Integrating cache space reservations with processing cycles reservations for a multiprocessor system on chip

Clara M. Otero Pérez, Jos van Eijndhoven
Philips Research Laboratories
{clara.otero.perez, jos.van.eijndhoven}@philips.com

Abstract

This article presents the initial steps towards integrating a cache space reservation mechanism with processing cycles reservations. Consumer electronics vendors increasingly deploy shared-memory multiprocessor SoCs, to balance flexibility (late changes, software download, reuse) and cost (silicon area, power consumption) requirements. The dynamic sharing of scarce resources by media applications jeopardizes robustness and predictability. Resource reservation is a well-known technique to improve robustness and predictability. Various resource reservations mechanisms address this problem individually for each resource such as processor cycles, cache space, and memory access cycles. However different resource types are very closely interrelated. We present a novel approach that aims to integrate the reservation mechanisms of the processing cycles and cache space.

1. Introduction

Multiprocessor systems on chip (SoC), such as Philips Nexperia [1], are rapidly entering the high-volume electronics market. Progressive IC technology steps reduce the impact of programmable hardware on the total silicon area and power budget. This permits SoC designers to shift more and more functionality from dedicated hardware accelerators to software, in order to increase flexibility and reduce hardware development cost.

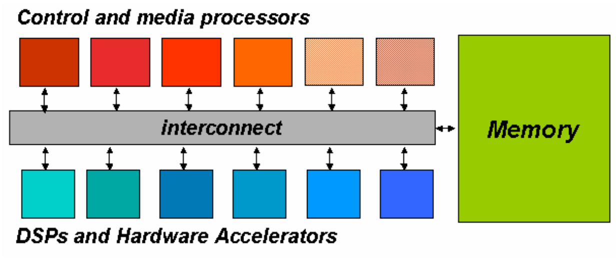

Figure 1 Heterogeneous SoC architecture with CPUs, DSPs, and accelerators communicating through shared memory.

However, these multiprocessor SoCs still combine flexibility—in the form of one or more programmable central processing units (CPU) and digital signal processors (DSP)—with the performance density of application-specific hardware accelerators. Figure 1 depicts such a heterogeneous SoC architecture as presented in [2].

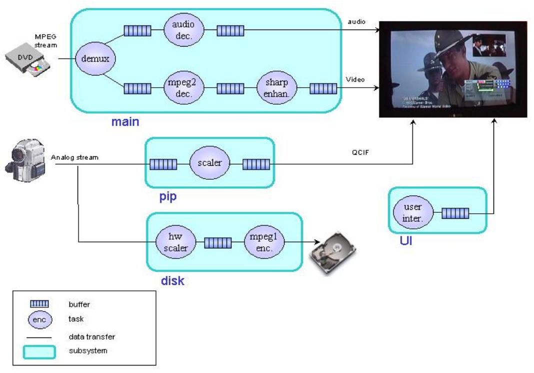

Figure 2 Media application example.

Figure 2 depicts an example of a streaming application [3] (mainly audio and video) that would run on such SoC. Our execution model for streaming applications consists of a connected graph in which the nodes represent either a task or a buffer. The interconnections represent the data transfer. The execution model is hierarchical. At higher levels of abstraction, a connected graph can again be viewed as a subsystem in a connected graph. Figure 2 depicts four such subsystems: main, pip, disk, and user interface (UI). The subsystems are denoted with the rounded rectangles.

2. Resource reservations

The sharing of resources is a potential source of interference that leads to unpredictability, and jeopardizes overall system robustness. Resource reservation is a well-known technique to improve robustness and predictability. It is based on four components, admission control, scheduling, accounting, and enforcement. When properly combined they provide guaranteed reservations.

Based on this approach, our system resource manager provides resource budgets to the subsystems. A *resource budget* is a guaranteed resource allowance. Guaranteed resource budgets provide temporal isolation, which contributes to robustness by bounding the interference caused by resource sharing.

The different resource types present in a SoC are very closely interrelated: for instance, a task running on a processor needs cache space to be able to execute. Therefore, the resource manager should address all shared resources in an integrated manner. The initial ideas towards this integration were presented in [4]. In this paper we address the integration of cache reservations with processing cycles reservations.

2. Cache space reservation

Figure 3 details the data path of a multiprocessor such as in Figure 1, in which a number of DSPs, CPUs, and accelerators communicate through shared memory, described in [5]. The architecture applies a two-level cache hierarchy to reduce memory bandwidth and latency requirements. The cache hierarchy is inclusive: a memory block can only be in a L1 cache if it also appears in the L2 cache. When a processing unit produces new data and stores it in its L1 cache, the L2 copy of that memory block becomes stale; in such cases a cache coherence protocol ensures that any consumer of the data always receives the updated L1 copy of the data (from L2). At any moment in time, a modified data item resides only in one of the L1 caches (and in L2). This property is intended to facilitate the partitioning of subsystems, consisting of multiple producer/consumer tasks, over multiple processors.

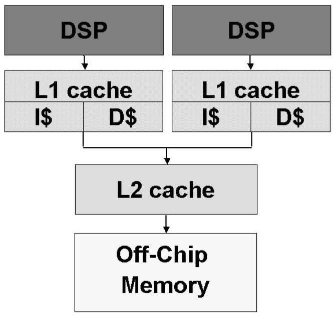

Figure 3 Data path for the memory hierarchy.

Traditional caches have been designed to work well for a single subsystem running on a single processing unit. In current systems, multiple subsystems execute concurrently sharing the L2 cache. These concurrent subsystems influence each other's performance by flushing each other's data out of the cache. Among the replacement and allocation techniques proposed, some of them use the concept of budgeting (or reservations). A given subsystem/task/thread has exclusive access to a specific part of the cache and will not suffer interference from other subsystems, which also have their own piece of cache. In [6] and [7] we can find examples of these budgeting which are spatial budgets.

Spatial budgeting improves subsystem performance by improving cache predictability. Furthermore, it enables compositionality of software subsystem. However, in resource constrained system the cache is also a scarce resource. This means that when a subsystem requests a cache budget, this cache space may not be available. In general, subsystems will not receive as much cache space as they require, with the derived performance penalty.

2. Processing cycles reservations

CPU-cycle budgets are provided to subsystems and must match the CPU cycle requirements of the subsystems. Media processing subsystems typically require periodic budgets with a budget value C (number of processing cycles), a granularity T (period of activation), and a deadline D. A periodic budget is replenished at regular intervals (T).

For lack of space but without loss of generality we explain in this section the simple case of periodic budgets and a reservation algorithm based on rate monotonic scheduling (RMS). This algorithm [8] was initially conceived for independently executing task. In our case, individual tasks are not independent whereas subsystems are. The same reasoning that used to apply to tasks applies now to subsystem budgets. We use an admission control algorithm that corresponds to the scheduling algorithm being used. If the admission control fails, the corresponding budgets cannot be guaranteed, therefore the subsystem corresponding to the budgets that causes the failure is not allowed to start (or to modify its resource requirements). When using RMS as scheduling algorithm a simple formula (1) for response time calculation from [9] is used:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j \le D_i \qquad (1)$$

In this formula, index i identifies the budget, T_i is the period, C_i is the budget capacity, R_i is the response time, and hp(i) is the set of all budgets with priority higher than i. The solution to this equation is the response time for task i. If the response time is smaller than the deadline the subsystem budget is guaranteed.

3. Integrated approach

To present the first step towards integrating the L2 cache resource reservation with processing cycles reservations assume a system with three independent subsystems, such as Main, Disk and UI from Figure 2, executing on two processors. Subsystem one and two execute on processor one and subsystem three executes on processor two. Note that no data is shared among the subsystems. The budget scheduling algorithm is rate monotonic scheduling. Figure 4 depicts the resulting execution behavior and the corresponding cache reservation. In this case the cache reservation for each subsystem is 1/3 of the total cache.

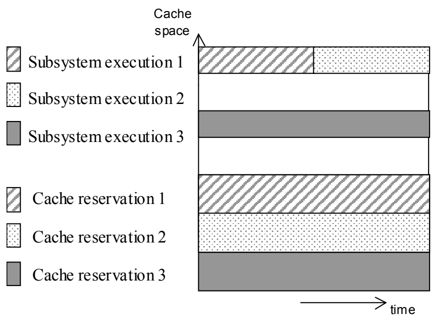

Figure 4 Cache reservation without using processing budget information

The cache reservation mechanism is controlled in software. We can use this mechanism dynamically by reserving cache space when the subsystem needs it and freeing (reallocating) it when the subsystem does not need it. The cache allocation decisions are made in software, and are enforced by novel extensions in the hardware cache controller. The difference with previous work is in the definition of "when the subsystem needs it". Until now the subsystem needed the cache space during its life time. In Figure 4, subsystem 1 keeps its cache reservation until the end of the period, even when it will not use it. Knowing that a subsystem will not execute for some time is not easy in the general case. However, if a processing budget is also provided then we can calculate exactly when a subsystem starts executing and when it will finish. The processing budget is available with granularity much smaller than the lifetime of the subsystem. For example a processing budget of 5 milliseconds each 10 milliseconds with respect to the lifetime of a subsystem of several hours.

Knowing the scheduling mechanism, and assuming equal periods for the processing budget we can calculate the worst case busy period per subsystem per processor. For the highest priority subsystem the start time is 0 and the finishing time is the response time calculated by (1). For the other subsystems, the start time is the response time of the adjacent higher priority subsystem and the finishing time is calculated by (1). In a more general case the earliest start time and latest end time can be calculated using the formulas from [10].

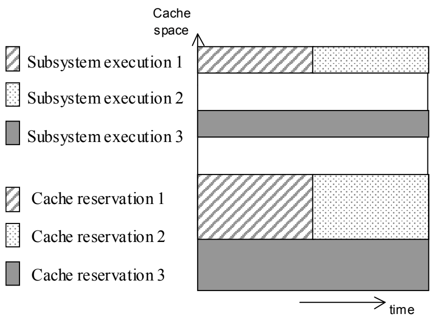

Figure 5 Optimized cache reservation by using processing budget information

Calculating the worst-case busy period, we can use the disjoint busy periods to maximize cache budget provision. In Figure 5 we can see how the cache space used by subsystem 1 is freed to be used by subsystem 2, maximizing overall cache space utilization.

This cache management mechanism is independent of (orthogonal to) other features like prefetching or invalidation for streaming data or multi-processor cache coherency.

4. Conclusion and future work

This article takes on the first challenge towards integrated resource management for SoC and presents a coherent approach that integrates cache space reservations with processing cycles reservations. By combining these mechanisms the cache space is more effectively used: when a subsystem does not need its reservation the space can be made available for other subsystems.

The current mechanism assumes a simple model where all budgets have the same period and fixed priority scheduling is used. We are working towards the extension of this mechanism towards the case of arbitrary periods and dynamic priorities.

4. Acknowledgment

The authors would like to thank Liesbeth Steffens, Martijn Rutten and Evert-Jan Pol for providing the initial ideas for this work.

6. References

[1] 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.

[2] P. Stravers and J. Hoogerbrugge, "Homogeneous multiprocessing and the future of silicon design paradigms", *in Proceedings of the International Symposium on VLSI Technology, Systems, and Applications(VLSI-TSA)*, Apr.2001

[3] C.M. Otero Pérez, E. Steffens, G.v. Loo, P.v.d. Stok, R. Bril, A. Alonso, M. Garcia Valls, and J. Ruiz, "QoS-based resource management for ambient intelligence," in *Ambient Intelligence: Impact on Embedded System Design,* T. Basten, M. Geilen, and H. de Groot, Eds., pp. 159-182. Kluwer Academic Publishers, 2003.

[4] C.M. Otero Pérez, M. Rutten, E. Steffens, J. van Eijndhoven and P. Stravers, "Resource Reservations in Shared Memory Multiprocessor SoC," in *Dynamic and robust streaming between connected consumer electronic devices,* P. van der Stok, Ed., 2005.

[5] J. van Eijndhoven, J. Hoogerbrugge, J. Nageswaran, P. Stravers, and A. Terechko, "Cache-Coherent Heterogeneous Multiprocessing as Basis for Streaming Applications," in *Dynamic and robust streaming between connected consumer electronic devices,* P. van der Stok, Ed., 2005.

[6] A. Molnos, M.J.M. Heijligers, S.D. Cotofana, and J. van Eijndhoven, "Compositional memory systems for multimedia communicating tasks", in *Proceedings of Design Automation and Test in Europe (DATE)*, Mar. 2005, Munich, Germany.

[7] I. Ravi, "CQoS: a framework for enabling QoS in shared caches of CMP platforms", in *Proceedings of the 18th annual international conference on Supercomputing*, pp. 257-266, ACM Press, 2004.

[8] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", *in Journal of the ACM*, vol. 20, no. 1, pp. 46-61, 1973

[9] M. Joseph and P. Pandya, "Finding response times in a real-time system", *in British Computer Society Computer Journal*, vol. 29, no. 5, pp. 390-395, Oct.1986

[10] R. J. Bril, "Real-time scheduling for media processing using conditionally guaranteed budgets", *PhD. Thesis, Technical University of Eindhoven*, 2004.