1

Parallel Embedded Computing Architectures

Michael Schmidt, Dietmar Fey and Marc Reichenbach
Embedded Systems Institute, Friedrich-Alexander-University Erlangen-Nuremberg
Germany

1. Introduction

It was around the years 2003 to 2005 that a dramatic change seized the semiconductor industry and the manufactures of processors. The increasing of computing performance in processors, based on simply screwing up the clock frequency, could not longer be holded. All the years before the clock frequency could be steadily increased by improvements achieved both on technology and on architectural side. Scaling of the technology processes, leading to smaller channel lengths and shorter switching times in the devices, and measures like instruction-level-parallelism and out-of-order processing, leading to high fill rates in the processor pipelines, were the guarantors to meet Moore's law.

However, below the 90 nm scale, the static power dissipation from leakage current surpasses dynamic power dissipation from circuit switching. From now on, the power density had to be limited, and as a consequence the increase of clock frequency came nearly to stagnation. At the same time architecture improvements by extracting parallelism out of serial instruction streams was completely exhausted. Hit rates of more than 99% in branch prediction could not be improved further on without reasonable effort for additional logic circuitry and chip area in the control unit of the processor.

The answer of the industry to that development, in order to still meet Moore's law, was the shifting to real parallelism by doubling the number of processors on one chip die. This was the birth of the multi-core area (Blake et al., 2009). The benefits of multi-core computing, to meet Moore's law and to limit the power density at the same time, at least at the moment this statement holds, are also the reason that parallel computing based on multi-core processors is underway to capture more and more also the world of embedded processing.

2. Task parallelism vs. data parallelism

If we speak about parallelism applied in multi-cores, we have to distinguish very carefully which kind of parallelism we refer to. According to a classical work on design patterns for parallel programming (Mattson et al., 2004), we can define on the algorithmic level two kinds of a decomposition strategy for a serial program in a parallel version, namely *task parallelism* and *data parallelism*. The result of such a decomposition is a number of sub-problems we will call tasks in the following. If these tasks carry out different work among each other, we call this task parallelism. In task parallelism tasks are usually ordered according to their data dependencies. If tasks are independent of each other these tasks can be carried out concurrently, e.g. on the cores of a multi-core processor. If one task produces an output which is an input for another task, these tasks have to be scheduled in a time serial manner.

This situation is different in the case of a given problem which can be decomposed according to geometric principles. That means, we have given a 2D or 3D problem space which is divided in sub regions. In each sub region the same function is carried out. Each sub region is further subdivided in grid points and also on each grid point the same function is applied to. Often this function requires also input from grid points located in the nearest neighbourhood of the grid point. A common parallelization strategy for such problems is to process the grid points of one sub region in a serial manner and to process all sub regions simultaneously, e.g. on different cores. Also this function can be denoted as a task. As mentioned, all these tasks are identical and are applied to different data, whereas the tasks in task parallelism carry out different tasks usually. Furthermore, data parallel tasks can be processed in a complete synchronous way. That means, there are only geometric dependencies between these tasks and no casual time dependencies among the tasks, what is once again contrary to the case of task parallelism. If there are time dependencies then they hold for all tasks. That is why they are synchronous in the sense that all grid points are updated in a time serial loop.

Task parallelism we find e.g. in applications of Computational Science. In molecular biology the positions of molecules are computed depending on electrical and chemical forces. These forces can be calculated independent from each other. An example of a data parallelism problem is the solution of partial differential equations.

2.1 Task parallelism in embedded applications

Where do we find these task parallelism in embedded systems? A good example are automotive applications. The integration of more and more different functionality in a car, e.g. for infotainment, driver assistance, different electronic control units for valves, fuel injection etc. lead to a very complex diversity that offers a lot of potential for parallelization, naturally requiring diverse tasks. The desire why automotive goes to multi-core is based on two reasons. One time there are lot of real-time tasks to fulfill for which a multi-core technology offers in principle the necessary computing power. A further reason is the following one. Today nearly every control unit contains its own single core micro controller or micro processor. Multi-core technology in combination with a broadband efficient network system offers the possibility to save components, too, by migrating functionality that is now distributed among a quite large number of compute devices to fewer cores. Automotive is just one example for an embedded system domain in which task parallelism is the dominant potential for parallelization. Similar scenarios can be found for robotics and automation engineering.

2.2 Data parallelism in embedded applications

As consequence one can state that the main parallelization strategy for embedded applications is task parallelism. However, there is a smaller but not less important application field in which data parallelism occurs. Evaluating and analyzing of data streams in optical, X-ray or ultra sonic 3D metrology requires data parallelism in order to realize fast response times. Mostly image processing tasks, e.g. fast execution of correlations, have to be fulfilled in the mentioned application scenarios. To integrate such a functionality in smart cameras, or even in in the electronics of measuring or drill heads, is a challenge for future embedded system design. In this chapter, we lay a focus in particular to convenient pipeline and data structures for applying data parallelism in embedded systems (see Chapter 4).

3. Principles of embedded multi-core processors

3.1 Multi-core processors in embedded systems

In this subsection, we show briefly a kind of evolutionary development comprising a stepwise integration of processor principles, known from standard processors, into embedded processors. The last step of this development process is the introduction of multi-core technology in embedded processors. Representative for different embedded processors, we select in this chapter the development of the ARM processor family as it is described in (Stallings, 2006). Maybe the most characteristic highlights of ARM processors are their small chip die sizes and their low power requirements. Both features are of course of high importance for applications in embedded environments. ARM is a product of ARM Inc., Cambridge, England. ARM works as a fabless company, that means they don't manufacture chips, moreover they design microprocessors and microcontrollers and sell these designs under license to other companies. Embedded ARM architectures can be found in many handheld and consumer products, like e.g. in Apple's iPod and iPhone devices. Therefore, ARM processors are probably not only one of the most widely used processors in embedded designs but one of the most world wide used processors at all.

The first ARM processor, denoted as ARM1, was a 32-bit RISC (Reduced Instruction Set Computer) processor. It arose in 1985 as product of the company Acorn, which designed the first commercial RISC processor, the Acorn RISC Machine (ARM), as a coprocessor for a computer used at British Broadcasting Corporation (BBC). The ARM1 was expanded towards an integrated memory management unit, a graphics and I/O processor unit and an enhanced instruction set like multiply and swap instructions and released as ARM2 in the same year. Four years later, in 1989, the processor was equipped with a unified data and instruction level one (L1) cache as ARM3. It followed the support of 32-bit addresses and the integration of a floating-point unit in the ARM6, the integration of further components as System-on-Chip (SoC) in the ARM6, and static branch prediction units, deeper pipeline stages and enhanced DSP (Digital Signal Processing) facilities. The design of the ARM6 was also the first product of a new company, formed by Acorn, VLSI and Apple Computer.

In 2009 ARM released with the Cortex-A5 MPCore processor their first multi-core processor intended for usage in mobile devices. The intention was to provide one of the smallest and most power-efficient multi-core processor to achieve both the performance, that is needed in smartphones, and to offer low costs for cheap chip manufacturing. Exactly like the ARM11 MP Core, another multi-core processor from ARM, it can be configured as a device containing up to 4 cores on one processor die.

3.2 Brief overview of selected embedded multi-core architectures

The ARM Cortex A9 processor (ARM, 2007) signifies the second generation of ARM's multi-core processor technology. It was also intended for processing general-purpose computing tasks in computing devices, starting from mobile devices and ending up in netbooks. Each single core of an ARM Cortex A9 processor works as a superscalar out-of-order processor (see Figure 1). That means, the processor consists of multiple parallel operable pipelines. Instructions fetched in these pipelines can outpace each other so that they can be completed contrary to the order they are issued. The cores have a two-level cache system. Each L1 cache can be configured from 16 to 64 KB that is quite large for an embedded processor. Using such a large cache supports the design for a high clock frequency of 2 GHz in

order to speed-up the execution of a single thread. In order to maintain the coherency between the cache contents and the memory, a broadcast interconnect system is used. Since the number of cores is still small, the risk is low that the system is running in bottlenecks. Two of such ARM Cortex A9 processors are integrated with a C64x DSP (Digital Signal Processor) core and further controller cores in a heterogeneous multi-core system-on-chip solution called TI OMAP 4430 (Tex, 2009). This system is intended also as general-purpose processor for smart phones and mobile Internet devices (MIDs). Typical data parallel applications do not approve as very efficient for such processors. In this sense, the ARM Cortex A9 and the TI OMAP 4430 processors are more suited for task parallel embedded applications.

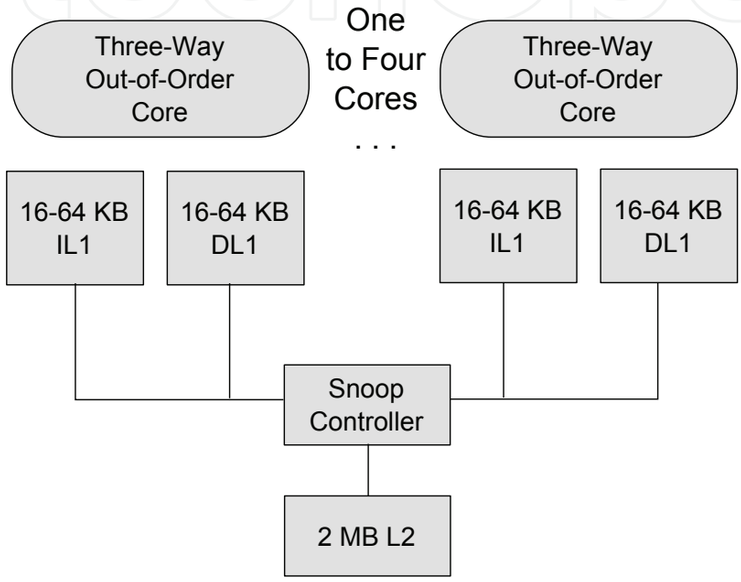


Fig. 1. Block diagram of the ARM Cortex-A9 MP, redrawn from (Blake et al., 2009)

Contrary to those processors, the ECA (Elemental Computing Array) (Ele, 2008) processor family targets to very low power processing of embedded data parallel tasks, e.g. in High Definition Video Processing or Software Defined Signal Conditioning. The architecture concept realized in this solution is very different from the schemes we find in the above described multi-core solutions. Maybe, it points in a direction also HPC systems will pursue in the future (see Chapter 5). The heart of that architecture is an array of fine-grain heterogeneous specialized and programmable processor cores (see Figure 2). The embedded processor ECA-64 consists of four clusters of such cores and each cluster aggregates one processor core operating to RISC principles and further simpler 15 ALUs which are tailored to fulfill specialized tasks. The programming of that ALUs happens similarly as it is done in Field-Programmable-Gate-Arrays (FPGAs).

An important constraint for the low power characteristics of the processors is the data-driven operation mode of the ALUs, i.e. the ALUs are only switched on if data is present at their inputs. Also the memory subsystem is designed to support low power. All processor cores in one cluster share a local memory of 32 kB. The access to the local memory has to be performed completely by software, which avoids to integrate sophisticated and power consuming hardware control resources. This shifts the complexity of coordinating concurrent

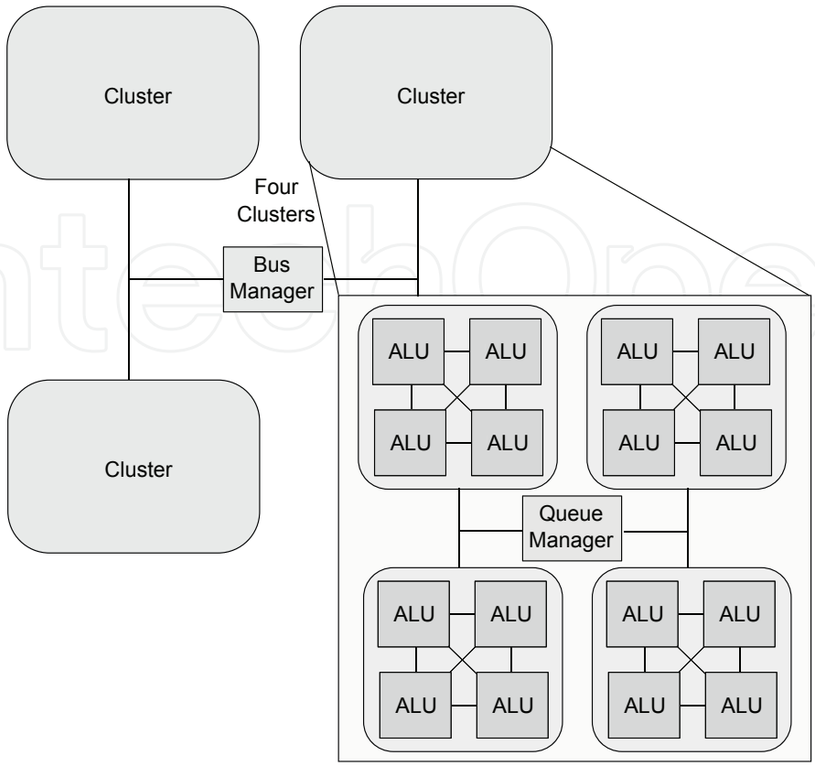


Fig. 2. Element CXI ECA-64 block diagram, redrawn from (Blake et al., 2009)

memory accesses to the software. The interconnect is hierarchical. Following the hierarchical architecture organization of the processor cores also the interconnect system has to be structured hierarchically. Four processor cores are tightly coupled via a crossbar. In one cluster four of these crossbar connected cores are linked in a point-to-point fashion using a queue system. On the highest hierarchical level the four clusters are coupled via a bus and a bus manager arbitrating the accesses of the clusters on the bus.

Hierarchically and heterogeneously organized processor, memory and interconnect systems, as we find it in the ECA processor, are pioneering in our view for future embedded multi-core architectures to achieve both high computing performance and low power processing. However, particular data parallelism applications require additional sophisticated data access patterns that consider the 2D or 3D nature of data streams given in such applications. Furthermore, they must be well-tailored to a hierarchical memory system to exploit the benefits such an organization offers. These are time overlapping of data processing and of data transfer to hide latency and to increase bandwidth by data buffering in pipelined architectures. To achieve that, we developed special data access templates, we will explain in detail in the next section.

4. Memory-management for data parallel applications in embedded systems

The efficient realization of applications with multi-core or many-core processors in an embedded system is a great challenge. With application-specific architectures it is possible to save energy, reduce latency or increase throughput according to the realized operations, in

contrast to the usage of standard CPUs. Besides the optimization of the processor architecture, also the integration of the cores in the embedded environment plays an important role. This means, the number of applied cores[1] and their coupling to memories or bus systems has to be chosen carefully, in order to avoid bottlenecks in the processing chain.

The most basic constraints are defined by the application itself. First of all, the amount of data to be processed in a specific time slot is essential. For processor-intensive applications the key task is to find an efficient processing scheme for the cores in combination with integrated hardware accelerators. The main problem in data-intensive applications is the timing of data provision. Commonly, the external memory or bus bandwidth is the main bottleneck in these applications. A load balancing between data memory access and data processing is required. Otherwise, there will be idle processor cores or available data segments cannot be fetched in time for processing.

Image processing is a class of applications which is mainly data-intensive and a clear example of a data parallel application in an embedded system. In the following, we will take a closer look at this special type of application. We assume a SoC with a multi-core processor and a fast but small internal memory (e.g. caches) and a large but slow external memory or alternatively a coupled bus system.

4.1 Embedded image processing

Image processing operations are basically distinguished in pre-processing operations and post-processing operations also known as image recognition (Bräunl, 2001). Image pre-processing operations, like filter operations for noise reduction, require only a local view on the image data. Commonly, an image pixel and its neighbours in a limited environment[2] are required for processing. Image recognition, on the other hand, requires a global view on the image and, therefore, a random access to the image pixels.

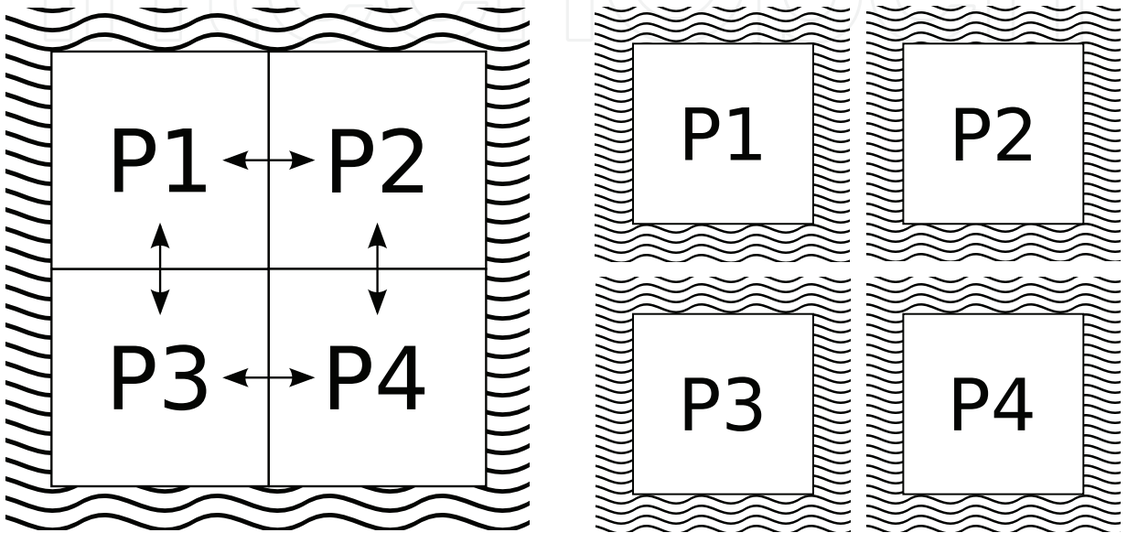
Image processing operations with only a local view on the image data allow a much better way of parallelization then post-processing operations, which are less or not parallelizable. Hence, local operations should be prefered, if possible, to ensure an efficient realization on a multi-core architecture in an embedded image processing system. Therefore, we have shown how some global image operations can be solved with only local operators. This concept is called *Marching Pixels* and was first introduced in (Fey & Schmidt, 2005). It allows for example the centroid detection of multiple objects in an image which is required in industrial image processing (Fey et al., 2010). The disadvantage of this approach is that the processing has to be realized iteratively.

To parallelize local image processing operations, there exist several approaches. One possibility is the *partitioning* of the image and the parallel processing of the partitions which will be part of Section 4.2. A further approach is a *streaming* of image data together with an adapted parallelization which is the subject-matter of Section 4.3. Also a combination of both approaches is possible. Which type of parallelization should be established depends strongly on the application, the used multi-core architecture and the available on-chip memory.

[1] degree of parallelization
[2] which is called *mask*, *sliding window* or also *stencil*

4.2 Partitioning

A partitioning of an image can be used, if the internal memory of an embedded multi-core system is not large enough to store the complete image. A problem occurs, if an image is partitioned for calculation. For the processing of an image pixel, a specific number of adjacent neighbours, in dependence of the stencil size, is required. For the processing of a partition boundary, additional pixels have to be loaded in the internal memory. The additional required area of these pixels is called *ghostzone* and is illustrated with waved lines in Figure 3. There are two ways for a parallel processing of partitions (Figures 3(a) and 3(b)).



(a) Parallel processing of one partition

(b) Independent parallel processing of several sub-partitions

Fig. 3. Image partitioning approaches

A partition could be loaded in the internal memory, shared for the different cores of a multi-core architecture, and this partition is processed in parallel by several cores as illustrated in Figure 3(a). The disadvantage is, that adjacent cores require image pixels from each other. This can be solved with a shared memory or a communication over a common bus system. In the second approach shown in Figure 3(b), every core gets a sub-partition with its own ghostzone area. Hence, no communication or data sharing is required but the overhead for storing ghostzone pixels is greater and more internal memory is required. If the communication overhead between the processor cores is smaller than the loading overhead for additional ghostzone pixels, then the first approach should be preferred. This is the case in closely coupled cores like fine-granular processor arrays for example.

The partitioning should be realized in squared regions. They are optimal with regard to the relationship between the partition area and the overhead for the ghostzone area. In (Reichenbach et al., 2011), we presented the partitioning schemes in more detail and developed an analytical model. The goal was to find an optimal set of system parameters depending on application constraints, to achieve a load balancing between a multi-core processor and an external memory or bus system. We presented a so called *Adapted Roofline Model* for embedded application-specific multi-core systems which was closely modeled on

the *Roofline Model* (Williams et al., 2009) for standard multi-core processors. Our adapted model is illustrated in Figure 4.

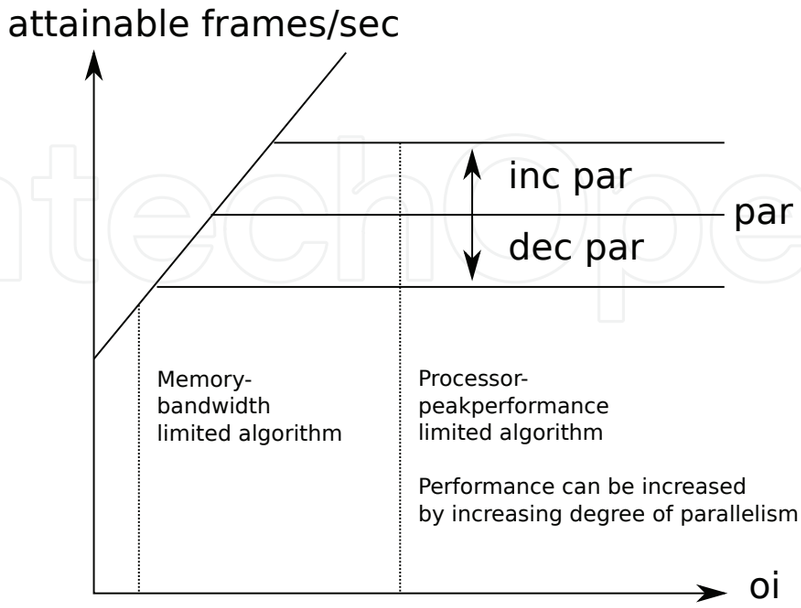


Fig. 4. Adapted roofline model

It shows the relationship between the processor performance and the external memory bandwidth. The horizontal axis reflects the operational intensity *oi* which is the number of operations applied to a loaded byte and is given by the image processing operation. The vertical axis reflects the achievable performance in frames per second. The horizontal curves with parameter *par* represent the multi-core processor performance for a specific degree of parallelization and the diagonal curve represents the limitation by the external memory bandwidth. Algorithms with a low operational intensity are commonly memory bandwidth limited. Only a few operations per loaded byte have to be performed per time slot and so the processor cores are often idle until new data is available. On the other hand, algorithms with a high operational intensity are limited by the peak performance of the processor. This means, there is enough data available per time step but the processor cores are working to capacity. In these cases, the achievable performance depends on the number of cores, i.e. the degree of parallelization. The points of intersection between the diagonal curve and the horizontal curves are optimal because there is an equal load balancing between processor performance and external memory bandwidth.

In a standard multi-core system, the degree of parallelization is fixed and the performance can be only improved with specific architecture features, like SIMD units or by exploitation of cache effects for example. In an application-specific multi-core system this is not necessarily the case. It is possible that the degree of parallelization can be chosen, for example if Soft-IP processors are used for FPGAs or for the development of ASICs. Hence, the degree of parallelization can be chosen optimally, depending on the available external memory bandwidth. In (Reichenbach et al., 2011) we have also shown how the operational intensity of an image processing algorithm can be influenced. As already mentioned, the Marching Pixel algorithms are iterative approaches. There exist also iterative image pre-processing operations like the *skeletonization* for example. All these iterative mask algorithms are known as iterative

stencil loops (ISL). By increasing the ghostzone width for these algorithms, it is possible to process several iterations for one loaded partition. This means, the operations per loaded byte can be increased. A higher operational intensity leads to a better utilization of the external memory bandwidth. Hence, the degree of parallelization can be increased until an equal load balancing is achieved which leads to an increased performance.

Such analytical models, like our Adapted Roofline Model, are not only capable for the optimized development of new application-specific architectures. They can also be used to analyze existing systems to find bottlenecks in the processing chain. In previous work, we developed an multi-core SoC for solving ISL algorithms which is called *ParCA* (Reichenbach et al., 2010). With the Adapted Roofline Model, we identified a bottleneck in the processing chain of this architecture, because the ghostzone width was not taken into account during the development of the architecture. By using an analytical model based on the constraints of the application, the system parameters like the degree of parallelization can be determined optimally, before an application-specific architecture is developed.

In conclusion, the partitioning can be used, if an image cannot be stored completely in the internal memory of a multi-core architecture. Because of the ghostzone, a data sharing is required if an image is partitioned for processing. If the cores of a processor are closely coupled, a partition should be processed in parallel by several cores. Otherwise, several sub-partitions with additional ghostzone pixels should be distributed to the processor cores. The partition size has to be chosen by means of the available internal memory and the used partition approach. If an application-specific multi-core system is developed, an analytical model based on the application constraints should be used to determine optimal system parameters like the degree of parallelization in relationship to the external memory bandwidth.

4.3 Streaming

Whenever possible, a streaming of the image data for the processing of local image processing operations should be preferred. The reason is, that a streaming approach is optimal relating to the required external memory accesses. The concept is presented in Figure 5.

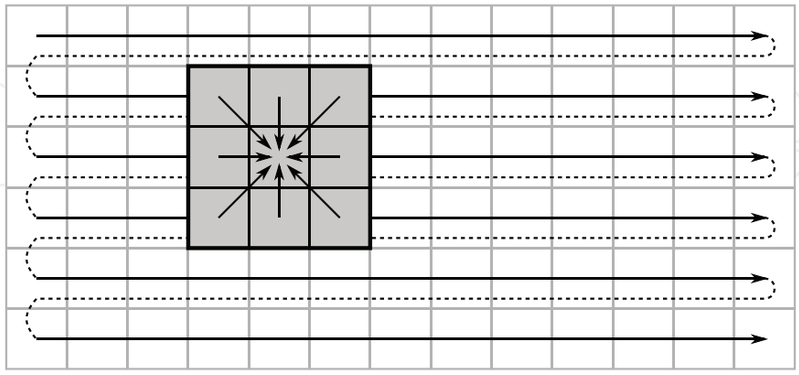


Fig. 5. Streaming approach

The image is processed from the upper left to the lower right corner for example. The internal memory is arranged as a large shift register to store several image lines. A processor core has access to the required pixels of the mask. The size of the shift register depends on the image

size and the stencil size. For a 3×3 mask, two complete image lines and three pixels have to be buffered internally. The image pixels are loaded from the external memory and stored in the shift register. If the shift register is filled, then in every clock cycle a pixel can be processed by the stencil operation from a processor core, all pixels are shifted to the next position and the next image pixel is stored in the shift register. Hence, every pixel of the image has to be loaded only once during processing. This concept is also know as *Full Buffering*.

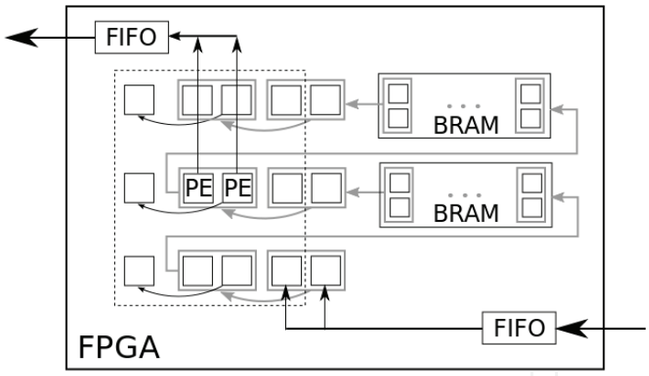
Strictly speaking, the streaming approach is also a kind of partitioning in image lines. But this approach requires a specially arranged internal memory which does not allow a random access to the memory as to the cache of a standard multi-core processor. Furthermore, a strict synchronization between the processor cores is required. Therefore, the streaming is presented separately. Nevertheless, this concept can be emulated with standard multi-core processors by consistent exploitation of cache blocking strategies as used in (Nguyen et al., 2010) for example.

In (Schmidt et al., 2011) we have shown that the Full Buffering can be used efficiently for a parallel processing with a multi-core architecture. We developed a generic VHDL model for the realization of this concept on a FPGA or an application-specific SoC. The architecture is illustrated for a FPGA solution with different degrees of parallelization in Figure 6. The processor cores are designated as *PE*. They have access to all relevant pixel registers required for the stencil operation. The shift registers are realized with internal dual-port Block RAM modules to save common resources of the FPGA. For a parallel processing of the image data stream, the number of shifted pixels per time step depends on the degree of parallelization. It can be adapted depending on the available external memory bandwidth to achieve a load balancing. Besides the degree of parallelization as parameter for the template, the image size, the bits per image pixel and also the pipeline depth can be chosen. The Full Buffering concept allows a pipelining of several Full Buffering stages and can be used for iterative approaches or for the consecutively processing of several image pre-processing operations. The pipelining is illustrated in Figure 7. The result pixels of a stage are not stored back in the external memory, but are fetched by the next stage. This is only possible, because there are no redundant memory accesses to image pixels when Full Buffering is used.

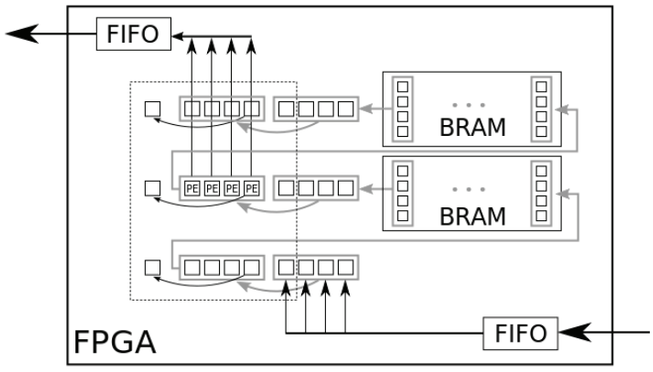
Depending on the stencil size, the required internal memory for a Full Buffering approach can be too large. But instead of using a partitioning, as presented before, a combination of both approaches is also possible. This means, the image is partitioned and a Full Buffering is applied for all partitions consecutively. For this approach, a partitioning of the image in stripes is the most promising. As already mentioned before, the used approach depends on the application constraints, the used multi-core architecture and the available on-chip memory. We currently expand the analytical model from (Reichenbach et al., 2011) in order that all cases are covered. Then it will be possible to predict the optimal processing scheme, for a given set of system parameters.

4.4 Image processing pipeline

In order to realize a complete image processing pipeline, it is possible to combine a streaming approach with an multi-core architecture for image recognition operations. Because the Marching Pixel approaches are highly iterative, we developed an ASIC architecture with a processor array fitted to the requirements of this special class of algorithms. The experiences from the *ParCA* architecture (Reichenbach et al., 2010) has gone into the development process



(a) Degree of parallelization p=2



(b) Degree of parallelization p=4

Fig. 6. Generic Full Buffering template for streaming applications

to improve the architecture concept and a new ASIC was developed (Loos et al., 2011). Because an image has to be enhanced, e.g. with a noise reduction, before the Marching Pixel algorithms can be performed efficiently, it is sensible to combine the ASIC with a streaming architecture for image pre-processing operations. An appropriate pipeline architecture was presented in (Schmidt et al., 2011). Instead of a application-specific multi-core architecture for image recognition operations, also a standard multi-core processor like ARM-Cortex A9-MP or the ECE-64 (see Chapter 3) can be used.

In this subchapter we pursued the question which data access patterns can be efficiently used in embedded multi-core processors for memory bound data parallel applications. Since many HPC applications are memory bound, too, the presented schemes can also be profitably used in HPC applications. This leads us to the general question of convergence between embedded computing and HPC which we want to discuss conclusively.

5. Convergence of parallel embedded computing and high performance computing

Currently a lot of people are talking of Green IT. Even if some think this is nothing else like another buzzword, we are convinced that all computer architects have the responsibility for future generations to think of energy-aware processor architectures intensively. In the past

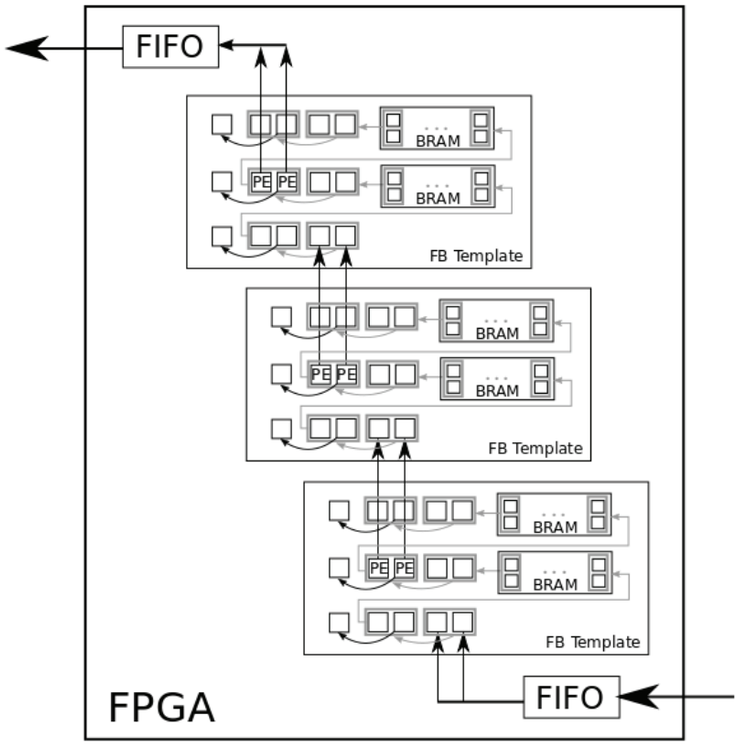


Fig. 7. Pipelining of Full Buffering stages

this was not valid in particular for the HPC community for which achieving the highest performance was the primary goal first of all. However, increasing energy costs, which cannot be ignored anymore, initiated a process of rethinking which could be the beginning of a convergence between methods used in HPC and in embedded computing design.

Therefore, one of the driving forces why such a convergence will probably take place is that the HPC community can learn from the embedded community how to design energy-saving architectures. But this is not only an one-sided process. Vice versa the embedded community can learn from the HPC community how to use efficiently methods and tools for parallel processing since the embedded community requires, besides power efficient solutions, more and more increasing performance. As we have shown above, this leaded to the introduction of multi-core technology in embedded processors. In this section, we want to point out arguments that speak for an adaptation of embedded computing methods in HPC(5.1) and vice versa (5.2). Finally we will take a brief look to the further development in this context (5.3).

5.1 Adaptation of embedded computing methods in HPC

If we consider a simple comparison of the achievable flop per expended watt, we see a clear advantage on the side of embedded processors (see Table 1). Shalf concludes in this context far-reaching consequences (Shalf, 2007). He says considering metrics like performance per power, not multi-core but many-core is even the answer. A moderate switching from single core and serial programs to modestly parallel computing will make programming much more difficult without receiving a corresponding award of a better performance-power ratio for this

- Power5 (server)
  - 389 mm^2
  - 120W@1900MHz
- Intel Core2 sc (laptop)
  - 130 mm^2
  - 15W@1000MHz
- ARM Cortex A8 (automobiles)
  - 5 mm^2
  - 0.8W@800MHz
- Tensilica DP (cell phones / printers)
  - 0.8 mm^2
  - 0.09W@600MHz
- Tensilica Xtensa (Cisco router)
  - 0.32 mm^2
  - 0.05W@600MHz

Table 1. Sizes and power dissipation of different CPU cores (Shalf, 2007)

effort. Instead he propagates the transition to many-core solutions based on simpler cores running at modestly lower clock frequencies. A loss of computational efficiency one suffers by moving from a more complex core to a much simpler core is manifoldly compensated by the enormous benefits one saves in power consumption and chip area. Borkar (Borkar, 2007) supports this statement and supplements that a mid- or maybe long-term shift to many-core can also be justified by an inverse application of Pollack's rule (Pollack, n.d.). This says that cutting a larger processor in halves of smaller processor cores means a decrease in computing performance of 70% in one core compared to the larger processor. However, since we have two cores now, we achieve a performance increase of 40% compared to the larger single core processor.

However, one has to note that shifting to many-core processors will not ease programmer's life in general. Particularly task parallel applications will sometimes not profit from 100s of cores at all due to limited parallelism in their inherent algorithm structure. Amdahl's law (Amdahl, 1967) will limit the speed-up to the serial fraction in the algorithm. The situation is different for data parallel tasks. Applying template and pipeline processing for memory bound applications in embedded computing, as we have shown it in Section 4, supports both ease of programming and exploiting the compute power given in many simpler cores. Doubtless, the embedded community has the most experience concerning power efficient design concepts which are now adapted from the HPC community and it is to expect that this trend will increase further. Examples that prove this statement can be seen already in practice. E.g. we will find processor cores in the design of the BlueGene (Gara et al., 2005) and SiCortex (Goodhue, 2009) supercomputers that are typically for embedded environments.

5.2 Adaptation of HPC methods in embedded computing

In the past the primary goal of the embedded computing industry was to improve the battery life, to reduce design costs and to bring the embedded product as soon as possible to market. It was easier to achieve these goals by designing simpler lower-frequency cores. Nevertheless, in the past the embedded community took over processor technologies like super scalar

units and out-of-order processing in their designs. This trend goes on. Massively parallel concepts which are typically for HPC applications are introduced in mainstream embedded applications. Shalf mentions in this context the Metro chip, which is the heart of Cisco's CRS-1 router. This router contains 188 general-purpose Tensilica cores (Ten, 2009). These programmable devices replaced Application Specific Integrated Circuits (ASICs) which were in that router in use before (Eatherton, 2005).

5.3 How the convergence will proceed?

Some experts expect that more and more the CPUs in future HPC systems will consist of embedded-like programmable cores combined with custom circuits, e.g. memory controllers, floating point units, and DSP cores for acceleration of specific tasks. Four years ago, Shalf predicted already that we will realize 2000 cores on one chip in 2011, a number closely to the number of transistors in the first Intel CPU 4004. We know now that this not happened. Possibly the time scaling for that predicted progress is longer than it was expected in the euphoria that came up in the first years when the multi-core/many-core era started. It is still possible that design processes change dramatically in the sense that Tensilica's CTO Chris Rowen is right when he says, "*The processor is the new transistor*". Definitely the two worlds, embedded parallel computing and HPC, which had been separated in the past, converged and it is exciting to see in the future where the journey will exactly end.

6. Conclusion

In this chapter we emphasized the importance of multi-core processing in embedded computing systems. We distinguished parallel applications between task vs. data parallel applications. Even if more task parallel applications can be found in embedded systems, data parallelism is a quite valuable application field as well if we think of image processing tasks. We pointed out by the development of the embedded ARM processor families and the ECA-64 architecture, which is in particular appropriate for data-parallel applications, that hierarchical and heterogeneous processors are pioneering for future parallel embedded processors. Heterogeneous processors will rule the future since they combine well-tailored performance cores for specific application with energy-aware computing.

However, it is a challenge to support data parallel applications for embedded systems by an efficient memory management. On the one side, standard multi-core architectures can be used. But they are not necessarily optimal in relationship to the available external memory bandwidth and, therefore, to the achievable throughput. By using application-specific architectures, an embedded multi-core system can be optimized, e.g. for throughput. The drawback of this is the increased development time for the system. As shown for image processing as field of application, a lot of constraints must be considered. The system parameters have to be chosen carefully, in order to avoid bottlenecks in the processing chain. A model for a specific class of applications, like presented in (Reichenbach et al., 2011), can help to optimize the set of parameters for the embedded system.

In addition the presented memory management system can also be exploited for memory bound data parallel applications in HPC. Anyway there is to observe that both worlds have learned from each other and we expect that this trend will continue. To strengthen this statement we pointed out different examples.

7. References

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities, *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), ACM, New York, NY, USA, pp. 483–485.
URL: *http://doi.acm.org/10.1145/1465482.1465560*

ARM (2007). *The ARM Cortex-A9 Processors*.
URL: *http://www.arm.com/pdfs/ARMCortexA-9Processor.pdf*

Blake, G., Dreslinski, R. G. & Mudge, T. (2009). A survey of multicore processors, *Signal Processing Magazine, IEEE* 26(6): 26–37.
URL: *http://dx.doi.org/10.1109/MSP.2009.934110*

Borkar, S. (2007). Thousand core chips: a technology perspective, *Proceedings of the 44th annual Design Automation Conference*, DAC '07, ACM, New York, NY, USA, pp. 746–749.
URL: *http://doi.acm.org/10.1145/1278480.1278667*

Bräunl, T. (2001). *Parallel Image Processing*, Springer-Verlag Berlin Heidelberg New York.

Eatherton, W. (2005). The push of network processing to the top of the pyramid, *in: Keynote Presentation at Proceedings ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, Princeton, NJ.

Ele (2008). *Element CXI Product Brief ECA-64 elemental computing array*.
URL: *http://www.elementcxi.com/downloads/ECA64ProductBrief.doc*

Fey, D. & Schmidt, D. (2005). Marching pixels: A new organic computing principle for high speed cmos camera chips, *Proceeding of the ACM*, pp. 1–9.

Fey et al., D. (2010). Realizing real-time centroid detection of multiple objects with marching pixels algorithms, *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops* pp. 98–107.

Gara, A., Blumrich, M. A., Chen, D., Chiu, G. L. T., Coteus, P., Giampapa, M. E., Haring, R. A., Heidelberger, P., Hoenicke, D., Kopcsay, G. V. & et al. (2005). Overview of the blue gene/l system architecture, *IBM Journal of Research and Development* 49(2): 195–212.
URL: *http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5388794*

Goodhue, J. (2009). Sicortex high-productivity, low-power computers, *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, IEEE Computer Society, Washington, DC, USA, pp. 1–.
URL: *http://dl.acm.org/citation.cfm?id=1586640.1587482*

Loos, A., Reichenbach, M. & Fey, D. (2011). Asic architecture to determine object centroids from gray-scale images using marching pixels, *Processings of the International Conference for Advances in Wireless, Mobile Networks and Applications*, Dubai, pp. 234–249.

Mattson, T., Sanders, B. & Massingill, B. (2004). *Patterns for Parallel Programming*, 1st edn, Addison-Wesley Professional.

Nguyen, A., Satish, N., Chhugani, J., Kim, C. & Dubey, P. (2010). 3.5-d blocking optimization for stencil computations on modern cpus and gpus, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, IEEE Computer Society, Washington, DC, USA, pp. 1–13.

Pollack, F. (n.d.). Pollack's Rule of Thumb for Microprocessor Performance and Area.
URL: *http://en.wikipedia.org/wiki/Pollack's_Rule*

Reichenbach et al., M. (2010). Design of a programmable architecture for cellular automata based image processing for smart camera chips, *Proceedings of the ADPC*, pp. A58–A63.

Reichenbach, M., Schmidt, M. & Fey, D. (2011). Analytical model for the optimization of self-organizing image processing systems utilizing cellular automata, *SORT 2011: 2nd IEEE Workshop on Self-Organizing Real-Time Systems*, Newport Beach, pp. 162–171.

Schmidt, M., Reichenbach, M., Loos, A. & Fey, D. (2011). A smart camera processing pipeline for image applications utilizing marching pixels, *Signal & Image Processing: An International Journal (SIPIJ)* Vol. 2(No. 3): 137–156.
URL: *http://airccse.org/journal/sipij/sipij.html*

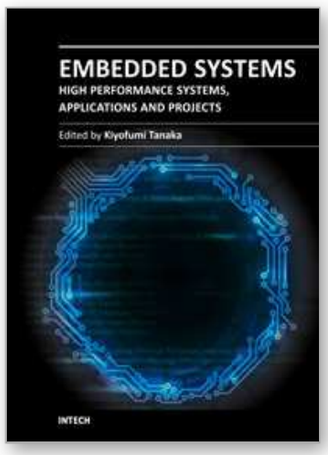
Shalf, J. (2007). The new landscape of parallel computer architecture, *Journal of Physics: Conference Series* 78(1): 012066.
URL: *http://stacks.iop.org/1742-6596/78/i=1/a=012066*

Stallings, W. (2006). *Computer Organization and Architecture - Designing for Performance (7. ed.)*, Pearson / Prentice Hall.

Ten (2009). *Configurable processors: What, why, how?*, *Tensilica Xtensa LX2 White Papers*. URL: *http://www.tensilica.com/products/literature-docs/white-papers/configurable-processors.htm*

Tex (2009). *OMAP4: Mobile applications plattform*.
URL: *http://focus.ti.com/lit/ml/swpt034/swpt034.pdf*

Williams, S., Waterman, A. & Patterson, D. (2009). Roofline: an insightful visual performance model for multicore architectures, *Commun. ACM* 52(4): 65–76.



Embedded Systems - High Performance Systems, Applications and Projects

Edited by Dr. Kiyofumi Tanaka

ISBN 978-953-51-0350-9
Hard cover, 278 pages
Publisher InTech
Published online 16, March, 2012
Published in print edition March, 2012

Nowadays, embedded systems - computer systems that are embedded in various kinds of devices and play an important role of specific control functions, have permeated various scenes of industry. Therefore, we can hardly discuss our life or society from now onwards without referring to embedded systems. For wide-ranging embedded systems to continue their growth, a number of high-quality fundamental and applied researches are indispensable. This book contains 13 excellent chapters and addresses a wide spectrum of research topics of embedded systems, including parallel computing, communication architecture, application-specific systems, and embedded systems projects. Embedded systems can be made only after fusing miscellaneous technologies together. Various technologies condensed in this book as well as in the complementary book "Embedded Systems - Theory and Design Methodology", will be helpful to researchers and engineers around the world.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Michael Schmidt, Dietmar Fey and Marc Reichenbach (2012). Parallel Embedded Computing Architectures, Embedded Systems - High Performance Systems, Applications and Projects, Dr. Kiyofumi Tanaka (Ed.), ISBN: 978-953-51-0350-9, InTech, Available from: http://www.intechopen.com/books/embedded-systems-high-performance-systems-applications-and-projects/parallel-embedded-computing-architectures

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen