLP and ILP (N extended cores)

In Fig. 7 (a)(b)(c), we show the speedup obtained by combining both TLP and ILP together with different percentages of f_{TLP}. Compared with Fig. 6, we can find the difference on the speedup where the number of processors ranges from 2 to 64. In this area, the symmetric many-core architecture has higher performance compared with the architecture with only one extended core of Fig. 6.

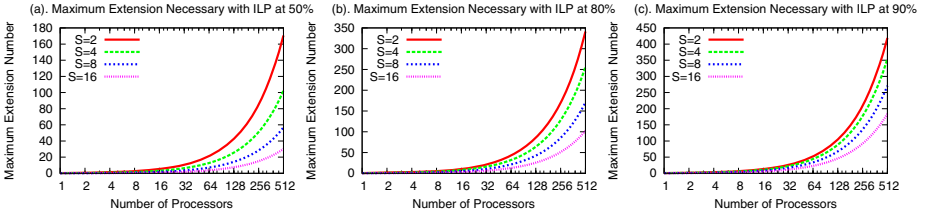

Fig. 8. Amdahl's law: ideal extensions needed to have the same performance for functionally asymmetric many-core architecture

3.3 Functional Asymmetric Architecture to Improve the Occupation Rate

For the model combining both TLP and ILP, we have the occupation rate of extended instructions/coprocessors defined by Eq. 8. In fact, this equation is just the same as Eq. 5. Because TLP is well separated with ILP, the occupation rate of extensions is independent of the processor number N and percentage of parallelism f_{TLP}. So Fig. 5 (d) is also suitable for this case.

$$\rho = \frac{\frac{f_{ILP}}{S}}{(1 - f_{ILP}) + \frac{f_{ILP}}{S}} \tag{8}$$

But we can use Eq. 8 to find how many extensions are enough to retain the same system performance with the functionally asymmetric many-core architecture. The idea is based on the ideal case shown in Fig. 3 where there is no conficts between two extension executions from different processors. Eq. 9 defines the overall requirement of extensions for multiple extensible MPSoC platforms when we take all N processors into account.

$$EXT_{ideal}(f_{ILP}, S, N) = \lceil \rho \cdot N \rceil = \lceil \frac{f_{ILP} \cdot N}{S \cdot (1 - f_{ILP}) + f_{ILP}} \rceil \tag{9}$$

Fig. 8 shows the number of extensions with different percentages of f_{ILP}. With the higher percentage of ILP (f_{ILP}) of an application, the more extensions are necessary to benefit from this speedup property. With fixed f_{ILP}, it is obvious to find that the more processors, the more extensions are needed. Because the occupation rate becomes low with a high speedup (S), with the higher speedup , the less extensions are required. As shown in Fig. 8, the necessary number of extensions is much lower than that of processors in the ideal case.

$$\begin{aligned} & Speedup_{ILP\_TLP\_CONFLICT}(f_{ILP}, f_{TLP}, S, N, X) \\ &= \frac{1}{Time_{Sequential} + Time_{Parallel}} \\ &= \frac{1}{(1 - f_{TLP}) \cdot ((1 - f_{ILP}) + \frac{f_{ILP}}{S}) + \frac{f_{TLP}}{N} \cdot ((1 - f_{ILP}) + \frac{f_{ILP}}{S} \cdot \frac{N}{Ext})} \end{aligned} \tag{10}$$

Different from this ideal case, we have the worst case shown in Fig. 4. In this case, system performance is decreasing because of the extension execution conflicts. We have the Eq. 10 to describe this case. In the end of this equation, N/Ext represents the delay caused by the extension execution conflicts and Ext is the extension number used in the architecture. In this research, we have the maximum value N for Ext which means the extension number is exact the same as the core number so that it becomes an extreme case as a symmetric architecture (Fig. 1(f)). The minimum value is $Ext = 1$ which represent the architecture with only one extended core as Fig. 2(b). As opposed to Eq. 6, Eq. 10 makes the assumption that the code accelerated by extensions can not be executed by simple cores even at lower speed. As the extension accelerated phases are executed only by extensions under this assumption, the extension execution conflicts problem is simplified.

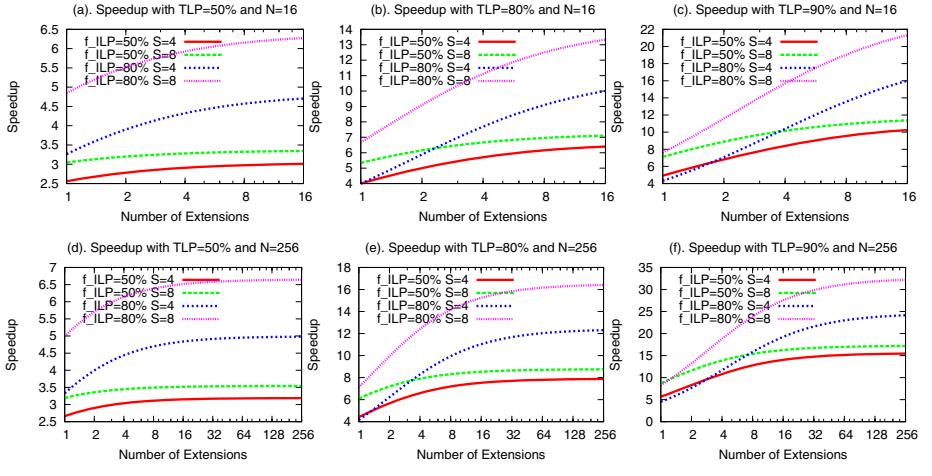

Fig. 9. Amdahl's law: with 16 and 256 cores, the number of extensions changes the system performance of the functionally asymmetric many-core architectures in the *conflict* case

In Fig. 9, we analyse the relationship between the number of extensions and the system performance. In this figure, we find that the low number of extensions decreases the system performance. The decrease rate depends on following parameters: N, S, f_{TLP} and f_{ILP}. With the same number of core N, the higher the thread level parallelism (f_{TLP}), the more extensions are needed to benefit from f_{TLP}. For the same reason, the higher the instruction level parallelism (f_{ILP}) and lower speedup rate (S), the more extensions are required. The core number N is also important. When N is small, adding extensions can always improve the system performance by accelerating the parallel phase. If there are many cores integrated in one system, the parallel phase is already well accelerated with these cores. Fig. 9(d)(e)(f) shows that a small percentage of extensions is enough to provide a high system performance.

Based on the analysis of the results from both the ideal and worst cases, we find that symmetric architectures are not the most efficient choice for the area and power

properties for many-core systems. The best solution would be the functionally asymmetric many-core architectures with just the necessary number of extensions to achieve a high occupation rate.

3.4 Constraints of the Theoretical Analysis

Our theoretical analysis clearly indicates that the functionally asymmetric many-core architecture has similar system performance but lower cost compared with the symmetric one. Some constraints and limitations of this analysis need to be listed:

- **Thread Migration Cost:** In functionally asymmetric many-core architectures, all threads/processes using extended instructions and coprocessors should be scheduled/migrated to the extended core with these extensions. This migration leads to some overhead which is not taken into account in our theoretical analysis. In the shared memory many-core architecture, this overhead mainly includes the cache-reload penalty which is much lighter than the overhead of *Non-Uniform Memory Access* **(NUMA)** and other distributed systems approaches [15]. For example, the migration framework detailed in [13] shows only a 5.3% slowdown for FFmpeg for this type of thread migration.
- **Thread Synchronization Cost:** As this theoretical analysis is based on Amdahl's law, applications are divided into sequential and parallel phases. To simplify the analysis, the cost of synchronizing the processes between those phases is ignored.
- **Common Case Analysis:** In our theoretical analysis, we focus only on the ideal and worst cases. In fact, the common case can also be theoretically studied using queueing theory. This is still to be done and is a natural future work of this paper.

4 Summary and Conclusion

This paper uses Amdahl's law to show that the occupation rate of cores decreases with the increasing number of cores because of the limited parallelism in application. Even worse, the occupation rate of extensions added to accelerate the computation decreases much faster than that of cores as only part of the application can be accelerated by extensions, even in the parallel execution parts. Our theoretical analysis shows that symmetric architectures in which each core has exactly the same extension waste chip area for both ideal and worst cases. To solve this issue, we advocate the use of functionally asymmetric many-core architectures which integrate much less extensions than cores. Our analysis show that they are well suited to provide performance gains at low cost for specific classes of applications, and are thus well suited as processing elements for embedded devices.

Acknowledgment

This work was supported in part by the European MEDEA+ SoftSoC project. The authors thank Tensilica for the university program license.

References

1. Knebel, P., Arnold, B., Bass, M., Kever, W., Lamb, J., Lee, R., Perez, P., Undy, S., Walker, W.: HP's PA7100LC: a low-cost superscalar PA-RISC processor. In: Compcon Spring 1993, pp. 441–447 (February 1993)
2. Goodacre, J., Sloss, A.: Parallelism and the ARM instruction set architecture. Computer 38(7), 42–50 (2005)
3. ITRS System Drivers technonical report 2007 (2007), http://www.itrs.net/Links/2007ITRS/Home2007.htm
4. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: AFIPS 1967 (Spring): Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, pp. 483–485. ACM, New York (1967)
5. Hill, M.D., Marty, M.R.: Amdahl's Law in the Multicore Era. IEEE Computer 41(7), 33–38 (2008)
6. Suleman, M.A., Mutlu, O., Qureshi, M.K., Patt, Y.N.: Accelerating critical section execution with asymmetric multi-core architectures. In: Soffa, M.L., Irwin, M.J. (eds.) ASPLOS, pp. 253–264. ACM, New York (2009)
7. Woo, D.H., Lee, H.H.S.: Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era. IEEE Computer 41(12), 24–31 (2008)
8. Gonzalez, R.E.: Xtensa: A Configurable and Extensible Processor. IEEE Micro 20(2), 60–70 (2000)
9. Altera Inc., NIOS II Microprocessor (2010), http://www.altera.com
10. ARM Inc., ARM NEON Technology (2010), http://www.arm.com/products/processors/technologies/neon.php
11. Tremblay, M., O'Connor, J.M., Narayanan, V., He, L.: VIS Speeds New Media Processing. IEEE Micro (16), 10–20 (1996)
12. Shen, H., Pétrot, F.: Novel task migration framework on configurable heterogeneous MP-SoC platforms. In: ASP-DAC 2009: Proceedings of the 2009 Asia and South Pacific Design Automation Conference, pp. 733–738 (2009)
13. Li, T., Brett, P., Knauerhase, R., Koufaty, D., Reddy, D., Hahn, S.: Operating system support for overlapping-ISA heterogeneous multi-core architectures. In: IEEE 16th International Symposium on High Performance Computer Architecture, HPCA 2010, pp. 1–12 (9-14, 2010)
14. Flamand, E.: Strategic Directions towards Multicore Application Specific Computing. In: Proceedings of the 2009 Design, Automation and Test in Europe Conference, Nice, France (April 2009); 1266 Keynote speach
15. Squillante, M.S., Nelson, R.D.: Analysis of task migration in shared-memory multiprocessor scheduling. In: SIGMETRICS 1991: Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 143–155. ACM, New York (1991)