Chapter #

CACHE-COHERENT HETEROGENEOUS MULTIPROCESSING AS BASIS FOR STREAMING APPLICATIONS

Subtitle

Jos van Eijndhoven, Jan Hoogerbrugge, Jayram Nageswaran,
Paul Stravers, Andrei Terechko

Philips Research Laboratories Eindhoven

Abstract:

New generation System-on-Chips will be extremely complex devices, composed from complex subsystems, relying on abstraction from implementation details. These chips will support the execution of a mix of concurrent applications that are not known in detail at chip design time. These SoCs require a significant degree of programmability to configure both the set of functions that must execute as well as the structure of the dataflow between these functions. To ease the programming effort multiprocessor computers have employed cache coherent share memory for decades, abstracting the average programmer from system complexity issues such as multiple processors and memory hierarchies.

Memory coherency in multiprocessor computers has a history of decades, and has proven to be an indispensable abstraction from system complexity towards the application programmer. This chapter describes a next generation SoC for the consumer electronics domain (e.g. audio/video, vision, robotics). It features heterogeneous multiprocessor subsystems with a snooping cache coherence protocol, combined in a system with distributed memory employing a directory coherency protocol. It is explained why and how the coherent memory model is indispensable for implementing both data transport and synchronization for multi-tasking streaming applications in distributed memory systems.

Keywords:System-on-Chip, multi-processor, cache coherency, streaming, memory hierarchy

1. INTRODUCTION

The semiconductor industry is facing enormous challenges with the creation and marketing of every new generation of System-on-Chip (SoC) in the consumer electronics market segment. These consumers want trendy products loaded with modern features, matched with their personal taste. However the creation of such systems today with hundreds of millions of transistors in hardware and tens of megabytes of embedded software is a task of daunting complexity. The design process goes through stages of specification, implementation and verification both for the hardware itself as for the embedded software, with contributions of multiple design groups spread over the world and over multiple companies. These processes take several tens to a few hundred man-years of effort, stretched over a few years of elapsed time for major new products. This trend is shown for instance in a 2003 analysis of IBS (Int. Business Strategies inc.), see Figure 1.

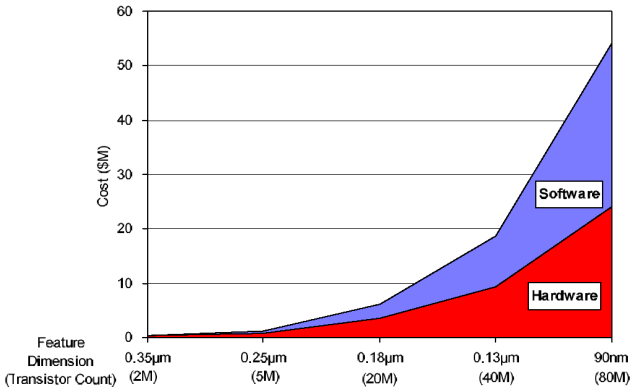

Figure 1 SoC development cost

In view of the fast changing and cost sensitive consumer market, such SoC products require a huge investment and carry high risk. This chapter describes the SoC architecture being created in the Philips Research 'CAKE' project [Str2001]. The project targets the consumer electronics media processing domain, aiming to address the SoC challenges mentioned above. The first implementation of this architecture is now in design. The following paragraphs describe a set of top-level design considerations that together form the key aspects of the CAKE architecture.

Programmability: Programmability is a key property that allows late changes in the product functionality, as well as adaptations to local market needs. Similarly, programmability can be regarded as an insurance cost to

reduce the investment risk of creating a silicon product with features that mismatch with new demands at the time the product reaches the market. Programmability also allows the re-use of the same silicon across different products, maybe created by different companies. This is an important aspect, as for domain-specific SoCs the silicon design cost will not be negligible in comparison with the production cost. If the chip supports an industry-standard programming model, functions can be created by re-use of standard software, which will give a tremendous reduction on system development cost and time. The main factor that opposes programmability is power consumption, leading to more specialized engines (or coprocessors) for some functions.

Parallel processing: The employment of multiple small processors to match the large computation workload as represented by today's streaming media processing, will be beneficial both for the silicon area as well as for the power consumption. The micro-architecture of current high-end microprocessors shows a clear problem of diminishing returns [Hen2003] [Die2003]. Our targeted products will typically have a system load consisting of a mixture of concurrently active tasks to occupy multiple CPU's in parallel. Furthermore, according to our experience, the audio and video processing algorithms easily allow an additional (lower) layer of thread-level parallelism by operating on multiple blocks of data in parallel, if this were needed to obtain sufficient spreading of CPU load. The use of multi-threading applications is becoming more-and-more accepted as generic programming model, as the trend towards chip-multiprocessing and CPU's with multi-thread facilities is picked up by all major processor developers [Hal2004]. To support re-use of industry-standard software, the CAKE architecture creates a global uniform and coherent memory view towards its processors, in accordance with the general-purpose computing world.

Tiling: Tiling creates an extra hierarchy layer in the system hardware architecture. The top-level architecture view shows a regular structure of homogenous tiles (subsystems) connected through (for instance) a two-dimensional torus network (see Figure 2) (For more information see the section on 'Static Networks' in [Fly1995], or [Dal2001]). Such architecture allows a trivial instantiation of both large (expensive) and small (cheap) products with scalable compute performance for little silicon design effort. Tiles can simply be replicated in layout, without the need for extensive verification per product instantiation.

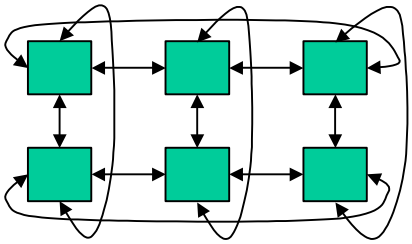

Figure 2 Tiling with a torus network

Torus structures are trivially mapped on silicon without long wires, creating predictable and scalable performance. If torus links are made available off-chip, then current chips can serve to create a prototype for next-generation silicon, to perform prototype software mapping and performance analysis. Inside a tile, a subsystem is created with a heterogeneous set of processors and an embedded memory, in complexity comparable with today's large SoCs. Heterogeneous processors allow a choice of processors for different types of tasks, thereby improving the computational efficiency (energy required to execute the task). The embedded memory is needed to ensure that most memory accesses of the processors can be served by tile-local memory, which avoids the access latency induced by the inter-tile network or off-chip access. The network latency would otherwise kill processor performance. Furthermore, having most data locally in the tile reduces power consumption for data transport, and saves precious bandwidth to off-chip bulk memory. With the growth of on-chip aggregate processor performance, it is almost inevitable that total on-chip memory needs to grow in capacity, taking an increasing percentage of chip area [Nai2002]. Proper use of streaming programming models for our media processing target domain, and support for tile-local stream-buffer allocation and stream-management over the inter-tile network, will help to fight-back the growing on-chip memory needs and thereby have more effective compute resources on given silicon area. Automatic prefetching of streaming data into the cache will help to hide the latency of the memory hierarchy with little effort for the application programmer [Wae2005].

Redundancy and self-test: With SoC complexities now in few hundred millions of transistors, breaking the one billion transistor mark soon, it becomes unrealistic to expect that after fabrication every individual transistor will indeed work correctly as expected. Aiming for perfect products would result in an uneconomical loss after fabrication and test. The CAKE architecture advocates multiple identical processors and memory blocks in every tile, and multiple identical tiles on chip. Clearly, if one processor out of ten (or a hundred) would be defect, the chip can still serve a quite useful computational load. Also if a one-megabit memory instance is faulty (after trying to exploit its built-in redundancy), the remaining memory

blocks could still support useful program execution. Similarly, at the top-level architecture, an entire faulty tile might not block overall use. Only a very small percentage of the chip area is really unique and indispensable, so there is only an extremely small chance that the chip will be really dead. A chip that is loaded with identical programmable compute facilities will have the intelligence and processing power to perform self-test at every cold boot-up of the system, and can configure itself to avoid the use of its faulty parts. This allows the semiconductor vendor to sell perfect products for a premium price, and sell the other products for price-sensitive applications with hardly any production loss. The consumer could observe a small degradation of its appliance over time, and buy new equipment before a total break down occurs. In that sense, consumer electronic devices would 'wear out', and could be treated with an attitude similar as clothes, furniture, and cars.

Creating chip multi-processors (CMPs) is a rapidly growing trend in industry: the trends as sketched above have broad applicability. Both old and modern examples are too numerous to list here [Hal2004]. A few nice examples that target the embedded market are the 4-core ARM module with L1-cache snooping [Kre2004], a 4-core PowerPC embedded in an FPGA [Kow2003], a 4-core MIPS with shared L2 cache and off-chip links for PCB-level tiling [Won2002], and of course the Sony/IBM Cell architecture that also advocates tiling for scalability (publications regarding its architecture are expected during 2005) [Suz2002].

The first test-chip that is currently designed according to the CAKE architecture, will contain a single tile only. Off-chip links allow the creation of a multi-tile system at PCB (printed circuit board) level. The on-chip (tile) memory is currently designed as a shared level-2 cache, which allows flexible use through software configuration. For processors, a recent version of Philips' TriMedia processor is used [Wae2005]. The TriMedia processor is optimized for audio and video media processing, and features a high computational density and power efficiency in its application domain.

Section 2 will provide more information on the CAKE architecture, in particular regarding the tile-local network and its cache coherency. Section 3 will describe approaches towards parallel programming, and shows some benchmarking results. Section 4 summarizes the current state of the project.

2. CAKE TILE ARCHITECTURE

The CAKE tile architecture comprises of multiple CPUs, various IP blocks, a shared L2 cache, and the interconnect network, see Figure 3 below.

IP blocks may be simple coprocessors (e.g. image enhancements, video scalars, MPEG decoders, etc.), programmable processors, or complete subsystems with multiple processing units and on-chip memory.

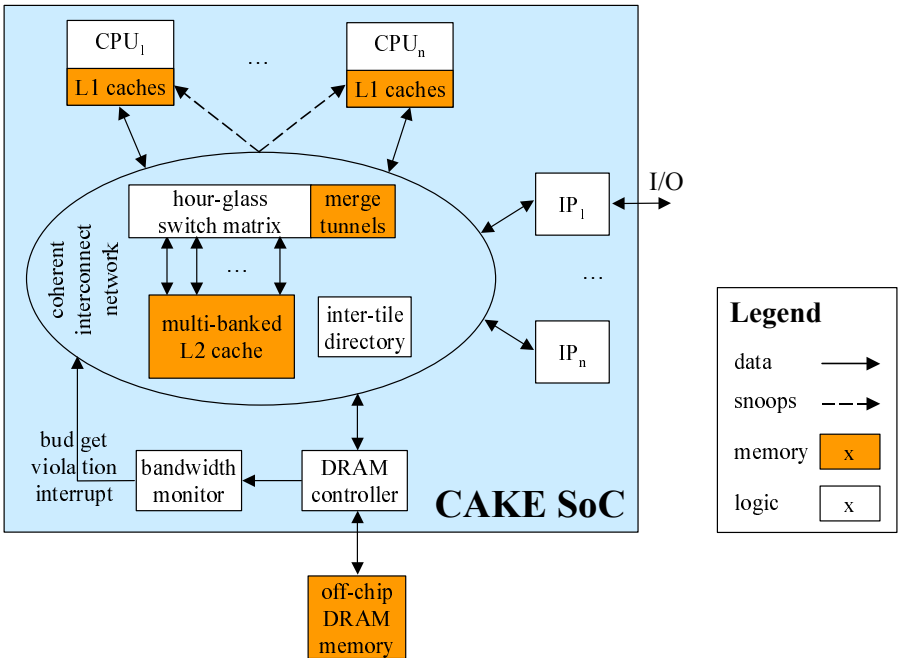

Figure 3 **CAKE Tile architecture**

CAKE's tile-local *coherent interconnect network* is capable of serving multiple concurrent transactions from the attached L1 caches and coprocessors. These transactions follow Philip's 'MTL' or ARM's 'AXI' protocols for memory requests, typically transferring a sequence of data-words per request. The tile network supports multiple outstanding request per port, with out-of-order completion. The block-transfers are supported with 'critical word first' delivery to reduce the CPU stall time (cache miss penalty). The interconnect network is created by a set of 'hour glass'[1] structures between the various groups of components. The 'hour glass' structures provide sufficient parallel data transfers to sustain processor performance, without unnecessary overhead of a fully-connected switch matrix. These 'groups of components' are (a) components that connect to the network:

[1] An 'hour glass' network provides connections between two sets of ports, through two switch matrices and intermediate set of '*n*' ports. This allows to dynamically configure paths between any of the two sets of ports, up to a maximum of '*n*' simultaneous paths.

– The various CPUs and IP modules (co-processors).
– DMA engines that create data transport through inter-tile links.
– Different ports of the DRAM controller.

and (b) components internal in the network:

– Multiple L2 cache banks, selected through address interleaving.
– Merge tunnels, used for direct (demand driven) inter-processor communication.

The main objective of this network is to provide a low-latency access from processors to their requested data. The RTL code for this interconnect network is generated from a set of parameters, to create different CAKE instances from the same code base. The chip that is currently in design will probably connect 9 TriMedias, one ARM or MIPS processor, 4 function-specific (video-) coprocessors, 4 PCI-Express interfaces, and 8 L2 cache banks, through 64-bit data paths. All processors and the tile-network are targeted to run at a 350MHz to 450MHz clock rate in a CMOS 65-nm process. (The achievable clock rate is strongly influenced by the SRAM performance, which is -at the time of writing- not accurately known for the targeted 65-nm process.)

To maintain sustained throughput, the interconnect and L2 control implements *hit-under-miss*. According to this policy the system does not block on an L2 miss. In fact, the interconnect keeps serving subsequent transaction(s) from other CPUs, in parallel with handling L2 refills. Furthermore, if the L2 miss is not caused by a demand load (but, for instance, a prefetch), the corresponding CPU does not get blocked either. To enable seamless integration of diverse CPUs running at various clock frequencies in respective islands of synchronicity, the interconnect network talks to the CPUs and IPs via *asynchronous clock domain bridges.* To avoid clock skew problems during back-end design, source-synchronous communication channels are employed to connect CPUs and IPs that are located further away from the central coherency controller and L2.

CAKE SoC interconnect network scales up to about 20 or 30 CPUs/IPs: beyond that the routing area and cache snooping traffic becomes harder to manage. To enable scalability beyond that we employ *tiling* with distributed shared memory (DSM) across multiple tiles (See for instance the section on “Scalable Multiprocessors” in [Fly1995]). Inter-tile communication is controlled by a directory-based coherence protocol with some support from firmware to handle remote misses. Each tile maintains a directory table. Each entry of the table specifies where the corresponding memory block resides (locally or remotely). By employing a cache snooping protocol inside a tile, and supporting a directory-based protocol between tiles, we believe the system is scalable to support beyond a hundred processors for the applications in our targeted domain (which in general have little data

dependency). In case of an L2 miss on a memory block from another tile, the hardware calls a firmware routine that orchestrates the inter-tile transfer. Such a transfer is typically carried out as message passing via DMA links and takes a few hundred cycles in our simulation model. The first CAKE chip will implement the inter-tile links as off-chip point-to-point PCI-Express channels. Using the global inter-tile coherent memory view is functionally transparent for the application programmer.

Compared to many prior-art embedded SoC designs [Oli2003][Intel] the CAKE SoC features the *L2 cache*, which boasts many advantages. First, it saves the off-chip DRAM bandwidth and associated power dissipation by serving many data accesses from L1 caches and coprocessors and keeping communication on-chip. Second, the L2 cache decreases the latency of memory accesses to a few dozens of cycles. Third, the L2 cache efficiently transfers data on and off the chip automatically in chunk sizes appropriate for the off-chip DDR, independent of smaller request sizes of the individual CPU's and coprocessors. This allows efficient use of DDR bandwidth, while re-using older (co-)processor designs that still employ smaller block sizes.

An SoC has multiple on-chip memories and CPUs with caches. Hence, the inevitable *cache coherence* problem [Hen2003]. For example:

a) If a process in CPU 'A' stores a value to a memory address, this updated value might remain inside A's cache for a while. If later a process on CPU 'B' wants to read the value in this address, it might miss in its cache and fetch an out-dated value from main memory.
b) A process in CPU 'A' has stored a new value to a memory address, and also (after a while) copied this data back to main memory. If a process on CPU 'B' wants to read this value, it might find a hit in its local cache and still use an out-dated value.

Note that this cache coherency problem even persists with a single process (or thread) without any inter-process communication: An SMP process scheduler might stop and later re-schedule the process on different CPU causing similar cache coherency problems.

The CAKE SoC architecture provides hardware cache coherence, exposing a simple shared memory model to the programmer. Our cache coherency is implemented using *snooping* in the tile interconnect, according to the standard MSI protocol outlined in Figure 4 below. Each L1 cache line can be found in one of the three states: Invalid, Shared or Modified. The state changes are caused by activities of the attached CPU (shown in solid lines) and by snoop requests that arrive from the other CPU's (shown in dashed lines). The cache coherence protocol ensures only a single exclusive copy of the cache line being modified. There can, however, be multiple copies of a cache line in the Shared state. (For more information, see the section "Memory Coherence in Shared Memory Multiprocessors" in [Fly1995].)

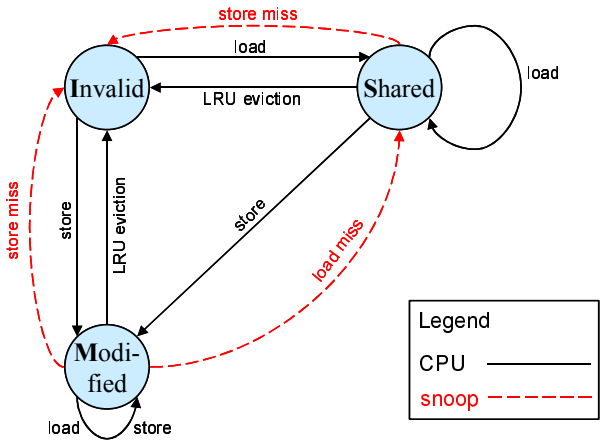

Figure 4 **MSI protocol state transitions**

The network broadcasts a snoop request upon receiving a transaction request from a CPU or an IP. Then the coherent CPUs reply to the requests with the status of the cache line in question. Our design can sustain a new snoop request every clock cycle. To have a snoop-answer within a few clock cycles, without disturbing the normal processing of the snooped CPU's, the L1 cache tags are duplicated into 'shadow tag' structures that are stored in the clock domain of the network (as opposed to the clock domains of the respective CPU's). Adding the copied shadow tags and snoop logic creates about 3% area overhead relative to a TriMedia CPU. For this small overhead the SoC obtains an industry-standard programming model, and time-consuming programmer effort is saved by avoiding complex SW porting issues and intriguing (non-reproducible) software bugs.

Snooped CPU's might need to start a *snoop action*, according to the MSI protocol, shown as dashed arrows in Figure 4. For example, if the snoop request asks the exclusive rights for a cache line and the line is found dirty in another CPU, then that other CPU performs the snoop action of write-back and invalidate of that line. Note that non-coherent CPUs and IPs also benefit from the coherent network, when they stream in data produced by the coherent CPU cluster. The *inter-CPU communication* (such as streaming data) is naturally realized by accessing the shared memory. First, the producer CPU creates data in its local L1 cache. Then, the consumer CPU asks for this data from the interconnect, which finds it in the L1 of the producer CPU through snooping. Finally, the interconnect network streams the data to the consumer CPU via a 'merge tunnel' (see Figure 3). The merge tunnel can combine several cache lines from different sources (L1, L2, DRAM), depending on the requested block size. The programmer does not have to explicitly program the data transport.

The L2 cache is currently targeted to have a capacity of 2 MByte, requiring about 16mm^2 for high-density SRAM. For high-definition video applications, embedded DRAM is an attractive option, allowing significantly larger capacity in the same silicon area. Unfortunately, availability of embedded DRAM is -at the moment of this writing- uncertain at our targeted tape-out date, but can be reconsidered for a later product. The L2 implementation furthermore requires a tag storage of 0.25mm^2 and negligible cache control logic. These area figures are to be considered relative to the CPU area, which is roughly 3mm^2 for each TriMedia or ARM processor, depending on instantiated CPU variation and L1 cache parameters.

Task *synchronization* heavily relies on coherency too. The coherent CPUs may rely on two well-known operations LL (Load Linked) and SC (Store Conditional) [Hen2003] to perform synchronization without stalling the memory subsystem. In particular, these operations allow to easily create memory-mapped semaphores. The implementation of the LL/SC operations relies on cache coherency, which provides the LL operation with the freshest data without interfering with other CPUs. SC in turn consults the cache line status and reuses the snooping mechanism to complete the atomic LL/SC pair. These two operations have been added to the TriMedia instruction set, to support efficient inter-thread synchronization and easy porting of external software that relies on memory-mapped semaphores.

CAKE SoC enables a predictable *compositional* system design through advanced resource management. The resource management ensures that integration and use of many software and hardware components that share resources (caches, DRAM bandwidth, CPU cycles) does not affect the expected performance of the individual components, or at least protects the performance of critical components by proper management. The resource management in CAKE includes DRAM bandwidth and *cache footprint management* by explicit partitioning of the cache space [Ote2005] [Mol2005]. All data transfers from the CPUs are accompanied by the task id, which selects a way-mask for the L2 cache. This way-mask protects a subset of the ways for victim assignment upon a cache miss. In other words, a L2 cache refill on behalf of one task can be prevented to evict a cache line of another critical task. Hence, task interference in the L2 cache is minimized. Note that cache hit detection uses all ways, thus enabling inter-task communication.

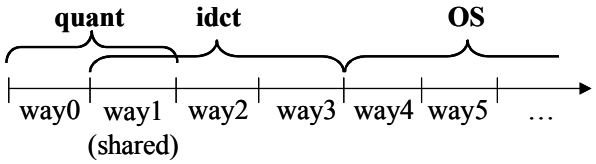

Figure 5 Way partitioning among the tasks

Task descriptors also identify the bandwidth domain the task belongs to. CAKE SoC provides bandwidth monitor registers keeping track of the *DRAM bandwidth* utilization per domain. When a domain has exhausted its allocated bandwidth budget, this is signaled to the (software) Quality of Service manager. The Quality of Service manager can decide to scale down the service of the violating domain or to leave its service level unaffected if the domain is meeting its deadlines. Task sharing of the DRAM bandwidth is limited by the respective bandwidth budgets controlled by dedicated bandwidth monitor registers. The hardware facilities to allow bandwidth and cache space control are now in detailed design, the management methods are clear [Ote2005], but actual verification of the overall system behavior and its tuning has still to be performed.

3. PROGRAMMING METHODLOGY

Application programming models are highly evolving area of research [Lee2002] and each type of programming model is tailor-made to exploit certain properties, which are essential for a particular class of applications and the targeted hardware. The multiprocessor features in CAKE can be exploited by parallellizing a single application or running many applications concurrently and using the underlying cache coherent network for communication. Various types of programming models had been proposed which can be broadly classified into streaming models (detailed reference to various models in [Wol2004]) and non-streaming models like series-parallel graph, Finite State Machine (FSM).

Kahn Process Network (KPN) [Wol2004] and its variants are one class of programming models appropriate for modeling streaming multimedia and signal processing applications. KPN based models have simple and well defined interfaces for high-level abstract specification of the application and thus are suitable for running on heterogeneous systems. The simple interface provided by KPN based models facilitates reuse of the component in the model for different applications. Unfortunately, KPN based models might have unpredictable execution time, deadlocks, or overuse of resources like memory. Another method of expressing concurrency is POSIX thread or Pthread [Nic1998]. Pthread provides a number of low-level, low-overhead primitives supporting multithreading and flexible communication between threads. Pthreads assume that all threads share a uniform address space and the underlying architecture should support this shared memory concept. Pthreads support highly dynamic thread creation depending upon the

application requirements and is a widely used industry standard for shared memory machines. But Pthreads have a very low-level of abstraction and increases the burden on programmers for both task management and thread communication. Yet another model called SDF (Synchronous Data Flowgraph) [Sri2000] is popular within DSP community because of the useful property that deadlock and resource requirements are decidable or determinable. But not all class of applications can be efficiently expressed by means of SDF with reasonable effort.

Consequently we see that a platform such as CAKE to be widely usable across a range of application domain, the underlying architecture should efficiently run a wide variety of programming models. Currently the CAKE architecture can support parallel models like TSSA[Oli2003], C-heap, YAPI, TTL[Wol2004] and Pthreads. Restricting an architecture or platform to run only one kind of programming model like SDF seriously hinders wider acceptance of the platform, because other models will run inefficiently and thus shut the door to the proponents of other programming models. Also complex applications need the support of mixed programming models because different parts of the application might be expressed more naturally by different programming models [Lee2002]. These application modeling issues prompted the CAKE architecture to support a shared-memory concept both inside and over the tile, as well as explicit streaming for more efficient inter-tile communication. Also the cache-coherency eases the burden on programmer to bother about the location of the latest data in the system.

The application design trajectory or flow shown in Figure 7 starts by explicitly capturing the parallelism or expressing the parallelism possible in the application. CAKE architecture uses the Trimedia compiler, which is very good in extracting the Instruction Level Parallelism (ILP), but extracting higher levels of parallelism is either manual or semi-automatic. Once the parallelism possible in the application is analyzed it can be captured using some of the programming models like KPN, Pthreads etc mentioned in the previous paragraph.

It is widely known that parallelism can be extracted by functional-, data- or mixed partitioning. In a functional partitioned model each task (thread) performs a distinct function in a pipelined fashion and communication between tasks is made explicit (see Figure 7). A functional partitioned model

- Facilitates easy reuse of the functional modules for other similar applications.
- Intuitively fits with the streaming application description.

– Is an appropriate model for systems where some tasks are implemented on function-specific coprocessors.

The main disadvantages are:

– Possibility for load imbalanced partitioning where a single task limits the speedup.
– Inter-task communication might consume high bandwidth, since streaming data needs to travel between processors.
– Large human effort is required for conversion of a monolithic sequential application into a functional parallel application because all communication between the functional modules need to be made explicit. This effort in practice limits the amount of obtainable parallelism.

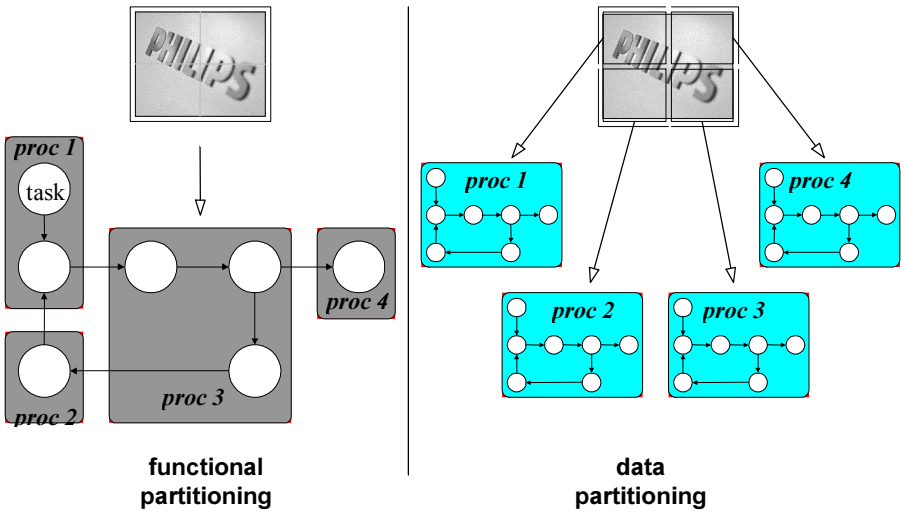

Figure 6 Functional versus Data Partitoning

On the other hand in data partitioned application models different input data is simultaneously processed by different threads executing the same function (see Figure 6). A data partitioned model has various advantages

– Good scalability: When the available number of processors changes than only the data distribution needs to change without changing the function implementation. Often, no code changes are needed at all, as the number of threads might be dynamically determined.
– Efficient usage of cache resources: The size of data partition can be selected to maximize data cache hit ratio.
– Allows natural load balancing depending upon the workload.

But also data parallelism extraction needs thorough study of each application and is algorithm specific. The exact scheme by which the data is partitioned and distributed to different tasks determines the performance. Still a major part of the data parallel models can be reused across various applications if within each data partitioned model some kind of modularity exists with a clean interface and encapsulation. So from our experiments we found out that data parallel models scale well for increasing or decreasing resources and also varying input streams resulting in natural load balancing and resource utilization. In practice a combination of functional and data parallellism will be applied. We foresee functional parallelism at course granularity, such as a video codec, an audio codec, or an image improvement algorithm. Inside those functions, multi-threaded data-level parallelism can be exploited if needed to achieve real-time throughput.

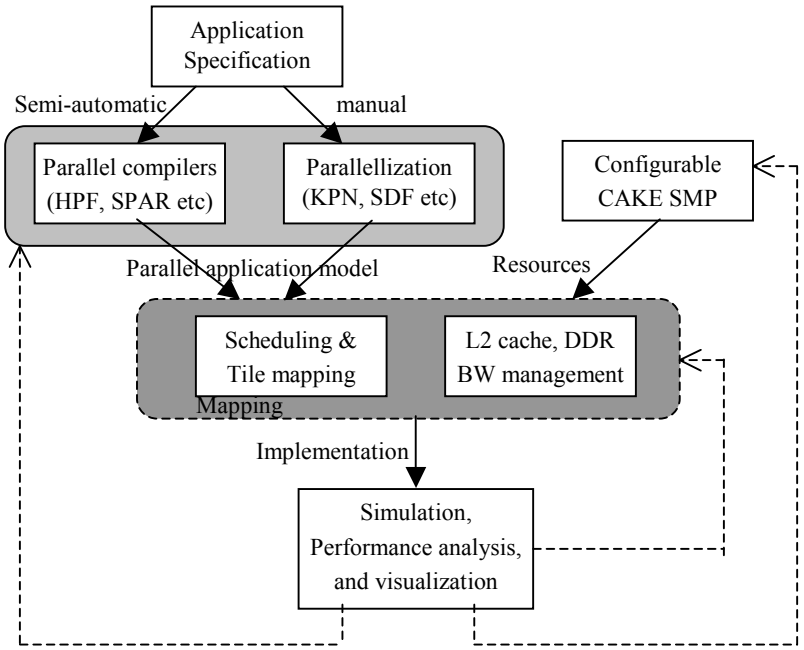

Figure 7: Application development trajectory

Once the parallel model of the application is developed, a mapping or binding process is carried out. Earlier the CAKE architecture was running a dedicated distributed, light-weight operating system kernel called the TRT (Tile RunTime) [Str2001], which supports fine-grain synchronization and fast context switching but lacked real-time support. Recently we successfully ported the SMP (Symmetric Multi Processing) version of the open-source, embedded real-time, operating system eCos [Mas2002]. Currently, the

CAKE software implements static task partitioning across the tiles and dynamic task scheduling inside the tiles. But other kinds of scheduling like static or quasi-static [Sri2000] etc can also be implemented, since the thread-scheduler is a well-separated functionality of the eCos system. Many applications where real-time constraints are not involved, dynamic scheduling offers the best usage of available CPU resources with quickest possible design integration. Other applications, where real-time deadlines and throughput constraints are more difficult to meet, applications need to be tuned for reserving shared resources (CPU cycles, DDR memory bandwidth, L2 cache footprint), improving the ILP (Instruction Level Parallellism) by source-code optimizations, and modifying the data or function thread-level parallellism. Even with a fully programmable SMP, sufficient real-time guarantees could still be achieved by means of efficient resource management provided by the CAKE architecture (see also [Ote2005]).

We mapped various applications like an MPEG-2 decoder [Str2001], an MPEG-2 encoder, 3D-TV rendering algorithms, an Open-GL 3D-GFX library, an H.264 decoder [Tol2003] and the SPLASH-2 benchmark [Woo1995], and found good scalability and performance on the CAKE architecture. Furthermore, the 'Archtest' program [Col1992] was mapped to thoroughly verify our memory consistency.

High-definition MPEG-2 decoding was easily parallellized by forking a new thread for the decoding of every slice. MPEG-2 ensures that slices can be decoded in parallel, and that slices span at most 16 video lines. As result, an HD image (1920x1080 pixels) can be processed with 68 threads in a data-parallel mode with very minor code changes.

Figure 8 shows the performance of the CAKE architecture for the SPLASH-2 benchmark suite. The simulations were performed with different numbers of TriMedia processors, that share an 8-bank 12MB L2 cache. The individual processors were configured with 16KB of L1 data cache with 128-byte line size. The benchmark code was run straight out-of-the-box, without any adaptation. We can see from the Figure 8 that even without any optimization the benchmark scales well. Further study is still needed to evaluate the details of these results.

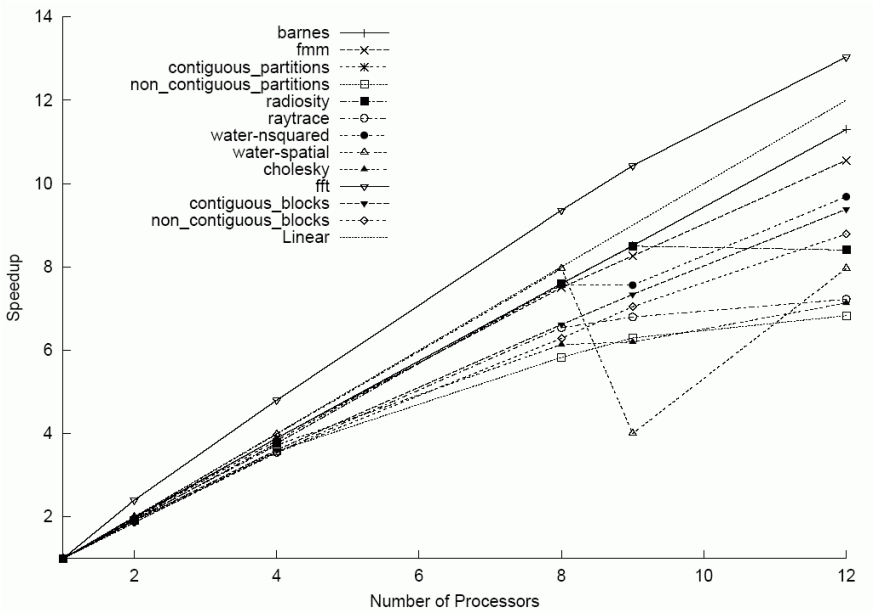

Figure 8 Out-of-the-box SPLASH-2 benchmark results

Clearly, many possible systems can be created with mixing HW architectural features with programming paradigms. In our view however, two particular combinations seem to fit naturally together:

a) Homogeneous multi-processors, in a shared-memory architecture, employing data-parallellism, and relying on dynamic task-to-CPU mapping.
b) Heterogeneous multi-processors, in a message-passing architecture, employing function-level parallellsim, using static task mapping.

Philips' SoCs for the embedded consumer-electronics (CE) domain have traditionally used the latter (b) style, mainly because of the high efficiency of function-specific processors. The (a) style has become prominent in general-purpose computing because of its flexibility and ease of programming. With the growing transistor-count per die, and the growing amounts of embedded software, we foresee a growing trend towards (a) also for the embedded CE market.

4. STATUS AND CONLUSIONS

The CAKE project has a tool suite up and running, partially consisting of standard tools, partially newly created software. The complete suite operates in a Linux environment and contains:

- A TriMedia compiler chain, a MIPS compiler, and a shared-object linker.
- Embedded software for a low-level boot-up of the chip, and the eCos embedded OS with our own HAL (hardware abstraction layer) for support of timers, interrupt controllers, and other device drivers.
- Application software to be executed by the embedded processors for:
  a) Testing correct behavior of the hardware. Besides basic CPU tests, multi-processors data exchange test are used, such as the generic 'Archtest' program [Col1992].
  b) Media processing application software that is used for performance measurements.
- The parameterized simulator with an almost cycle-accurate system model, built on top of the SystemC simulation kernel.
- Several (interactive graphical) tools for evaluating the simulated system performance based on off-line visualization of generated trace files.

Currently RTL (Verilog) code is being made for a first test-chip that implements a single tile of the CAKE architecture concepts. Individual Verilog modules are tested through co-simulation, embedded in the overall SystemC system model. Tape-out is expected in a CMOS 65-nm technology by the end of 2005. Creating this chip and its initial applications is done as a cooperative effort between Philips Research, Philips Semiconductors and Philips Consumer Electronics.

Parallel benchmarks like SPLASH-2 and Archtest did run on the CAKE platform without any modification. eCos itself was ported, including its optional libraries such as the Posix layer, with full multi-processing and real-time support, requiring only little effort for its platform-specific HAL. This proves that the project created a platform with industry-standard programmability, of which cache-coherency is an indispensable ingredient. As such, this platform paves the way for the software re-use as demanded by future embedded systems. The concepts and the architecture realized in this project will establish a strong basis for evolutionary product growth well into the next decade, reaching billions of transistors and hundreds of processors on a single die, in an economically sound way.

REFERENCES

[Col1992] W. W. Collier, *Reasoning about parallel architectures*, Prentice-Hall, 1992 (See also http://www.mpdiag.com/)

[Dal2001] W. J. Dally and B. Towless "Route Packets, Not Wires: On-chip Interconnection Networks", *proc. DAC2001 38th Design Automation Conf.*, pp. 684-689, Las Vegas, USA, Jun. 2001

[Die2003] K. Diefendorff and Y. Duquesne, "Complex SoCs require new architectures", *EEdesign*, Sep. 2002

[Fly1995] M. J. Flynn, *Computer Architecture: Pipelined and Parallel processor design*, Jones and Bartlett Publ., 1995

[Hal2004] T. R. Halfhill, "Deluge of Multicore Processors for PC's, Servers, Embedded Systems", *Microprocessor report*, Vol. 18, pp. 16-20, Sept. 2004

[Hen2003] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach (3rd ed.)*, Morgan Kaufmann Publ., 2003

[Intel] Intel, Network Processors http://www.intel.com/design/network/products/npfamily/index.htm

[Keu2000] K.Keutzer et al. "System-level design: Orthogonalization of concerns and platform-based design," IEEE Trans. On CAD of Integrated Circuits and Systems., December 2000

[Kow2003] J. Kowalczyk, "Multiprocessor systems", *White paper: Virtex-II Series*, Xilinx WP162, Apr. 2003

[Kre2004] K. Krewell, "ARM Opens Up to SMP", *Microprocessor Report*, Vol. 18, pp. 1 and 5-7, May 2004

[Lee2002] Edward A. Lee, "Embedded Software", *Advances in Computers,* Vol. 56, Academic Press, London, 2002

[Mas2002] Anthony J. Massa, *Embedded Software Development with eCOS*, Prentice Hall, 2002 (for eCos, see also: http://ecos.sourceware.org/)

[Mol2005] A. Molnos, M. Heijligers, J.T.J. van Eijndhoven, "Compositional memory systems for multimedia communication tasks", accepted for publication in: *proc. Design Automation and Test in Europe*, Munich, Germany, Mar. 2005

[Nai2002] R. Nair, "Effect of increasing chip density on the evolution of computer architectures", *IBM J. on Research and Develop.,* Vol. 46, No. 2/3, pp. 223-234, March/May 2002

[Nic1998] Bradford Nichols et. al., *Pthreads Programming*, O'Reilly Publishers, 1998

[Oli2003] J.A. de Oliveira and H. van Antwerpen, "The Philips Nexperia Digital Video Platform," in *Winning the SoC Revolution,* G. Martin and H. Chang, Eds., pp. 67-96. Kluwer Academic, 2003.

[Ote2005] C. Otero Perez et.al. "Resource reservations in shared-memory multiprocessor SoCs", *other chapter in this book*

[Sri2000] S. Sriram and S. S. Bhattacharyya, "Embedded Multiprocessors: Scheduling and synchronization", *Marcel Dekker Inc.*

[Str2001] P.Stravers, J.Hoogerbrugge, "Homogeneous multiprocessing and the future of silicon design paradigms", *Proc. Of International symposium on VLSI Tech. Systems and applications*, 2001

[Suz2002] M. Suzuoki and T. Yamazaki, "Computer Architecture and Software Cells for Broadband networks", *US Pat. Appl. 2002/0138637A1*, Sep. 2002

[Tol2003] E.B.Van der tol, et. al, "Mapping of H.264 decoding on a multiprocessor architecture", Proceedings of SPIE conference on Image and Video Communication, Vol 5022, Jan 2003

[Wae2005] J.W. van de Waerdt et.al, "Motion Estimation Performance of the TM3270", accepted for publication in: *proc. 20th ACM Symp. on Applied Computing (SAC2005)*, Santa Fe, New Mexico, Mar. 14-17, 2005

[Wol2004] Pieter van der Wolf et. al., "Design and programming of embedded multiprocessors: An interface-centric approach", *proc. CODES+ISSS'04*, Stockholm, Sweden, pp. 206-217, Sept. 2004

[Won2002] W. Wong, "Quad 64-bit Multiprocessors targets comm. applications", *Electronic Design*, Oct. 2002. (See also Broadcom 'BCM1480' product brief, Sep. 2004)

[Yan2002] P.Yang et. al, "Managing Dynamic Concurrent Tasks in embedded real-time multimedia systems", proc. 15th Int. Symp. On System Synthesis (*ISSS'02)*, Kyoto, Japan, pp. 112-119, ACM Press, 2002