Multicore Systems on Chip

Luigi Carro and Mateus Beck Rutzig

Abstract This chapter discusses multicore architectures for DSP applications. We explain briefly the main challenges involved in future processor designs, justifying the need for thread level parallelism exploration, since instruction-level parallelism is becoming increasingly difficult and unfeasible to explore given a limited power budget. We discuss, based on an analytical model, the tradeoffs on using multiprocessor architectures over high-end single-processor design regarding performance and energy. Hence, the analytical model is applied to a traditional DSP application, illustrating the need of both instruction and thread exploration on such application domain. Some successful MPSoC designs are presented and discussed, indicating the different trend of embedded and general-purpose processor market designs. Finally, we produce a thorough analysis on hardware and software open problems like interconnection mechanism and programming models.

1 Introduction

Industry competition in the current wide electronics market makes the design of a device increasingly complex. New marketing strategies have been focusing on increasing the product functionalities to attract consumer's interest. However, the convergence of different functions in a single device produces new design challenges, making the device implementation even more difficult. To worsen this scenario, designers should handle well known design constraints as energy consumption and process costs, all mixed in the difficult task to increase the

L. Carro (✉)
Federal University of Rio Grande do Sul, Bento Gonçalves 9500, Porto Alegre, Brazil
e-mail: carro@inf.ufrgs.br

M.B. Rutzig
Federal University of Santa Maria, Av. Roraima 1000, Santa Maria, Brazil
e-mail: mateus@inf.ufsm.br

S.S. Bhattacharyya et al. (eds.), *Handbook of Signal Processing Systems*,
DOI 10.1007/978-1-4614-6859-2_17,

processing capability, since users desire nowadays the equivalent of a portable supercomputer. Therefore, the fine tuning of these requirements is fundamental to produce the best possible product, which consequently will obtain a wider market.

The convergence of functionalities to a single product enlarges the application software layer over the hardware platform, increasing the range of heterogeneous code that the processing element should handle. The clear examples of such convergence with mixed behavior are iPhone and Android phones. Aiming to balance manufacturing costs and to avoid overloading of the original hardware platform with extra processing capabilities, way beyond the original hardware design scope, there is a natural lifecycle that guides the functionality execution scheme in the platform. Here, the functionality lifecycle is divided into three phases: introduction, growth and maturity. Commonly, during the introduction phase, which reflects the time when a novel functionality is launched in the market, due to doubts about the consumer acceptance, the logical behavior of the product is described using well known high level software languages like C++, Java and .NET. This execution strategy avoids costs, since the target processing element, usually a general-purpose processor (GPP), is the very same of (or very close to) the previous product. After market consolidation the growth and maturity phase start. At this time, the functionality is used in a wide range of products, and its execution tends to be closer to the hardware to achieve better energy efficiency, speedup, and to avoid overloading in the general purpose processor.

Generally, two methods are used to approach the required functionality of a product to the underlying hardware, aiming to explore the benefits of a joint development. The first technique evaluates small applications parts, which possibly have huge impact in the whole application execution. For example, this used to be the scenario of embedded systems some time ago, or of some very specialized applications in general-purpose processors. After the profiling and evaluation phase, chosen code parts are moved to specialized hardwired instructions that will extend the processor instruction set architecture (ISA) and assist a delimited range of applications. MMX, SSE and 3DNow! are successful ISA extensions that have been created aiming to support certain application domains, in those cases, multimedia processing.

A second technique uses a more radical approach to close the gap between the hardware and the required functionality. Its entire logic behavior is implemented in hardware, aiming to build an application specific integrated circuit (ASIC). ASIC development can be considered a better design solution than ISA extensions, since it provides better energy efficiency and performance.

Due to several reasons the current scenario of the electronics market is changing, since over 1.5 billion of embedded devices were shipped in 2011 [20] showing an increasing by 11% in comparison with 2010. Besides the already explained drawbacks regarding the traditional product manufacturing, such designs need to worry about battery life, size and, for critical applications, reliability. Aiming to achieve their hard requirements, system-on-a-chip (SoC) is largely used in the embedded domain. The main proposal of a SoC design is to integrate in a single die the processing element (in most cases a GPP is employed), memory,

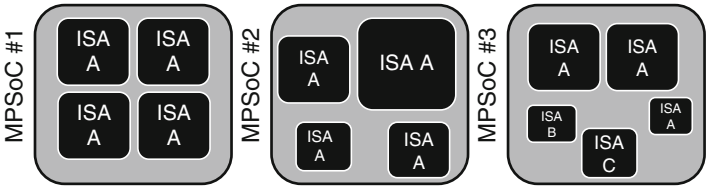

Fig. 1 Different MPSoC approaches

communication, peripheral and external interfaces. This eases the deployment of a product, by freeing designers from the hard system validation task. The mobileGT SoC platform [6] exemplifies a successful employment of a SoC for the embedded domain. BMW, Ford, General Motors, Hyundai and Mercedes Benz use the cited SoC in their cars to manage the information and entertainment applications.

However, SoC designs do not sustain the required performance for the growing market of heterogeneous software behaviors, which has been running on top of the available GPP processor. Moreover, the instruction level parallelism (ILP) exploration is no longer an efficient technique, in terms of energy consumption, to improve performance of those processors, due the limited ILP available in the application code [22]. Despite the great advantages shown on ISA extensions employment, like the MMX strategy, this approach relies on a high design and validation time, which goes against the need of a fast time-to-market of embedded systems. ASIC employment usually depends on a complex and costly design process that could also affect the time-to-market required by the company. In addition, this architecture attacks only a specific application domain, and can fail to deliver the required performance to software behaviors that are out of this domain. However, both ASIC and ISA extension are supplied by their high-performance under a limited application domain.

This critical scenario dictates the need for changes on the hardware platform development paradigm. A new organization that appears as a possible solution for the current design requirements is the Multiprocessor System-on-chip (MPSoC). Many advantages can be obtained by combining different processing elements into a single die. The computation time can clearly benefit since, at the same time, several different programs could be executed in the available processing elements. In addition, the flexibility to combine different processing elements appears as a solution to the heterogeneous software execution problem, since designers can select the set of processing elements that best fit in their design requirements. Figure 1 illustrates three examples of MPSoC with different set of processing elements. MPSoC #1 is composed of only general-purpose processors (GPP) with the same processing capability. In contrast, the MPSoC #2 aggregates GPP with distinct computation capability. As another example, MPSoC #3 is assembled to another computation purpose, since it contains three different processing paradigms. More details about each MPSoC approach are explained in the next sections.

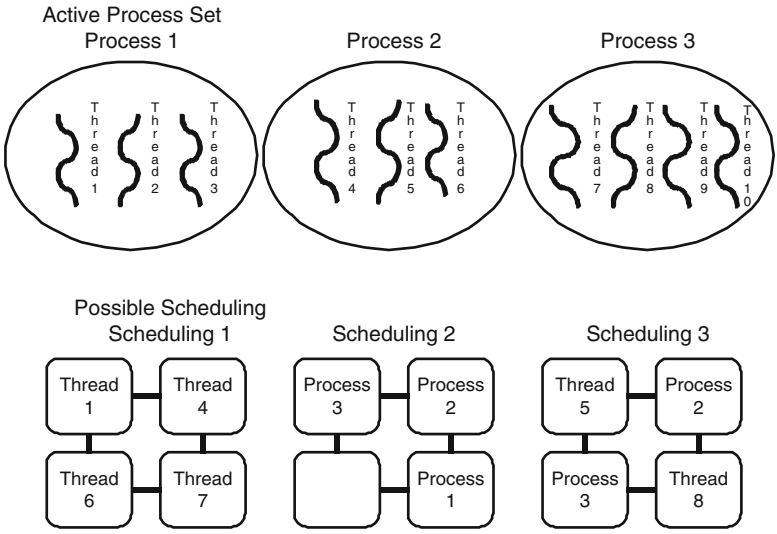

Fig. 2 Scheduling methodologies for a given active process set

MPSoC provides several advantages, and three highlight among all: performance, energy consumption and validation time. MPSoC design introduces a new parallel execution paradigm aiming to overcome the performance barrier created for the instruction level parallelism limitation. In contrast to the ILP exploration paradigm that can be supported both for hardware or software, current coarser parallelism detection strategies are made only for the software team. The hardware team is only responsible for encapsulating and making the communication infrastructure to build a single die with a suitable number of processing elements. The software team splits and distributes the paralyzed code among the MPSoC processing elements. Although both teams might work together, the reality is that there is no strong methodology nowadays that can support software development for multiprocessing chips for any kind of application.

We define parallel code as composed of threads or processes. A thread is contained inside a process, and differs from this by its lightweight context information. Moreover, a process has a private address space, while a set of threads shares the same address space used to make communication with each other in an easier fashion. Figure 2 illustrates an active process set environment composed of three threaded processes. Scheduling 1 reflects the traditional Thread-Level Parallelism (TLP), exploiting the parallel execution of active processes parts. Scheduling 2 methodology supports only single-threaded parallelism exploitation, in Process-Level Parallelism (PLP) only processes are scheduled, regardless of their internal thread number. Finally, there is mixed parallelism exploitation, illustrated by Scheduling 3, working both in thread and process scheduling levels. The last approach is widely used in the commercial MPSoC due to its greater flexibility on scheduling methodology. The scheduling policies rely on the MPSoC processing

elements set (Fig. 1). Therefore, the above scheduling approaches can be associated with the previous MPSoC processing element sets. Supposing that all GPP of MPSoC #1 and MPSoC #2 have the same ISA, Scheduling 1 methodology only fits in this hardware modeling due the binary compatibility of threads code. In contrast, the three examples of MPSoC illustrated in Fig. 1 support the Scheduling 2 and 3.

Commonly, parallel execution in multiprocessing platforms is not only used to achieve better speedup, but energy consumption is a big reason for its usage as well. Several factors, both in hardware and software level, contribute to the lower energy consumption of MPSoC architectures. Considering the architecture design, there is no dedicated hardware to split and distribute the application code over the processing elements, since the software team does such work during application development time. A responsible for software distribution is needed, and this is the main problem.

Embedded platforms employ several techniques to save energy in multiprocessing designs. Dynamic voltage and frequency scaling (DVFS) is applied at the entire chip, changing, at runtime, the global MPSoC voltage and frequency [26]. The management policy is based on some system tradeoff like processing elements load, battery charge or chip temperature. Commonly, DVFS is controlled by software, which provides unsuitable time overhead to change the processing element status. Nevertheless, DVFS can affect the performance and even cause drawbacks on real-time based systems. DVFS can also be applied independently on each processing element. Software team during profiling can insert frequency scaling instructions into code, to avoid hardware monitoring for DVFS. Hence, the computational load of each processor is individually monitored, making chip power management more accurate [2, 18]. More recently, aiming to decrease the delay of power management based on software, Core i7, a multiprocessing platform from Intel, employs a dedicated built-in microcontroller to handle DVFS [9, 31]. The built-in chip power manager frequently looks at the temperature and power consumption aiming to turn off independent cores when they are not being used. The DVFS microcontroller can be seen as an ASIC that supplies only the power management routine, which makes clear the convergence of dedicated processing elements to a single die.

Consumers are always waiting for products equipped with exciting features. Time-to-market appears as an important consumer electronics constraint that should be carefully handled. Researches explain that 70% of the design time is spent in the platform validation [1] being an attractive point for optimization. Regarding this subject, MPSoC employment softens the hard task to shrink time-to-market. Commonly, an MPSoC is built by the combination of validated processing elements that are aggregated into a single die as a puzzle game. Since each puzzle block reflects a validated processing element, the remaining design challenge is to assemble the different blocks. Actually, the designer should only select a communication mechanism to connect the entire system, easing the design process by the use of standard communication mechanisms.

Up to now, only explanations about hardware characteristics of MPSoC were mentioned. However, software partitioning is a key feature in an MPSoC system. Considering software behavior, a computational powerful MPSoC becomes useless

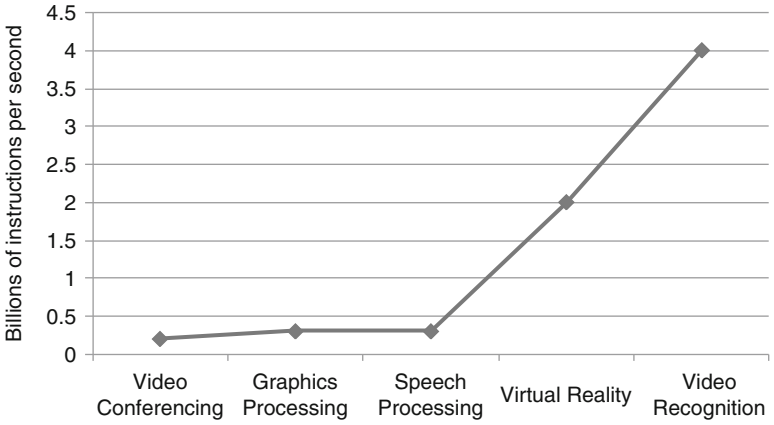

Fig. 3 Throughput requirements of traditional DSP applications, adapted from [7]

if the application partitioning is poorly performed, or if application does not provide a minimum thread or process level parallelism to be explored by the computational resources. Amdahl's law argues that an application speedup is limited by its sequential part. Therefore, if an application needs 1 h to execute, being 5 min sequential (almost 9% of entire application code), the maximum speedup provided for a multiprocessing system is 12 times, no matter how many processing elements are available. Therefore, the designer ability to perform software partitioning and the limited parallel code are the main challenges on an MPSoC design development.

Today, we are usually in touch with functionalities like video conferencing, graphics and speech processing. These functionalities are widespread in digital cameras, MP3, DVD, cell phones and generic communication devices. To support their executions several DSP algorithms are running inside of these devices. Fast Fourier Transform (FFT), Finite Impulse Response (FIR), Discrete Cosine Transform (DCT) and IEEE 802.11 communication standard are examples of these algorithms. The growing convergence of different functionalities to a single device has been increasing, and it is difficult for a single hardware platform to reach the performance requirements of these applications.

Many DSP functionalities and their throughput requirements are shown in Fig. 3. Let us suppose a mobile device composed by all the functionalities shown in Fig. 3, and also let us suppose that they are running on a 1 GHz 8-issue superscalar processor. The processor throughput needed to support the execution is almost seven billion instructions per second. The application execution scenario becomes hard for a general-purpose processor, since the maximum performance of the baseline processor reaches eight billion of instructions per second (with perfect ILP exploitation and perfect branch prediction). In this way, DSP processors are always present in real life products, aiming to achieve, with their specialized DSP instructions, the necessary performance for the DSP applications. However, even DSP processors are suffering to reach the high performance required when many

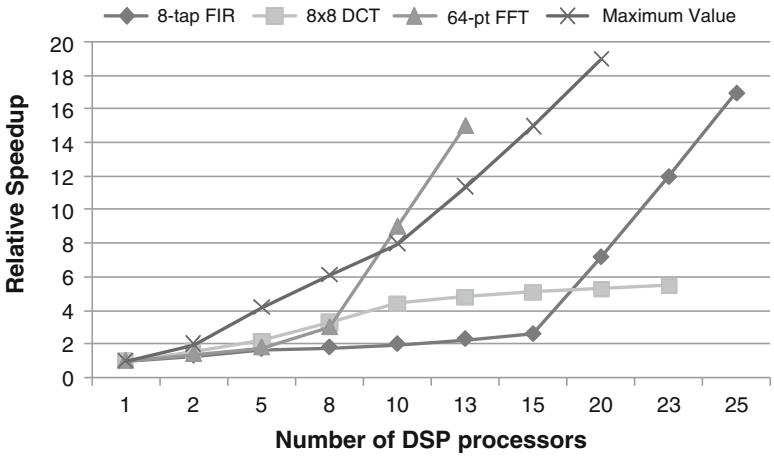

Fig. 4 Speedup of traditional DSP application on MPSoC system [8]

applications are deployed. Hence, the multiprocessing system appears, also in the DSP domain, as a solution to supply the high performance requirements. The major advantage of multiprocessing elements on the DSP domain is the inherent Thread-Level Parallelism, not present in most traditional applications [15].

Figure 4 illustrates the speedup provided by a multiprocessing system for DSP applications. The speedup for the Maximum Value application increases almost linearly with the number of processor. The eight-tap FIR and 64-pt FFT also show optimistic speedups when the number of processing elements increases. On the other hand, the 8×8 Discrete Cosine Transform provides the smallest speedup among the application workload. Clearly, this application is a good example for Amdahl's law, demonstrating that multiprocessing systems can fail to accelerate applications that have a meaningful sequential part.

The rest of this chapter is divided into five major sections. First, a more detailed analytical modeling is provided to show the actual design space of multiprocessing devices in the DSP field. Next, an MPSoC hardware taxonomy is presented, aiming to explain the different architectures and applicability domain. MPSoC systems are classified taking into account two points of view: architecture and employed organization. A dedicated section demonstrates some successful MPSoC designs. A section discusses open problems in MPSoC design, covering from software development, communication infrastructure. Finally, future challenges like how can one overcome Amdahl's law limitations are covered.

2 Analytical Model

In this sub-section, we try to figure out the potential of single parallelism exploitation by modeling a multiprocessing architecture (*MP-MultiProcessor*) composed of

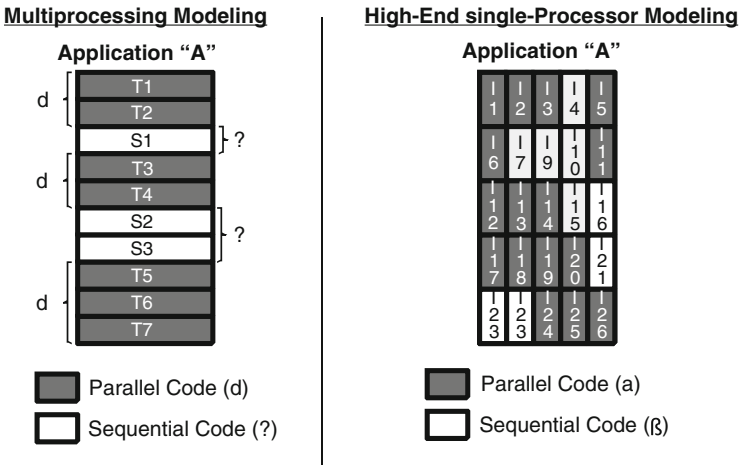

Fig. 5 Modeling of the (**a**) multiprocessor system and the (**b**) high-end single-processor

many simple and homogeneous cores without any capability to explore instruction level parallelism (ILP) to elucidate the advantages of Thread Level Parallelism (TLP) exploitation. We also compare its Execution Time (ET) to a high-end single processor *SHE* (*Single High-End*) model, which is able to exploit only ILP available in applications.

We have considered different amounts of fine- (instruction) and coarse- (thread) level parallelism available in the application code to investigate the performance potentials of the both aforementioned architectures.

Considering a part of a given application code, we classify it in four different ways:

- α – the instructions that can be executed in parallel in a single processor;
- β – the instructions that cannot be executed in parallel in a single processor;
- δ – the amount of instructions that can be distributed among the processors of the multiprocessor environment.
- γ – the amount of instructions that cannot be split, and therefore must be executed in one of the processors among those in the multiprocessor environment.

Figure 5 exemplify how the previously stated classification, considering a certain application "A", would be applied. In the example shown in this figure, when the application is executed in the multiprocessor system (Fig. 5a), 70% of the application code can be parallelized at some degree (i.e. divided in threads) and executed on different cores at the same time, so $\delta = 0.7$ and $\gamma = 0.3$. On the other hand, when the very same application "A" is executed on the high-end single-processor (Fig. 5b), in 64% of the application code instructions can be executed in parallel at some degree, so $\alpha = 0.64$ and $\beta = 0.36$.

2.1 Performance Comparison

Let us start with the basic equation relating Execution Time (ET) with instructions,

$$ET = Instructions * CPI * CycleTime, \tag{1}$$

where *CPI* is the mean number of cycles necessary to execute an instruction, and *Cycletime* is the clock period of the processor.

In this model, information about cache accesses, performance of the disk or any other I/O operation is not considered. However, although simple, this model can provide interesting performance clues on the potential of multiprocessing architectures and aggressive instruction level parallelism exploitation for a wide range of different applications classes.

2.1.1 Low End Single Processor

Based on Eq. (1), for a Low-End Single *SLE* (*Single Low End*) processor, the execution time can be written as:

$$ET_{SLE} = Instructions\,(\propto CPI_{SLE} + \beta CPI_{SLE})\,CycleTime_{SLE}. \tag{2}$$

Since the low-end processor is a single-issue processor, it is not able to exploit ILP. Therefore, classifying instructions in α and β as previously stated does not make much sense. In this case, α is always equal to zero and β equal to one, but we will keep the notation and their meaning for comparison purposes.

2.1.2 High End Single Processor

In the case of a high-end ILP exploitation architecture, based on Eqs. (1) and (2), one can state that ET_{SHE}*(Execution Time of the High End Single Processor)* is given by the following equation:

$$ET_{SHE} = Instructions\,(\propto CPI_{SHE} + \beta\ CPI_{SLE})\,CycleTime_{SHE}. \tag{3}$$

As already explained, coefficients α and β refer to the percentage of instructions that can be executed in parallel or not (this way, $\alpha + \beta = 1$), respectively. *CycleTime*$_{SHE}$ represents the clock cycle time of the high-end single processor.

The CPI_{SHE} is usually smaller than 1, because a single high-end processor can exploit high levels of ILP, thanks to the replication of functional units, branch prediction, speculative execution, mechanisms to handle false data dependencies and so on. A typical value of CPI_{SHE} for a current high-end single processor is 0.62

[8], which shows that more than one instruction can be issued and executed per cycle. The CPI_{SHE}, could also be written as $\frac{\propto CPI_{SLE}}{\text{issue}}$, where *Issue* is the number of instructions that can be issued in parallel to the functional units, when considering the average situation (i.e. a High-End Single processor would have the same CPI as the CPI of a Low-End Processor divided by the mean number of instructions issued per cycle). Thus, based on Eq. (3), one gets:

$$ET_{SHE} = Instructions \left(\frac{\propto CPI_{SLE}}{issue} + \beta\ CPI_{SLE} \right) CycleTime_{SHE}. \tag{4}$$

Having stated the equation to calculate the performance of both high-end and low-end single processor models, now the potential of using a homogeneous multiprocessing architecture to exploit TLP is studied. Because we are considering that such architecture is built by the replication of low-end processors (so that a large number of them can be integrated within the same die), a single low-end processor does not have any capability to exploit the available ILP of each thread.

If one considers that each application has a certain number of sequences of instructions that can be split (transformed to threads) to be executed on several processors, one could write the following equation, based on Eqs. (1) and (2):

$$ET_{MP} = Instructions \left(\frac{\delta}{P} + \gamma \right) (\alpha CPI_{SLE} + \beta CPI_{SLE})\ CycleTime_{MP}, \tag{5}$$

where δ is the amount of sequential code that can be parallelized (i.e. transformed into multithreaded code), while γ is the part of the code that must be executed sequentially (so no TLP is exploited). *P* is the number of low-end processors that is available in the chip. As can be observed in the second term of the Eq. (5), because the single low end processor is considered, the multiprocessor architecture does not exploit ILP ($\alpha = 0$ and $\beta = 1$). Therefore, when one increases the number of processors *P*, only the part of code that presents TLP (δ) will benefit from the extra processors.

2.2 High-End Single Processor Versus Homogeneous Multiprocessor Chip

Based on the above reasoning, now we compare the performance of the high-end single processor to the multiprocessor architecture. Since power is crucial in an embedded system design, we have chosen a certain total power budget as a fair performance factor to compare both designs. Thus, based on Eqs. (3) and (5), one can consider the following equation:

$$\frac{ET_{SHE}}{ET_{MP}} = \frac{\left[Instructions\left(\propto \frac{CPI_{SLE}}{issue} + \beta CPI_{SLE}\right) CycleTime_{SHE}\right]}{\left[Instructions\left(\frac{\delta}{P} + \gamma\right)(\propto CPI_{SLE} + \beta CPI_{SLE}) CycleTime_{MP}\right]}. \quad (6)$$

If one considers that in the model of the multiprocessor environment, a single low end processor is not capable of exploiting instruction level parallelism, and then $\propto = 0$, one can reduce the Eq. (6) to:

$$\frac{ET_{SHE}}{ET_{MP}} = \frac{\left[Instructions\left(\propto \frac{CPI_{SLE}}{issue} + \beta CPI_{SLE}\right) CycleTime_{SHE}\right]}{\left[Instructions\left(\frac{\delta}{P} + \gamma\right)(0 * CPI_{SLE} + 1 * CPI_{SLE}) CycleTime_{MP}\right]}, \quad (7)$$

and, by simplifying (7), one gets

$$\frac{ET_{SHE}}{ET_{MP}} = \frac{\left[Instructions\left(\propto \frac{CPI_{SLE}}{issue} + \beta CPI_{SLE}\right) CycleTime_{SHE}\right]}{\left[Instructions\left(\frac{\delta}{P} + \gamma\right)(CPI_{SLE}) CycleTime_{MP}\right]}. \quad (8)$$

We are also considering that, as a homogeneous multiprocessor design is composed of several low-end processors with a very simple organization, those processors could run at much higher frequencies than a single and complex high-end processor. Therefore, we will assume that

$$\left(\frac{1}{CycleTime_{MP}}\right) = K * \left(\frac{1}{CycleTime_{SHE}}\right), \quad (9)$$

where K is the frequency adjustment factor to equal the power consumption of the homogeneous multiprocessor with the high-end single processor.

By merging and simplifying Eqs. (8) and (9), one gets:

$$\frac{ET_{SHE}}{ET_{MP}} = \left[\frac{1}{\frac{\delta}{P} + \gamma}\right]\left[\frac{\propto \frac{CPI_{SLE}}{issue} + \beta CPI_{SLE}}{CPI_{SLE}}\right] K. \quad (10)$$

According to Eq. (10), a machine based on a high-end single core will be faster than a multiprocessor-based machine if $\left(\frac{ET_{SHE}}{ET_{MP}}\right) < 1$. This equation also shows that, although the multiprocessor architecture with low-end simple processors could have a faster cycle time (by a factor of K), that factor alone is not enough to define performance, as demonstrated in the second term between brackets in Eq. (10). Because the high-end processor can execute many instructions in parallel, better performance improvements can be obtained, as long as ILP is the dominant factor, instead of TLP.

To better illustrate this point, let us imagine the extreme case: $P = \infty$, meaning that infinite processors are available. In addition, if one considers that the multipro-

cessor design is composed of single low end processors that do not exploit ILP and, therefore, $\propto CPI_{SLE}$ is always equal to zero, it can be removed from the equation. Therefore, Eq. (10) reduces to:

$$\frac{ET_{SHE}}{ET_{MP}} = \left[\frac{\propto \frac{CPI_{SLE}}{issue} + \beta CPI_{SLE}}{\gamma CPI_{SLE}} \right] K. \tag{11}$$

Let us consider that the execution of the very same application on both multiprocessor and single high-end architectures presents exactly the same amount of sequential code, so $\beta = \gamma$. In this case, the operating frequency (given by the K factor) will determine which architecture runs faster if the issue width of the high-end superscalar processor also tends to infinite.

In another example, if one applies the same Eq. (11) in a scenario where an application presents 10% of sequential code ($\beta = \gamma = 0.1$) and is executing on a four issue high-end single processor, the operating frequency of the four issue high-end single processor should be only 20% ($K = 0.8$) greater than the multiprocessor to achieve the very same execution time. On the other hand, if that the application now presents 90% of sequential code ($\beta = \gamma = 0.9$), the high-end single processor should run 3.2 times ($K = 0.31$) faster than the multiprocessor design. With these corner cases, one can conclude that when applications present small parts of its code that cannot be parallelized, both architectures running at the same operating frequency will present almost the same performance regardless the number of processors in a multiprocessor system. For applications with huge amount of sequential code, complex single processors must run at higher frequencies than multiprocessors systems to achieve the same performance.

2.3 Applying the Analytical Model in Real Processors

Given the analytical model, one can briefly experiment it with numbers based on real data. Let us consider a high-end single core: a 4-issue SPARC64 superscalar processor with CPI equal to 0.62 [8]; and a multiprocessor design composed of low-end single-issue TurboSPARC processors with CPI equal to 1.3 [8]. A comparison between both architectures is done using the equations of the aforementioned analytical model. In addition, we consider that the TurboSPARC has 5,200,000 transistors [16], and that the SPARC64 V design [24] requires 180,000,000 transistors to be implemented. For the multiprocessing design we add 37% of area overhead due to the intercommunication mechanism [25]. Therefore, aiming to make a fair performance comparison among the high-end single core and the multiprocessor system, we have devised an 18-Core design composed of low-end processors that has the same area of the 4-issue superscalar processor and consumes the same amount of power.

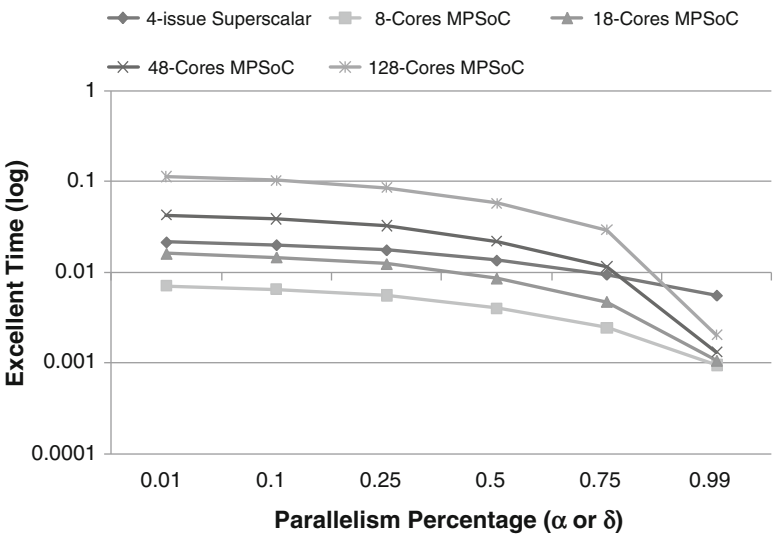

Fig. 6 Multiprocessor system and Superscalar performance regarding a power budget using different ILP and TLP; $\alpha = \delta$ is assumed

Figure 6 shows, in a logarithmic scale, the performance of the superscalar processor, when parameters α and β change, and the performance of the many in-order TurboSPARC cores, when the δ and γ and the number of processors (from 8 to 128) varies. The x-axis of Fig. 6 represents the amount of the instruction- and thread-level parallelism in the application, considering that the α factor is only valid for the superscalar processor, while δ is valid for all the multiprocessing systems' setups.

The goal of this comparison is to demonstrate which technique better explores its particular parallelism type at different levels, considering six values for both ILP and TLP. For instance, $\delta = 0.01$ means that a hypothetic application only shows 1% of TLP available within its code (in the case of the multiprocessing systems). In the same way, when $\alpha = 0.01$, it is assumed that only 1% of the total number of instructions can be executed in parallel on the superscalar processor. In these experiments, we considered the same power budget for the high-end single core and the multiprocessor approaches. In order to normalize the power budget of both approaches we have tuned the adjustment factor K of Eq. (9). For that, we fixed the power consumption of the 4-issue superscalar to use it as the reference, changing the operating frequency (K factor) of the remaining approaches to achieve the same power consumption.

Thus, the operating frequency of the 8-Core multiprocessing system must be 3 times higher than the one of the four-issue superscalar processor. For the 18-Core setup, the operating frequency must be a 25% higher than the reference value. Since a considerable number of cores is employed in the 48-Core setup, it must execute 2 times slower than the superscalar processor to operate under the same power budget.

Finally, the operating frequency of the 128-Core design must be 5.3 times lower than the superscalar.

In the leftmost side of Fig. 6, one considers any application that has a minimum amount of instruction ($\alpha = 0.01$) and thread ($\delta = 0.01$) level parallelism. In this case, the superscalar processor is slower than the 8- and 18-Core designs since the parallelism is insignificant, the higher operating frequency of both multiprocessing system is responsible for faster execution. Moreover, when the application shows higher parallelism levels ($\alpha > 0.25$ and $\delta > 0.25$), the 18- and 8-Core better handles the extra TLP available than the superscalar does with the ILP, presenting more performance. So, considering only the 18-Core design, the multiprocessing system achieve better performance with the same area and power budget in the whole spectrum of parallelism available.

However, as more cores are added in a multiprocessor design, the overall clock frequency tends to decrease, since the adjustment factor K must be decreased to respect the power budget. Therefore, the performance of applications that present low TLP (small δ) worsens when the number of cores increases. Applications with $\delta = 0.01$ in Fig. 6 are good examples of this case: performance is significantly affected as the number of cores increases. As another representative example, even when almost the whole application presents high TLP ($\delta > 0.99$), the 128-Core design takes longer than the other multiprocessor designs. Figure 6 concludes that the increasing on the number of cores not always produces a satisfactory tradeoff among energy, performance and area.

2.4 Energy Comparison

If the superscalar processors usually does not perform well in most cases, when one measures energy consumption, several factors combine to further affect its employment on embedded systems: power in CMOS circuits is proportional to the switching capacitance, to the operating frequency and to the square of the power supply. In this simple model, we will assume that the power is dissipated only in the data path. This is clearly overly optimistic for what regards the power dissipated by a superscalar, but this can also give an idea of the lower bound of energy dissipation in the high-end single processor.

The power dissipated by a high-end single processor can be written as

$$P_{SHE} \approx issue * C * \left(\frac{1}{CPI_{SHE}}\right) * V_{SHE}^2, \tag{12}$$

where C is the capacitance switching of a single issue processor, and V_{ss} is the voltage the processor is operating on. The term $\left(\frac{1}{CPI_{SHE}}\right)$ is included to consider the extra power needed during the speculation process to sustain performance with a CPI smaller than 1. The energy of the high-end single processor is given by:

$$E_{SHE} = P_{SHE} * T_{SHE}. \tag{13}$$

Power consumed by a homogeneous multiprocessing system is given by

$$P_{MP} \approx P * C * \left(\frac{1}{CPI_{SLE}}\right) * V_{MP}^2. \tag{14}$$

Again, as in the case of superscalar processor, the term considering the CPI of the single low-end processor $\left(\frac{1}{CPI_{SLE}}\right)$ has been also included, while its energy is given by

$$E_{MP} = P_{MP} * T_{MP}. \tag{15}$$

When one compares both, it is possible to write

$$\begin{aligned} \frac{E_{SHE}}{E_{MP}} &= issue * \left[\left(\propto \frac{CPI_{SLE}}{issue} + \beta CPI_{SLE}\right) * \frac{1}{CPI_{SHE}} * \frac{1}{K}\right] \\ &= P * \left[\left(\frac{\delta}{P} + \gamma\right) * (\alpha CPI_{SLE} + \beta CPI_{SLE})\right] * \frac{1}{CPI_{MP}}, \end{aligned} \tag{16}$$

and, by simplifying (16), one gets

$$\frac{E_{SHE}}{E_{MP}} = \frac{V_{SHE}^2 \left[(\propto + issue * \beta) * \frac{1}{K}\right]}{V_{MP}^2 [(\delta + P\gamma)]}. \tag{17}$$

Equation (17) demonstrates that both approaches unnecessarily spend power when there is no ILP or TLP available since there is no power management technique modeled to reduce power supply (V_{SHE}^2 and V_{MP}^2). Figure 7 shows the energy results considering the same power budget, as it was already done in the performance model. For this first experiment, we do not consider the communication overhead for the multiprocessing environment that will be modeled later. In addition, we only show the energy of 8- and 18-Core Designs, since the conclusions of these setups are also valid for the rest of the setups.

The high-end single processor organization spends higher energy than the 18-Core multiprocessor the same amount of energy when considering all levels of available parallelism since the latter is faster than the former in all cases (Fig. 6).

To obey the given power budget, the 8-Core multiprocessor runs 3 times faster than four-issue superscalar and the 18-Core multiprocessor. Thus, as the 8-Core Design present 3 times lower execution time than the 4-issue superscalar, the former spends 3 times less energy. When the parallelism is more exposed the superscalar approaches to the 8-Core Design, since its execution time decreases. Multiprocessors composed of a significant number of cores present worst performance in applications with low/medium TLP (Fig. 6). Consequently, in those cases and if no power management techniques are considered (e.g., cores are turned off when not used), energy consumption of such multiprocessor designs tend to be higher than those with fewer cores. As can be seen in Fig. 7, the 8-Core multiprocessor

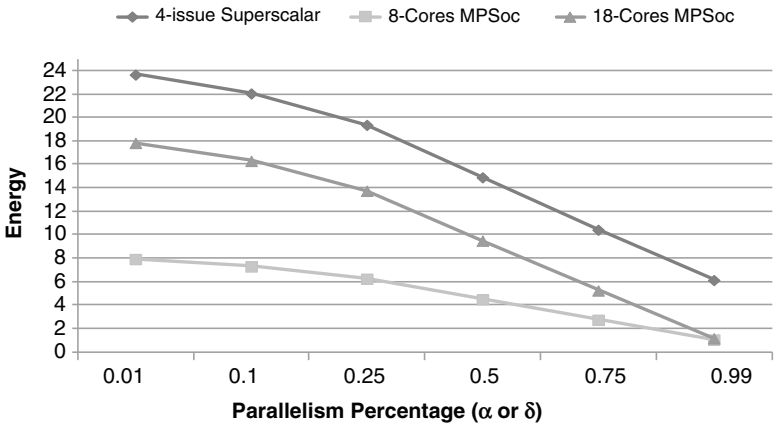

Fig. 7 Multiprocessing Systems and High-end single processor energy consumption; $\alpha = \delta$ is assumed

consumes less energy than the 18-Core for low/medium TLP values ($\delta < 0.75$). However, when applications present greater thread level parallelism ($\delta > 0.9$), the energy consumed by the 18-Core multiprocessor reaches the same values as the 8-Core design, thanks to the better usage of the available processors.

2.5 DSP Applications on a MPSoC Organization

We evaluate the superscalar and MPSoC performance considering an actual DSP application execution. An 18-tap FIR filter is used to measure the performance of both approaches handling a traditional DSP application. The C-like description of the FIR filter employed in this experiment is illustrated in Fig. 8. Superscalar machines explore the FIR filter instruction level parallelism in a transparent way, working on the original binary code. Unlike the superscalar approach, to explore the potential of the MPSoC architecture there is a need to make manual source code annotations in order to split the application code among many processing elements. In this way, some code highlights are shown in Fig. 8 to simulate annotations, indicating the necessary number of cores to explore the ideal thread level parallelism of each part of the FIR filter code. For instance, the first annotation considers a loop controlled for IMP_SIZE value, which depends on the number of FIR taps. In this case, 54 loop iterations are done since the experiment regards an 18-tap FIR filter.

The OpenMP [4] programming language provides specific code directives to easily split loop iterations among processing elements. Using OpenMP directives, the ideal exploration of this loop is done through 54-core MPSoC, each one being responsible for running single loop iteration. However, when the amount of processing elements is lower than the number of loop iterations, OpenMP combines

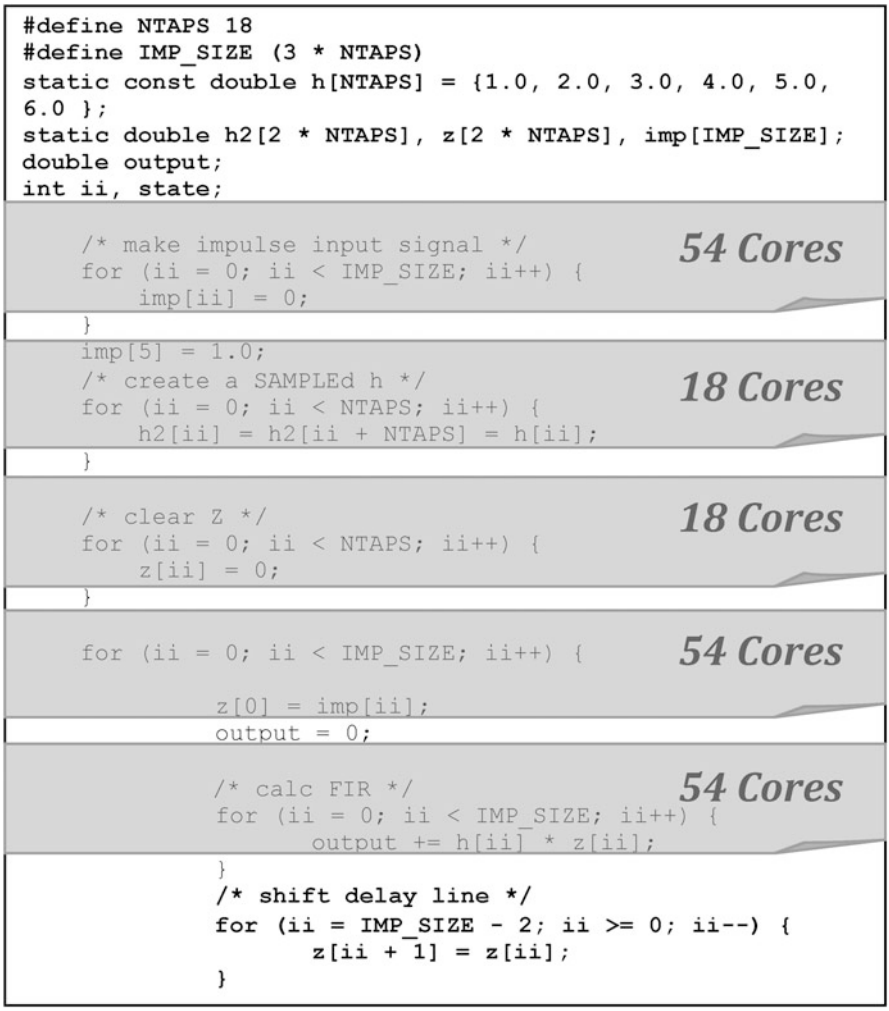

Fig. 8 C-like FIR filter

them in groups to distribute tasks among the available resources. Hence, regarding the execution of 54 loop iterations into 18-core MPSoC, OpenMP creates 18 groups, each one composed of three iterations. Figure 8 demonstrates that almost the entire FIR code can be parallelized, since its code is made up of several loops. In general, traditional DSP applications (ex: FFT and DCT) have a loop-based code behavior suitable for OpenMP loop parallelization. However, some loops cannot be parallelized due data dependency among iterations. For instance, the last loop of the FIR filter description shown in Fig. 8 demonstrates this behavior, since shifting array values presents dependencies among all loop iterations.

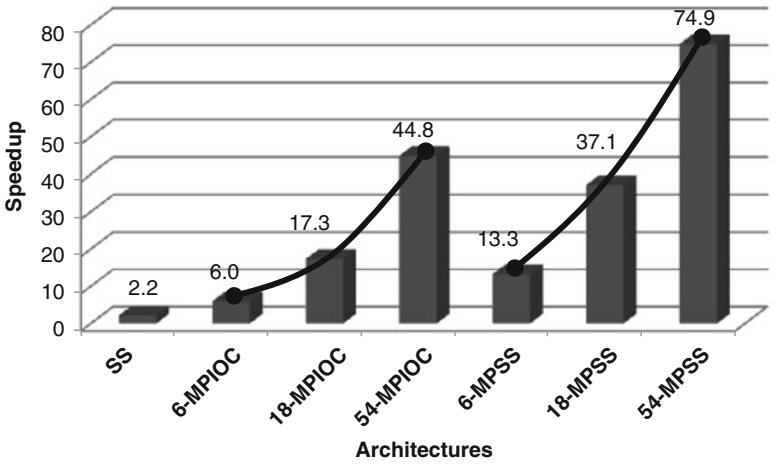

Fig. 9 Speedup provided in 18-tap FIR filter execution for superscalar, MPSoC and a mix of both approaches

Aiming to illustrate the impact on performance of TLP and ILP exploration on DSP applications, we evaluated the 18-tap FIR execution over three different architectures: a four-issue Superscalar (SS); 6- 18- and 54-core MPSoCs based on pipelined cores, with no ILP exploration capabilities (MP_{IOC}). Finally, in order to have a glimpse on the future, we imagined a 6- 18- and 54-Cores MPSoCs based on a four-issue superscalar processor, able to explore both ILP and TLP (MP_{SS}). We have extracted the speedup with a tool [16] that makes all data dependence graphs of the application. After, considering the characteristics of the evaluated architectures, the execution time of each graph is measured in order to obtain their speedup over the baseline processor. It is important to point out that instruction and thread communication overhead has not been taken into account in this experiment.

The results shown in Fig. 9 reflect the speedup provided over a single pipelined core performance running the C-like description of the 18-tap FIR filter presented in Fig. 8. The leftmost bar shows the speedup provided for the ILP exploration of a four-issue superscalar processor. In this case, the execution time of the Superscalar processor is 2.2 times lower than that of a pipelined core, showing that the FIR filter has neither high nor low ILP, since a four-issue superscalar processor could potentially achieve up to 4 times the performance of a pipelined core.

Considering the MPSoC composed of pipelined cores, the 6-core machine provides almost a linear speedup, decreasing by 5.96 times the single pipelined core execution time. This behavior is maintained when more pipelined cores are inserted. However, when 18-tap FIR filter is explored for the maximum TLP (54-MP_{IOC}), a speed up of only 44.8 times is achieved, showing that even applications which are potentially suitable for TLP exploration could present non-linear speedups. This can be explained by the sequential code present inside of each loop iteration.

Amdahl's Law shows that it is not sufficient to build architectures with a large number of processors, since most parallel applications contain a certain amount of sequential code [23]. Hence, there is a need to balance the number of processors with a suitable ILP exploration approach to achieve greater performance. The MP_{SS} approach combines TLP with ILP exploration of four-issue superscalar aiming to show that simple TLP extraction is not enough to achieve linear speedups even for DSP applications with high TLP. Figure 9 illustrates the speedup of the MP_{SS} approach. As it can be noticed, 6-MP_{SS} accelerates the 18-tap FIR filter more than twice 6-MP_{IOC}, since higher ILP is explored for the superscalar processor. In this case, when the first loop of Fig. 8 is split among the MPSoC, each core executes nine loop iterations providing a large room for ILP exploration. However, when the number of cores increases, the sequential code decreases, making less room for the ILP optimization. Nevertheless, the 18-tap FIR filter execution in 54-MP_{SS} is 66% faster than 54-MP_{IOC} execution showing the need for a mixed parallelism exploration.

Summarizing, most DSP applications benefit of thread level parallelism exploration thanks to their loop-based behavior. However, even applications with high TLP could still obtain some performance improvement by also exploiting ILP. Hence, in a MPSoC design ILP techniques also should be investigated to conclude what is the best fit considering the design requirements. Finally, this subsection also shows that replications of simple processing elements leaves a significant optimization possibility unexplored, indicating that heterogeneous MPSoC could be a possible solution to balance the performance of the system.

3 MPSoC Classification

Due to the large amount of application domains that the MPSoC design principle can be applied to, some methods to define the advantages/disadvantages of each one should be defined. Commonly, a MPSoC design is classified only considering its architecture type, as homogeneous or heterogeneous. In this section the classification is done from two points of view: architecture and organization modeling.

3.1 Architecture View Point

When processors are classified under the architecture view point, the instruction set architecture (ISA) is the parameter used to distinguish them. The first step of a processor design is the definition of its basic operation capabilities, aiming to create an ISA to interface those operations to the high level programmer. Generally, an ISA is typically classified as reduced instruction set computer (RISC) or complex instruction set computer (CISC). The strategy of RISC architectures is making the processor data path control as simple as possible. Their instructions

set are non-specialized, covering only the basic operations needed to make simple computations. Besides its reduced instruction set, the genuine RISC architectures provide additional features like instructions issued at every clock cycle execution, special instructions to perform memory accesses, register-based operations and a small clock cycle, thanks to their pipelined micro-operations. On the other hand, the CISC architecture strategy aims at supporting all required operations on hardware, building specialized instructions to better support high level software abstractions. In fact, a single CISC instruction reflects the behavior of a several RISC instructions, with different amount of execution cycles and memory accesses. Hence, the data path control becomes more complex and harder to manage than RISC architectures. Both strategies are widely employed in processor architectures, always guided by the requirements and constraints of the design. DSP processors better fit the CISC characteristics, since there are many specialized instructions to provide efficient signal processing execution. Multiply-accumulate, round arithmetic, specialized instructions for modulo addressing in circular buffers and bit-reversed addressing mode for FFT are some examples of specialized CISC instructions focused on accelerating specific and critical parts of the DSP application domain.

In the architecture point of view, there are several ways to combine processing elements inside a die. Figure 1 presents three examples of MPSoC with different processing element arrangement. As can be noticed in this figure, MPSoC #1 is composed of four processing elements with the very same ISA, and this MPSoC arrangement is classified as a homogeneous MPSoC architecture. The same classification is given for the MPSoC #2 that only differs from the first for the diversity on computing time of its processing elements. Details about this organization fashion are given in the next section.

Homogeneous MPSoC architectures have been widespread in general-purpose processing domain. As already explained, the current difficulties found on increasing the applications performance by using ILP techniques, aggregated to the large density provided by the high transistor level integration, encouraged the coupling of several GPP into a single die. Since, in most cases, the baseline processor is already validated, the inter-processor communication design is the unique task to be performed on the development of a homogeneous MPSoC architecture (although this turned out to be not such an easy task). In the software side, programming models are used to split the code among the homogeneous cores, making the hard task of coding parallel software easier. OpenMP [4] and MPI [14] are the most used programming models providing a large number of directives that facilitate such a hard task. More details about programming languages are provided in the last section.

For almost 5 years now, Intel and AMD have been using this approach to speed up their high-end processors. In 2006, Intel has shipped its first MPSoC based on homogeneous architecture strategy. Intel Core Duo is composed of two processing elements that make communication among themselves through an on-chip cache memory. In this project, Intel has thought beyond the benefits of MPSoC employment and created an approach to increase the process yield. A new processor market line, called Intel Core Solo, was created aiming to increase the process yield

by selling even Core Duo dies with manufacturing defects. In this way, Intel Core Solo has the very same two-core die as the Core Duo, but only one core is defect free.

Recently, embedded general-purpose RISC processors are following the trend of high-end processors coupling many processing elements, with the same architecture, on a single die. Early, due to the hard constraints of these designs and the few parallel applications that would benefit from several GPP, homogeneous MPSoC were not suitable for this domain. However, the embedded software scenario is getting similar to a personal computer one due the large amount of simultaneous applications running over its GPP. ARM Cortex-A9 MPSoC processor is the pioneer to employ homogeneous multiprocessing approach into embedded domain, coupling four Cortex-A9 cores into a single die. Each processing element uses powerful techniques for ILP exploration, as superscalar execution and SIMD instruction set extensions, which closes the gap between the embedded processor design and high-end general purpose processors.

Heterogeneous MPSoC architectures support different market requirements than homogeneous ones. Generally, the main goal of the homogeneous approach is to provide efficiency on a large range of applications behaviors, being widespread used in GPP processor. On the other hand, heterogeneous approach aims to supply efficient execution on a defined range of application behavior, including some particular ISA extensions to handle specific software domains. MPSoC #3 (Fig. 1) better exemplifies a heterogeneous architecture composed of three different ISAs. This architecture style is widely used in the mobile embedded domain, due to the need for high performance with energy efficiency. The heterogeneous MPSoC assembly depends a lot on the performance and power requirements of the application. Supposing a current cell phone scenario where there are several applications running over the MPSoC. Commonly, designers explore the design space by trying to find critical parts of code that could be efficiently executed in hardware. After, these parts are moved to hardware, achieving better performance and energy efficiency. In an MPSoC used by a current cell phone there are many heterogeneous processing elements providing efficient execution for specific tasks, like video decompression and audio processing.

Samsung S3C6410 better illustrates the embedded domain trend to use MPSoCs. This heterogeneous architecture handles the most widely used applications on embedded devices like multimedia and digital signal processing. Samsung's MPSoC is based on an ARM 1176 processor, which includes several ISA extensions, such as DSP and SIMD processing, aiming to increase the performance of those applications. Its architecture is composed of many other dedicated processing elements, called hard-wired multimedia accelerators. Video Codec (MPEG4, H.264 and VC1), Image Codec (JPEG), 2D and 3D accelerators have become hard-wired due to the performance and energy constraints of the embedded systems. This drastic design change illustrates the growth and maturity phase of the functionality lifecycle discussed in the beginning of this chapter. In this phase, the electronic consumer market already has absorbed these functionalities, and their hard-wired execution is mandatory for energy and performance efficiency.

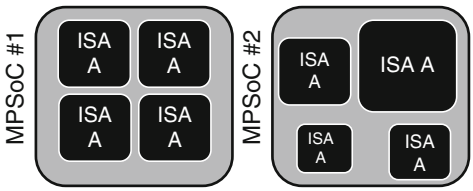

Fig. 10 Examples of homogeneous and heterogeneous organizations

3.2 Organization View Point

In contrast with the previous section, this part of the chapter explores the heterogeneous versus homogeneous arrangement considering the MPSoC organization as a classification parameter. This means that all MPSoC designs discussed here have the same ISA, and they are classified by the use of similar/dissimilar processor organization.

The homogeneous MPSoC organization, as already explained, has been widespread in general-purpose processing. Intel and AMD replicate their cores into a single die to explore the process/thread parallelism, moving the responsibility of process/thread scheduling to the operating system or software team. Also, the ARM Cortex design has introduced MPSoC in the embedded domain with a homogeneous MPSoC organization employment. When a simple replication of the very same processing elements occurs, the design is classified as homogeneous, both in architecture and organization. Their processing elements have the same processing capability, area and energy consumption, which delimits the design space exploration, since no special task scheduling could be employed.

The process/thread scheduling, in an MPSoC design, is an important feature to be explored, in order to create a tradeoff between the requirements and design constraints, aiming to find the right balance between performance and power consumption. Supposing the two approaches illustrated in Fig. 10, MPSoC #1 is composed by replications of the same processing element, being classified as homogeneous both in architecture and organization. On the other hand, MPSoC #2, from the organization point of view, demonstrates different processing elements despite having the same ISA. The last MPSoC fashion is classified as a heterogeneous MPSoC in the organization point of view, and homogeneous MPSoC in the architecture point of view. MPSoC #1 supports only naive task scheduling approaches, since all processing elements have the same characteristics. In this organization fashion, the task is always allocated in the same processing element organization, meaning that there is no choosing for a smart task scheduling to achieve certain design requirements, such as energy efficiency. In contrast, MPSoC #2 provides flexibility on task scheduling. An efficient algorithm could be developed to support design tradeoffs regarding requirements and constraints of the device. Supposing a scenario with three threads (thread#1, thread#2, and thread #3) that require the following performance level: low, very low and very high. The MPSoC #1 scheduling approach simply ignores the computation requirements, executing the three threads into their powerful processing elements, causing a waste of

energy resources. An efficient scheduling algorithm can be employed in MPSoC #2 to avoid the previous drawback, exploiting its heterogeneous organization. Supposing a dynamic task scheduling algorithm that saves information about threads execution, one can fit the thread computation need regarding the different processing capabilities available in MPSoC #2 which allows an energy-efficient design. For the given example, thread#1 is executed in the middle-size processing element, thread#2 in the smallest one and thread#3 in the largest processing element.

4 Successful MPSoC Designs

Many successful MPSoC solutions are in the market, for both high-end performance personal computers and highly-constrained mobile embedded devices. Commonly, as already explained, high-end performance personal computers provide robust GPP use with architecture and organization shaped on a homogeneous way due to the large variation of software behavior that runs over these platforms. On the other hand, MPSoCs for mobile embedded devices supply many ISAs, through application specific instruction set processor to provide an efficient execution on a restricted application domain. This subsection delimits MPSoC exploration to embedded domain due their widely usage of DSP processors. Some successful embedded MPSoC platforms are discussed, demonstrating the trends for mobile multiprocessors.

4.1 OMAP Platform

In 2002, Texas Instruments has launched in the market an Innovator Development kit (IDK) targeting high performance and low power consumption for multimedia applications. IDK provides an easy design development, with open software, based on a customized hardware platform called open multimedia applications processor (OMAP). Since its launching, OMAP is a successful platform being used by the embedded market leaders like Nokia with its N90 cell phones series, Samsung OMNIA HD and Sony Ericsson IDOU. Currently, due to the large diversity found on the embedded consumer market, Texas Instruments has divided the OMAP family in two different lines, covering different aspects. The high-end OMAP line supports the current sophisticated smart phones and powerful cell phone models, providing pre-integrated connectivity solutions for the latest technologies (3G, 4G, WLAN, Bluetooth and GPS), audio and video applications (WUXGA), including also high definition television. The low-end OMAP platforms cover down-market products providing older connectivity technologies (GSM/GPRS/EDGE) and low definition display (QVGA).

Launched in 2009, OMAP4440 covers the connectivity besides high-quality video, image and audio support. This mobile platform came to supply the need for

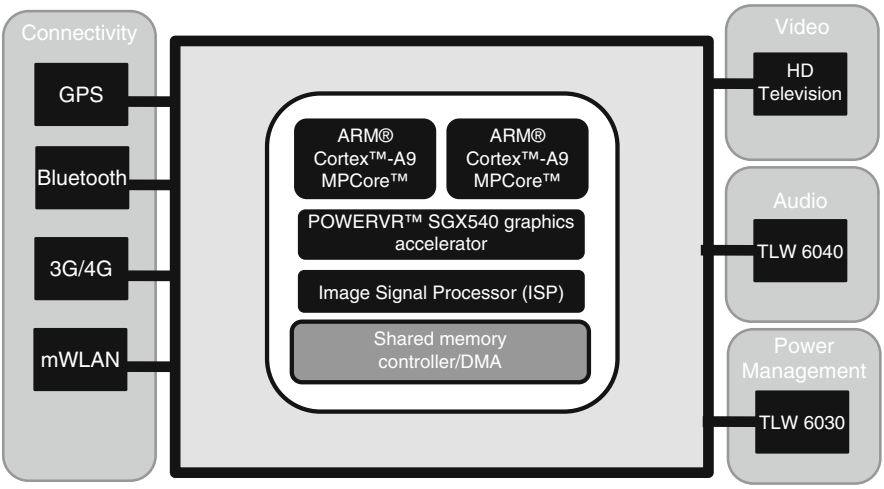

Fig. 11 OMAP4440 block diagram

the increasingly multimedia applications convergence in a single embedded device. As can be seen in Fig. 11, this platform incorporates a dual-core ARM Cortex A9 MPCore providing higher mobile general-purpose computing performance. A novel power management hardware technique available in ARM Cortex A9 MPCore balances the power consumption with the performance requirements, activating only the cores that are needed for a particular execution. Also, due to the high performance requirement of today smart phones, up to eight threads can be concurrently fired in the MPCore, since each core is composed of four single-core Cortex A9. The single-core ARM Cortex A9 implements speculative issue out-of-order superscalar execution, SIMD instruction set and DSP extensions, showing almost the same processing power as a personal computer into an embedded mobile platform. Excluding the ARM Cortex MPCore, the remainder processing elements are dedicated to multimedia execution.

The PowerVR™ SGX540 multi-threaded graphics accelerator employs state-of-art support for 2D and 3D graphics, including OpenGL and EGL APIs. Several image enhancements like digital anti-aliasing, digital zoom and auto-focus are provided by the Image Signal Processor (ISP), which focus on increasing the quality of the digital camera. The processing capabilities of ISP enables picture resolutions of up to 20-megapixel, with an interval of 1 s between shots. The high memory bandwidth needed for ISP is supplied by a DMA controller that achieves a memory-to-memory rate of 200 megapixel/s at 200 MHz. To conclude the robust multimedia accelerators scenario, OMAP4440 allows a true 1080p multi-standard high definition record and playback at 30 frames per second rate, by its IVA3 hardware accelerator. IVA DSP processor supports a broad of multimedia codec standards, including MPEG4 ASP, MPEG-2 MP, H.264 HP, ON2 VP7 and VC-1 AP.

The large amount of OMAP4440 processing elements makes its power dissipation rates unfeasible for mobile embedded devices. Texas Instruments has handled this drawback by integrating on the OMAP 440 design a power management technology. Smartreflex2 supports, both at the hardware and software level, the following power management techniques: Dynamic Voltage and Frequency Scaling (DVFS), Adaptive Voltage Scaling (AVS), Dynamic Power Switching (DPS), Static Leak Management (SLM) and Adaptive Body Bias (ABB). DVFS supports multiple voltage values, aiming to balance the power supply and frequency regarding the performance needed in a given execution time. AVS is coupled to DVFS to provide a greater balance on power consumption, considering hardware features like temperature. The coupling of both techniques delivers the maximum performance of OMAP4440 processing elements with the minimum possible voltage at every computation. Regarding leakage power, the 45 nm OMAP4440 MPSoC die already provides a significant reduction in the consumption. DPS and SLM techniques allow lowest stand-by power mode avoiding leakage power. Finally, ABB works at transistor level enabling dynamic changes on transistor threshold voltages aiming to reduce leakage power. All techniques are supported by the integrated TLW6030 power management chip illustrated in Fig. 11. In order to support energy efficient audio processing, Texas has included the TWL6040 integrated chip that acts as an audio back-end chip, providing up to 140 h of music playback in airplane mode.

Recently, Texas Instrument released its latest high-end product, OMAP543x family platform [27]. Several technological improvements over the OMAP4440 were coupled to OMAP5430. The ARM Cortex-A9 was replaced to ARM Cortex-A15 running at 2 GHz. ARM reports that the former performs 40% faster the latter. Two ARM Cortex-M4 were inserted in the SoC to provide low power consumption when offload computation mode is activated. The multicore PowerVR SGX544-MPx replaced the single core SGX540 graphics accelerator, being capable of encapsulate up to 16 processing elements, now supporting OpenGL and DirectX application programming interfaces. Its 28 nm fabrication process produces lower power consumption than the 45 nm of OMAP4440, which improves the battery life. The OMAP543x family is divided in two target markets: area-sensitive and cost-sensitive. The former includes the OMAP5430 platform and it is targeted to smartphones and tablets encapsulating several imaging interfaces. The latter includes the OMAP5432 platform being focused on mobile computing which requires higher bandwidth and lower interface to handle images. In terms of organization and architecture, Texas Instrument keeps both heterogeneous encapsulating more specialized processing elements to achieve low-power computation.

4.2 Samsung SoCs

As OMAP, Samsung MPSoC designs are focused on multimedia-based development. Their projects are very similar due to the increasing market demand for powerful multimedia platforms, which stimulates the designer to take the same

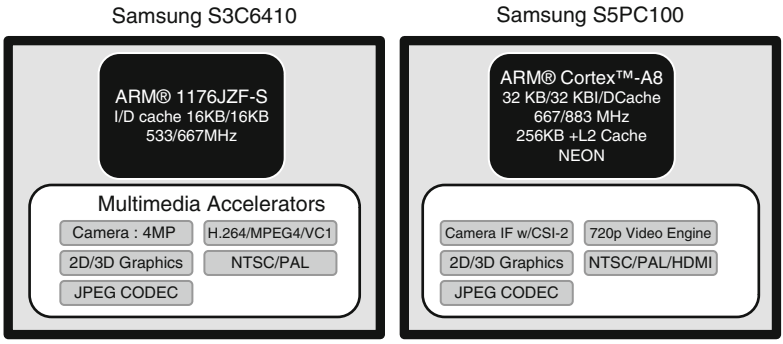

Fig. 12 Samsung S3C6410 and S5PC100 MPSoC block diagrams

decision to achieve efficient multimedia execution. Commonly, the integration of specific accelerators is used, since this reduces the design time avoiding validation and testing time.

In 2008, Samsung has launched the most powerful of the Mobile MPSoC family. At first, S3C6410 was a multimedia MPSoC like OMAP4440. However, after its deployment in the Apple iPhone 3G employment, it has become one of the most popular MPSoCs, shipping three million units during the first month after the launching. Recently, Apple has developed iPhone 3GS, which assures better performance with lower power consumption. These benefits are supplied by the replacement of the S3C6410 architectures with the high-end S5PC100 version. In this subsection, we will explore both architecture highlighting the platform improvements between S3C6410 and S5PC100. The S5PC100 was the last Samsung SoC employed by Apple in the iPhone. After that, Apple encapsulates his own chips in iPhone 4 and 4S, the Apple A4 and Apple A5 chips, respectively. However, these chips still being fabricated by Samsung.

Following the multimedia-based MPSoC trend, Samsung platforms are composed of several application specific accelerators building heterogeneous MPSoC architectures. S3C6410 and S5PC100 have a central general-purpose processing element, in both cases ARM-based, surrounded by several multimedia accelerators tightly targeted to DSP processing. As can be noticed in Fig. 12, both platforms skeleton follows the same execution strategy changing only the processing capability of their IP cores. This is an interesting design modeling approach, since while traditional architecture designs suffer for long design/validate/test phases to build a new architecture, MPSoC ones shorter these design times only by replacing their low-end cores for high-end ones, creating a new MPSoC architecture. This can be verified in the Fig. 12, small platform changes are done from S3C6410 to S5PC100 aiming to increase the MPSoC performance. More specifically, a nine-stage pipelined ARM 1176JZF-S core with SIMD extensions is replaced for a 13-stage superscalar pipelined ARM Cortex A8 providing greater computation capability for general-purpose applications. Besides its double-sized L1 cache

compared with ARM1176JZF-S, ARM Cortex A8 also includes a 256KB L2 cache avoiding external memory accesses due L1 cache misses. NEON ARM technology is included in ARM Cortex A8 to provide flexible and powerful acceleration for intensive multimedia applications. Its SIMD based-execution accelerates multimedia and signal processing algorithms such as video encode/decode, 2D/3D graphics, speech processing, image processing at least 2× of previous SIMD technology.

Regarding multimedia accelerators, both MPSoCs are able to provide suitable performance for any high-end mobile devices. However, S5PC100 includes the latest codec multimedia support using powerful accelerators. A 720p multi format codec (MFC) video is included aiming to provide high quality capture and playback at 30 frames per second rate. Also, previous video codec as MPEG4, VCI and H.264 are available in this platform. Besides the S3C6410 TV out interfacing PAL and NTSC, the new MPSoC version includes high definition multimedia interface (HDMI) standard allowing high definition video formats. Summarizing, all technologies included in S5PC100 platform accelerate multimedia applications employing DSP approaches.

In 2010, Samsung launched the first SoC of its *Exynos* platforms, named as *Exydos* 3110 [28], which was employed in the high-end cell phones and tablets developed in such year (i.e. Samsung Galaxy S, Samsung Galaxy Tab). The second SoC of this family, named as *Exynos* 4210 [29], keeps being the most employed SoC in the high-end cell phones of Samsung. The best seller, Galaxy S II (i9100), contain an *Exynos* 4210 composed of dual-core Cortex A9 running at 1.2 GHz coupled to an ARM 4-Core Mali-400MP graphics processing unit. However, the newer version of this cell phone, Galaxy S II (i9100G), changes the *Exynos* 4210 to an already discussed SoC, a 1.2 GHz dual core TI OMAP 4430 with a PowerVR SGX540 GPU.

4.3 Apple SoCs

In March of 2010, Apple has announced its first SoC named as Apple A4, an ARM Cortex-A8 running at 1 GHz and a PowerVR SGX535 GPP were coupled in a single die to push up the performance of the iPAD, iPhone 4 and iTouch. In 2011, Apple introduced the multicore concept in their devices launching Apple A5 SoC. This SoC improves the performance over the Apple A4 by integrating a dual-core ARM Cortex-A9 together with a dual-core PowerVR SGX543MP, iPAD 2 and iPhone 4S explore such powerful performance. Recently, Apple has announced its latest SoC, named as Apple A5X. The changes over the previous version happen only in the GPU. Apple A5X encapsulates a quad-core PowerVR SGX544MP4 which drastically increases its video processing capabilities. The new iPAD, equipped with such engine, achieves a display resolution of 2,048 × 1,536, meaning a million more pixels than the HDTV standard.

4.4 Other MPSoCs

Other MPSoCs have already been released in the market, with different goal from the architectures discussed before. Sony, IBM and Toshiba have worked together to design the Cell Broadband Engine Architecture [3]. The Cell architecture combines a powerful central processor with eight SIMD-based processing elements. Aiming to accelerate a large range of application behaviors, the IBM PowerPC architecture is used as GPP processor. Also, this processor has the responsibility to manage the processing elements surrounding it. These processing elements, named as synergistic processing elements (SPE), are built to support streaming applications with SIMD execution. Each SPE has a local memory, but no hardware is employed to manage it, which avoid that these memories directly access the system memory. These facts make the software development for the Cell processor even more difficult, since the software team should be aware of this local memory, and manage it at the software level to better explore the SPE execution. Despite its high processing capability, the Cell processor does not yet have a large market acceptance due to the intrinsic difficulty to produce software. Sony has lost parts of the gaming entertainment market after the Cell processor was deployed in the Playstation console family, since game developers had not enough knowledge to efficiently explore the complex Cell architecture.

Homogeneous MPSoC organization is also explored in the market, mainly for personal computers with general-purpose processors, because of the huge amount of different applications these processors have to face, and hence it is more difficult to define specialized hardware accelerators. In 2005, Sun Microsystems announced its first homogeneous MPSoC, composed of up to eight processing elements executing the full RISC SparcV9 instruction set. UltraSparc T1, also called Niagara [11], is the first multithreaded homogeneous MPSoC, and each processing element is able to execute four threads concurrently. In this way, Niagara can handle, at the same time, up to 32 threads. Recently, with the deployment of UltraSparc T2 [10], this number has grown to 64 concurrent threads. Niagara MPSoC family targets massive data computation with distributed tasks, like the market for web servers, database servers and network file systems.

Intel has announced its first MPSoC homogeneous prototype with 80 cores, which is capable of executing 1 trillion floating-point operations per second, while consuming 62 W [21]. Hence, the x86 instruction set architecture era could be broken, since their processing elements is based on the very long instruction word (VLIW) approach, letting to the compiler the responsibility for the parallelism exploration. The interconnection mechanism used on the 80-core MPSoC uses a mesh network to communicate among its processing elements [5]. However, even employing the mesh communication turns out to be difficult, due to the great amount of processing elements. In this way, this ambitious project uses a 20 MB stacked on-chip SRAM memory to improve the processing elements communication bandwidth.

Graphic processing unit (GPU) is another MPSoC approach aiming to accelerate graphic-based software. However, this approach has been arising as promise architecture also to improve general-purpose software. Intel Larrabee [17] attacks both applications domain thanks to its CPU- and GPU-like architecture. In this project Intel has employed the assumption of energy efficiency by simple cores replication. Larrabee uses several P54C-based cores to explore general-purpose applications. In 1994, P54C was shipped in CMOS 0.6 μm technology reaching up to 100 MHz and does not include out-of-order superscalar execution. However, some modifications have been done in the P54C architecture, like supporting of SIMD execution aiming to provide more powerful graphic-based software execution. The SIMD Larrabee execution is similar to but powerful than the SSE technology available in the modern x86 processors. Each P54C is coupled to a 512-bit vector pipeline unit (VPU), capable of executing, in one processor cycle, 16 single precision floating point operations. Also, Larrabee employs a fixed-function graphics hardware that performs texture sampling tasks like anisotropic filtering and texture decompression. However, in 2009, Intel discontinued Larrabee project.

NVIDIA Tesla [12] is another example of MPSoC based on the concept of a general-purpose graphic processor unit. Its massive-parallel computing architecture provides support to Compute Unified Device Architecture (CUDA) technology. CUDA, the NVIDIA's computing engine, supports the software developer by easing the application workload distribution and by providing software extensions. Also, CUDA provides permission to access the native instruction set and memory of the processing elements, turning the NVIDIA Tesla to a CPU-like architecture. Tesla architecture incorporates up to four multithreaded cores communicating through a GDDR3 bus, which provides a huge data bandwidth.

In 2011, NVIDIA introduced the project named Tegra 3 [30], also called Kal-el mobile processor. This project is the first to encapsulate four processors in a single die for mobile computation. The main novelty introduced by this project is the Variable Symmetric Multiprocessing (vSMP) technology. vSMP introduces a fifth processor named "Companion Core" that executes tasks a low frequency for active standby mode, as mobile systems tend to keep in this mode for most time. All five processors are ARM Cortex-A9, but the companion core is built in a special low power silicon process. In addition, all cores can be enabled/disabled individually and when the active standby mode is on, only the "Companion Core" works, so battery life can significant improved. NVIDIA reports that the switching from the companion core to the regular cores are supported only by hardware and take less than 2 ms being not perceptible to the end users. In comparison with Tegra 2, NVIDIA previous platform, vSMP achieves up to 61% of energy savings on running HD video playback. Tegra 3 is inside of several tablets developed by Asus and Acer, such as Asus Eee Pad Trasformer Prime, Trasformer Pad Infinity and Acer Iconia.

Summarizing all MPSoC discussed before, Table 1 compares their main characteristics showing their differences depending on the target market domain. Heterogeneous architectures, such as OMAP, Samsung and Cell, incorporate several specialized processing elements to attack specific applications for highly-

Table 1 Summarized MPSoCs

	Architecture	Organization	Cores	Multithreadedcores	Intercon-nection
OMAP 4440	Heterogeneous	Heterogeneous	2 ARM Cortex A91 PowerVR540 Image Signal Processors	Yes (GPU)	Integrated Bus
OMAP543x	Heterogeneous	Heterogeneous	2 ARM Cortex A152 ARM Cortex M41 PowerVR SGX544 Image Signal Processors	Yes (GPU)	Integrated Bus
Samsung S3C6410/S5PC100	Heterogeneous	Heterogeneous	ARM1176JZF-S5 multimedia accelerators	No	Integrated Bus
Samsung Exynos 3110	Heterogeneous	Heterogeneous	1 ARM Cortex A81 PowerVR540	No	Integrated Bus
Samsung Exynos 4210	Heterogeneous	Heterogeneous	2 ARM Cortex A91 ARM Mali 400 MP4	No	Integrated Bus
NVIDIA Tegra 3	Heterogeneous	Heterogeneous	1Quad ARM Cortex-A15 MPCore + companion core1 GeForce GPU	No	Integrated Bus
Cell	Heterogeneous	Heterogeneous	1 PowerPC 8 SPE	No	Integrated Bus
Niagara	Homogeneous	Homogeneous	8 Sparc V9 ISA	Yes (4 threads)	Crossbar
Intel 80-Cores	Homogeneous	Homogeneous	80 VLIW	No	Mesh
Intel Larrabee	Homogeneous	Homogeneous	n p54C x86 cores SIMD execution	No	Integrated Bus
NVIDIA Tesla (GeForce8800)	Homogeneous	Homogeneous	128 Stream Processors	Yes (up to 768 threads)	Network

constrained mobile or portable devices. These architectures have multimedia-based processing elements, following the trend of embedded systems. Homogeneous architectures still use only homogeneous organizations, coupling several processing elements with the same ISA and the processing capabilities. Heterogeneous organizations have not been used on homogeneous architectures, since the variable processing capability is supported dynamically by power management like DVFS. Unlike heterogeneous architectures, homogeneous ones aims at the general-purpose processing market, handling a wide range of applications behavior by replicating general purpose processors.

5 Open Problems

Due to the short history of MPSoCs, several design points are still open. In this section, we discuss two open problems of MPSoC design: interconnection mechanism and MPSoC programming models.

5.1 Interconnection Mechanism

The interconnection mechanism has a very important role in an MPSoC design, since it is responsible for supporting the exchange of information between all MPSoC components, typically between processing elements or processing elements and some storage component. The development of the interconnection mechanism in a MPSoC should take into account the following aspects: parallelism, scalability, testability, fault tolerance, reusability, energy consumption and communication bandwidth [13]. However, there are several communication interconnection approaches that provide different qualitative levels regarding the cited aspects. As can be noticed in Table 1, there is no agreement on the interconnection mechanism, since each design has specifics constraints and requirements that guide the choices of the communication infrastructure, always considering its particular aspects.

Commonly, buses are the most used mechanism on current MPSoC designs. Current buses can achieve high speeds, and buses have additional advantages like low cost, easy testability and high communication bandwidth, encouraging the use of buses in MPSoC designs. The weak points of this approach are poor scalability, no fault tolerance and no parallelism exploitation. However, modifications on its original concept can soften these disadvantages, but could also affect some good characteristics. Segmented bus is an original bus derivative to increase the performance, communication parallelism exploration and energy savings [7]. This technique divides the original bus in several parts, enabling concurrent communication inside of each part. However, bus segmentation impacts on the scalability, and makes the communication management between isolated parts harder. Besides their disadvantages, for obvious historical reasons buses are still widely used in MPSoC

designs. Intel and AMD are still using integrated buses to make the communication infrastructure on their high-end MPSoC, due to the easier implementation and high bandwidth provided.

Crossbars are widely used on network hardware like switches and hubs. Some MPSoC designers have been employing this mechanism to connect processing elements [10]. This interconnection approach provides huge performance, allowing communication between any processing elements and the smallest possible time. However, high area cost, energy consumption and poor scalability discourage its employment. AMD Opteron family and Sun Niagara use crossbars to support high communication bandwidth within their general-purpose processors.

Network-on-chip has been emerging as a solution to couple several processing elements [21]. This approach provides high communication parallelism, since several connecting paths are available for each node. In addition, as the technology scales, wire delays increase (because of the increased resistance derived form the smaller cross-section of the wire), and hence shorter wires, as used in NoCs could soften this scaling drawback. Also, its explicit modular shape positively affects the processing elements scalability, and can be explored by a power management technique to support the simple turning off of idle components of the network on chip. NoC disadvantages include the excessive area overhead and high latency of the routers. Intel 80-core prototype employs a mesh style network-on-chip interconnection to supply the communication of its 80 processing elements.

5.2 MPSoC Programming Models

For decades, many ILP exploration approaches were proposed to improve the processor performance. Most of those works employed dynamic ILP exploration at hardware level, becoming an efficient and adaptive process used in superscalar architectures, for instance. Also, traditional ILP exploration free software developers from the hard task to explicit, in the source code, those parts that can be executed in parallel. Some works [19] can even translate code with enough ILP into TLP, so that more than one core can execute the code, exploiting ILP also at the single issue core level, when embedded in a multiprocessing device. However, MPSoC employment relies on manual source code changes to split the parallel parts among the processing elements. Currently, software developers must be aware of the MPSoC characteristic like the number of processing elements to allocate the parallel code. Thus, the sequential programming knowledge should migrate to the parallel programming paradigm. Due to these facts, parallel programming approaches have been gaining importance on computing area, since easy and efficient code production is fundamental to explore the MPSoC processing capability.

Communication between processing elements is needed whenever information exchange among the threads is performed. Commonly, this communication is based either on message passing or on shared memory techniques. Message passing leaves the complex task of execution management to the software description level.

The software code should contain detailed description of the parallelization process since the application developer has the complete control to manage the process. The meticulous message passing management by the software becomes slower than shared memory. However, this approach enables robust communication employment providing the complete control to the software developer to make this job. Message Passing Interface (MPI) [14] is a widely used standard protocol that employs message passing communication mechanism. MPI provides an application programming interface (API) that specifies a set of routines to manage inter processes communication. The advantage of MPI employment over other mechanisms is that both data and task parallelism can be explored, with the cost of more code changes to achieve parallel software. In addition, MPI relies on network communication hardware, which guides the performance of the entire system.

Shared memory communication uses a storage mechanism to hold the necessaries information for threads communication. This approach provides simpler software development, since thanks to a global addressing system most of the communication drawbacks are transparent to the software team. The main drawback of shared memory employment is the bottleneck between processing element and memory, since several threads could try to concurrently access the same storage element at a certain time. Memory coherency also can be a bottleneck for shared memory employment, since some mechanism should be applied to guarantee data coherency. OpenMP [4] employs shared memory communication mechanism to manage the parallelization process. This approach is based on a master and slave mechanism, where the master thread forks a specified number of slave threads to execute in parallel. Thus, each thread executes a parallelized section of the application into a processing element independently. OpenMP provides easier programming than MPI, with greater scalability, since a smaller number of code changes should be done to increase the number of spawned threads. However, in most cases OpenMP code coverage is limited to highly parallel parts and loops. This approach is strongly based on a compiler, which must translate the OpenMP directives to recognize what section should be parallelized.

6 Future Research Challenges

As discussed throughout this chapter, although MPSoCs can be considered a consolidated strategy for the development of high performance and low energy products, there are still many open problems that required extra research effort. Software partitioning is one of the most important open issues. That is, taking a program developed in the way programs have been developed for years now, and making it work in an MPSoC environment by the use of automatic partitioning is very difficult already for the homogeneous case, not to mention the heterogeneous one. Matching a program to an ISA and to a certain MPSoC organization is an open and challenging problem.

Another research direction should contemplate hardware development. From the results shown in Sect. 2 it is clear that heterogeneous SoCs, capable of exploring parallelism at the thread and also at the instruction level, are the most adequate to obtain real performance and energy gains. Unfortunately, in an era where fabrication costs demand huge volumes, the big question to be answered regards the right amount of heterogeneity to be embedded in the MPSoC fabric.

References

1. Anantaraman, A., Seth, K., Patil, K., Rotenberg, E., Mueller, F.: Virtual simple architecture (VISA): Exceeding the complexity limit in safe real-time systems. In: Proc. 30th Annual Int. Symposium Computer Architecture, pp. 350–361. San Diego, CA (2003)
2. Bergamaschi, R., Han, G., Buyuktosunoglu, A., Patel, H., Nair, I., Dittmann, G., Janssen, G., Dhanwada, N., Hu, Z., Bose, P., Darringer, J.: Exploring power management in multi-core systems. In: Proc. Asia and South Pacific Design Automation Conf. Seoul, Korea (2008)
3. Chen, T., Raghavan, R., Dale, J.N., Iwata, E.: Cell broadband engine architecture and its first implementation: A performance view. IBM J. Res. Dev. **51**(5), 559–572 (2007)
4. Dagum, L., Menon, R.: OpenMP: An industry-standard API for shared-memory programming. IEEE Comput. Sci. Eng. **5**(1), 46–55 (1998)
5. Du, J.: Inside an 80-corechip: The on-chip communication and memory bandwidth solutions (2007). URL http://blogs.intel.com/research/2007/07/inside_the_terascale_many_core.php
6. Freescale, Inc. URL http://www.freescale.com/webapp/sps/site/ homepage.jsp
7. Guo, J., Papanikolaou, A., Marchal, P., Catthoor, F.: Physical design implementation of segmented buses to reduce communication energy. In: Proc. Asia and South Pacific Design Automation Conference, pp. 42–47. Yokohama, Japan (2006)
8. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: A free, commercially representative embedded benchmark suite. In: Proc. IEEE Int. Workshop Workload Characterization, pp. 3–14. Austin, TX (2001)
9. Intel core i7 processor SDK webinar: Q and A transcript from the 8:00 a.m. PST. URL http://software.intel.com/sites/webinar/corei7-sdk/intel-core-i7-8am.doc
10. Johnson, T., Nawathe, U.: An 8-core, 64-thread, 64-bit power efficient Sparc SoC (Niagara2). In: Proc. Int. Symposium Physical Design, p. 2. Austin, TX (2007)
11. Kongetira, P., Aingaran, K., Olukotun, K.: Niagara: A 32-way multithreaded Sparc processor. IEEE Micro 25(2), 21–29 (2005)
12. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro 28(2), 39–55 (2008)
13. Marcon, C., Borin, A., Susin, A., Carro, L., Wagner, F.: Time and energy efficient mapping of embedded applications onto NoCs. In: Proc. Asia and South Pacific-Design Automation Conference (2005)
14. Pacheco, P.S.: Parallel Programming with MPI. Morgan Kaufmann Publishers, Inc. (1996)
15. Reifel, M., Chen, D.: Parallel digital signal processing: An emerging market. Application Note (1994). URL http://focus.ti.com/lit/an/spra104/spra104.pdf
16. Rutzig, M.B., Beck, A.C., Carro, L.: Dynamically adapted low power ASIPs. In: Reconfigurable Computing: Architectures, Tools and Applications, Lecture Notes In Computer Science, vol. 5453, pp. 110–122. Springer-Verlag, Berlin, Germany (2009)
17. Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Hanrahan, P.: Larrabee: A many-core x86 architecture for visual computing. In: ACM SIGGRAPH 2008 Papers, pp. 1–15. Los Angeles, CA (2008)
18. Shi, K., Howard, D.: Challenges in sleep transistor design and implementation in low-power designs. In: Proc. 43rd Annual Conf. Design Automation, pp. 113–116 (2006)

19. Song, Y., Kalogeropulos, S., Tirumalai, P.: Design and implementation of a compiler framework for helper threading on multi-core processors. In: Proceedings of the 14th international Conference on Parallel Architectures and Compilation Techniques, pp. 99–109 (2005)
20. IDC Analyze the Future. Available at http://www.idc.com/getdoc.jsp?containerId=prUS23297412
21. Vangal, S., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., Finan, D., Iyer, P., Singh, A., Jacob, T., Jain, S., Venkataraman, S., Hoskote, Y., Borkar, N.: An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS. In: Digest of Technical Papers IEEE Int. Solid-State Circuits Conf., pp. 98–589 (2007)
22. Wall, D.W.: Limits of instruction-level parallelism. SIGPLAN Not. **26**(4), 176–188 (1991)
23. Woo, D.H., Lee, H.H.: Extending Amdahl's law for energy-efficient computing in the many-core era. Computer 41(12), 24–31 (2008)
24. Diefendorff, K. Hal Makes Sparcs Fly. [S.l.]. 1999
25. Intel. Inside an 80-core chip: The on-chip communication and memory bandwidth solution, 2007. Available at: http://blogs.intel.com/research/2007/07/inside_the_terascale_many_core.php.
26. Zhao, G., Kwan, H.K., Lei, C.U., Wong, N.: Processor frequency assignment in three-dimensional MPSoCs under thermal constraints by polynomial programming. In: Proc. IEEE Asia Pacific Conf. Circuits Syst., pp. 1668–1671 (2008)
27. OMAP 5430 white paper. Available at http://www.ti.com/lit/an/swpt048/swpt048.pdf
28. Exynos 3 Single platform. Available at http://www.samsung.com/global/business/semiconductor/minisite/Exynos/products3110.html
29. Exynos 4 Dual platform. Available at http://www.samsung.com/global/business/semiconductor/minisite/Exynos/products4210.html
30. NVIDIA white paper. Available at http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-whitepaper-0911b.pdf
31. Intel Corp. White paper. Available at http://www.intel.com/content/dam/doc/white-paper/intel-microarchitecture-white-paper.pdf