Multi-core PMCs: analysis and architectural definition

Document Information

Author	Mikel Fernández, Francisco J. Cazorla,
Contributors	Luca Fossati, Jaume Abella
Reviewer	Jaume Abella, Leonidas Kosmidis, Javier Jalle
Keywords	Performance Monitoring Counters

Table of Contents

1 Introduction

The main goal of this activity is to understand the philosophy of the Performance-Monitoring Counter (PMC) implemented in current processors in this activity. We are not seeking for a list of all performance-monitoring counters (PMCs) of the analysed processors but to understand their philosophy. We are after a set of monitoring counters that provide some information about the worst-case behavior of the application, so rather than PMCs they should be called WCMCs (worst-case monitoring counters). It is also worth noting that we focus on inter-task interference effects so our focus is going to be on analyzing existing multicore processors PMC.

Before starting this activity our intuition was that PMCs in current processors provide information about inter-task interferences explicitly. As a first step to confirm this intuition, this document analyses the main features of some of the multicore products of some of the major chip providers: Intel, ARM, IBM and Freescale.

We also analyse the PMC support in the Aeroflex Gaisler NGMP, and compare its infrastructure and events to the ones available in the other multicore processors. Finally, a set of WCMCs are proposed to be added in the NGMP based on the knowledge obtained in the analysis of the existing multi-core processors.

2 IBM POWER7

The IBM POWER7 is an 8-core multicore processor in which each core is 4-thread SMT. Moreover, every core is divided into two clustered execution pipelines, with each one supporting two threads. Threads in the same cluster share more resources than threads in different clusters, although threads in different clusters still share some on-core resources. Hence, resources are shared a) between threads in different cores, b) between all threads running in a core, c) between threads running in a cluster and d) between threads running in different clusters.

The POWER7 core has a dedicated 32-KB 4-way instruction cache and a 32-KB data cache. The first level of address translation is formed by an instruction effective-to-real-address translation (IERAT) and a D-ERAT. The I-ERAT consists of two 32-entry cache-like structures. The D-ERAT consists of two 64-entry cache-like structures. The second level of address translation is comprised of a Segment Lookaside Buffer (SLB) with 32 entries per thread and a Translation Lookaside Buffer (TLB) with 512 entries. Each core has a 256KB L2 cache and a local 4MB L3 region that can be shared between all cores forming a 32MB global L3 cache.

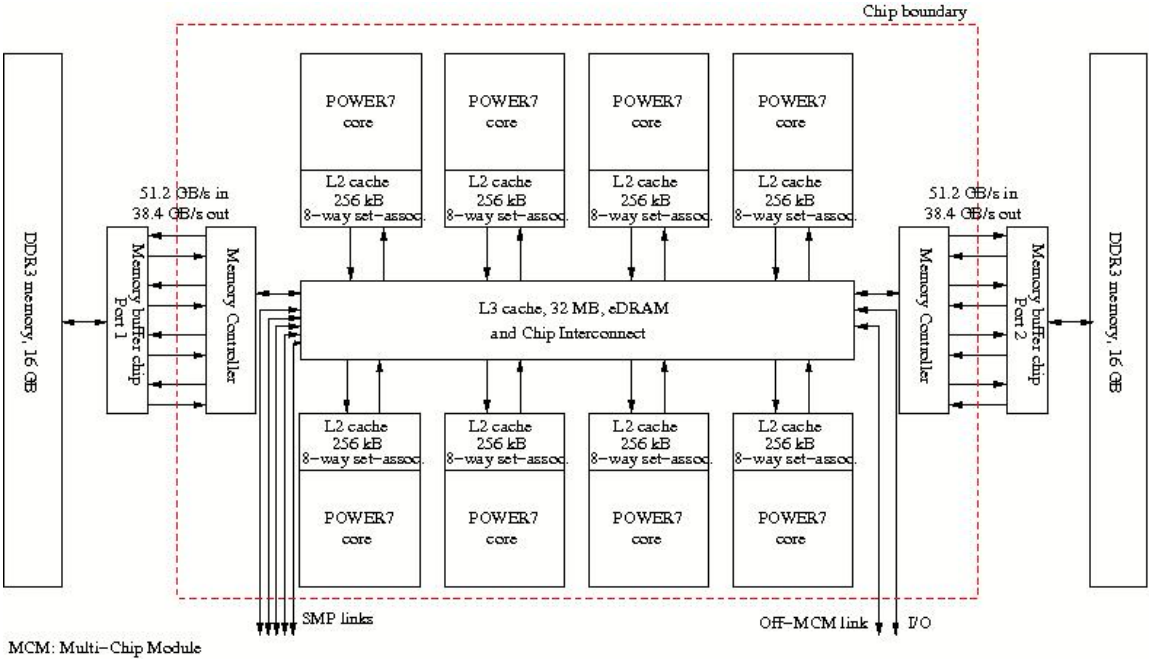

Figure *1 Block diagram of the IBM POWER7*

Figure *2* provides a view of the core resources at the core level and how they are shared among the threads in the different clusters.

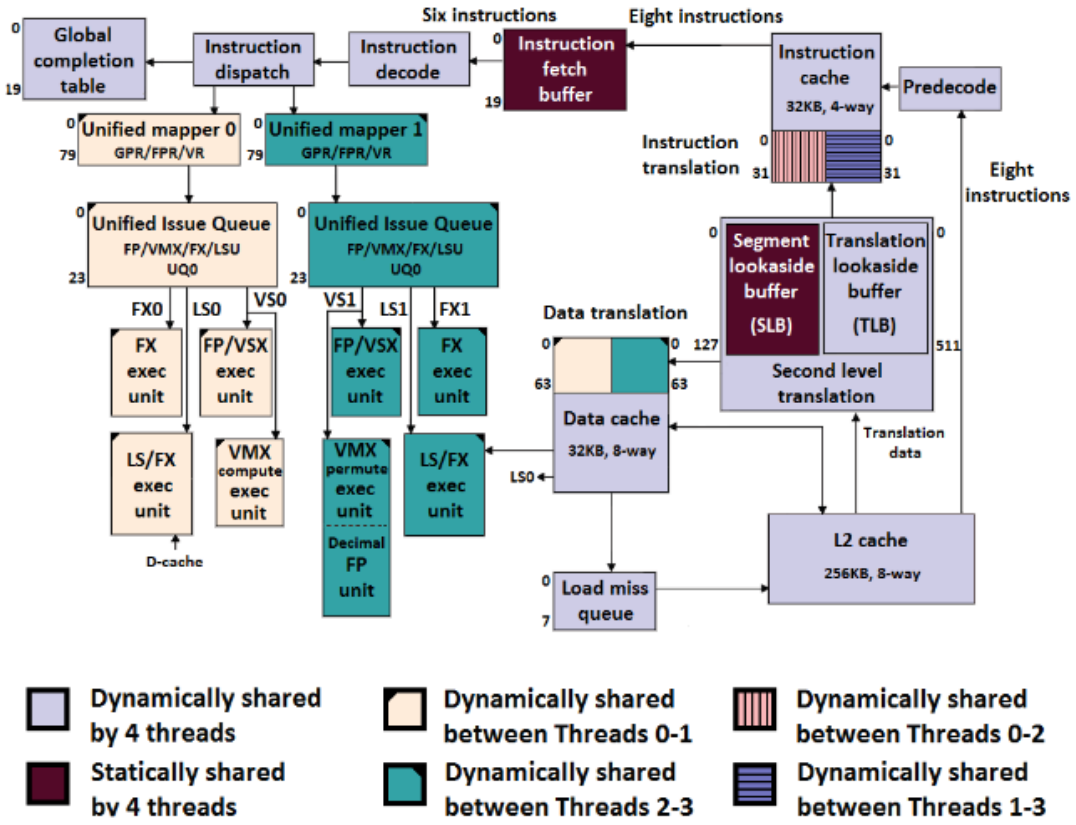

Figure *2 Block diagram of the core resources and how they are shared. [P7-desc].*

Program performance analysis in such a complex core with resources shared among different threads is complex. The POWER7 [P7-PMCs] processor comprises a Performance Monitoring Unit (PMU) with six thread-level Performance Counter Monitors (PCMs). Four of these are programmable from software to monitor the desired (four) events at the same time. There are more than 500 possible performance events that can be read. However, performance counters are defined by groups and the PMU can only watch events of the same groups at one time. Some counters are per-thread and others per-core.

We can regard the 'type of events' covered by a particular PMC as follows:

- Number of cycles a resource is full (with this causing the stall of the processor),
- Number of cycles a resource is empty. This can be of a private resource of a shared resource. The latter meaning that none of the threads that can generate a request to that resource have done so.
- Number of instructions of a given type
- Number of events of a given type (e.g. prefetch requests sent, …)
- Number of references to a given resource (e.g. L2 accesses, …)
- Quantity of data transferred
- Stall cycles due to inter-task conflict

Table 1 *Example of PMCs in the POWER7*

PMC name	*Type/subtype*	*Description*
PM_1PLUS_PPC_DISP	cyc count	Cycles at least one Instr. Dispatched
PM_BR_MPRED	Inst. Count	Number of Branch Mispredictitions
PM_CMPLU_STALL_DCACHE_MISS	Stall cycle count	Completion stall caused by D cache miss
DATA_FROM_L2	data count	Data loaded from L2. The processor's Data Cache was reloaded with data from the local chiplet's L2 cache due to a demand load
PM_GCT_NOSLOT_IC_MISS GCT	Empty cycle count	empty by I cache miss Cycles when the Global Completion Table has no slots from this thread because of an Instruction Cache miss.
M_L3_MISS L3 Misses	Event count	Total L3 references that miss the L3 on a per core basis. This event is delivered from the L2 domain, so must be scaled accordingly (divided by 2)
PM_L2_DC_INV	Event count. Inter-task interference?	Dcache invalidates from L2 Total L1 dcache Invalidates sent by L2 over reload bus on a per core basis. The L2 is inclusive of L1 icache and L1 dcache, so it has to invalidate the caches when it rolls over. This event is delivered from the L2 domain, so must be scaled accordingly (divided by 2)
PM_CMPLU_STALL_THRD	stall cycle count Inter-task conflict	Completion Stalled due to thread conflict. Group ready to complete but it was another thread's turn

2.1 CPI stack

An interesting feature of POWER7 is that people from IBM has created a CPI stack of the POWER7 based on its PMCs. The CPI stack relates functional processor stages with performance counters to show which CPU functional unit is generating stalls.

The bar on the left *cycles* shows the total cycles in which the application was running. These cycles are breakdown intro 3 categories in the second column.

Table 1. Partial POWER 7 CBM

Cycles (PM_RUN_CYC)	Completion Stall Cycles <C> (PM_CMPLU_STALL)	Stall by FXY <C2> (PM_CMPLU_STALL_FXU)	FXU Multi-cycle Instruction <C2A> (PM_CMPLU_STALL_DIV)	
			FXU Other * (C2-C2A) (PM_CMPLU_STALL_FXU_OTHER)	
		Stall by VSU <C3> * (C3A + C3B + C3C) (PM_CMPLU_STALL_VSU)	Stall by Scalar <C3C> (PM_CMPLU_STALL_SCALAR)	Stall by Scalar Long <C3C1> (PM_CMPLU_STALL_SCALAR_LONG)
				Stall by Scalar Other <C3C2> * (C3C - C3C1) (PM_CMPLU_STALL_SCALAR_OTHER)
			Stall by Vector <C3B> (PM_CMPLU_STALL_VECTOR)	Stall by Vector Long <C3B1> (PM_CMPLU_STALL_VECTOR_LONG)
				Stall by Vector Other <C3B2> * (C3B - C3B1) (PM_CMPLU_STALL_VECTOR_OTHER)
			Stall by DFU <C3A> (PM_CMPLU_STALL_DFP)	
		Stall by LSU <C1> (PM_CMPLU_STALL_LSU)	Stall by Reject <C1A> (PM_CMPLU_STALL_REJECT)	Translation Stall <C1A1> (PM_CMPLU_STALL_ERAT_MISS)
				Other Reject <C1A2> (C1A - C1A1) (PM_CMPLU_STALL_ERAT_OTHER)
			Stall by D-cache Miss <C1B> (PM_CMPLU_STALL_DCACHE_MISS)	
			Stall Store <C1B> (PM_CMPLU_STALL_STORE)	
			LSU Other <C1D> * (C - C1A - C1B - C1C) (PM_CMPLU_STALL_LSU_OTHER)	
		Stall due SMT <C4> (PM_CMPLU_STALL_THRD)		
		Stall due IFU <C5> (PM_CMPLU_STALL_IFU)	Stall due BRU <C5A> (PM_CMPLU_STALL_BRU)	
			Other IFU Stall <C5B> * (C5 - C5A) (PM_CMPLU_STALL_IFU_OTHER)	
		Other Stall <C6> * (C - C1 - C2 - C3 - C4 - C5) (PM_CMPLU_STALL_OTHER)		
	GCT Empty Cycles <B> (PM_GCT_NOSLOT_CYC)	GCT Empty due Icache Miss <B1> (PM_GCT_NOSLOT_IC_MISS)		
		GCT Empty due branch Mispredict <B2> (PM_GCT_NOSLOT_BR_MPRED)		
		GCT Empty due branch Mispredict and Icache Miss <B3> (PM_GCT_NOSLOT_BR_MPRED_IC_MISS)		
		GCT Empty Other * (B1 - B1 - B2 - B3) (PM_GCT_EMPTY_OTHER)		
	Completion Cycles <A> (PM_GRP_CMPL)	Base COmpletion Cycles <A1> (PM_1PLUS_PPC_CML)		
		Overhard of expansion * (A-A1)		

Figure *3 POWER7 CPI stack model*

The focus is put on the Global Completion Table (GCT or Reorder Buffer, ROB). We have the following 3 categories: the cycles in which the application is *stalled* due to any reason *Completion_Stall_Cycles*; the cycles in which the application is stalled because some resources are idle *GCT_empty_cycles*. And the cycles in which instructions are actually being committed *Completion_Cycles*. *Completion_Stall_Cycles*, in a third breakdown, is split into more detail categories that generate the slowdown the Fixed-Point Unit, The Vector unit, the load store unit, the instruction-fetch unit and the stall due to SMT execution. The other components are similarly breaked down.

Metrics in white boxes are derived from a single PMC, while, metrics in gray boxes are derived from different PMCs.

3 Intel Family

Intel processors designs focus on maximizing average performance. Usually Intel processors feature superscalar execution, complex branch predictors, out of order execution, and several levels of cache memories (up to three). PMCs provided by Intel focus on providing performance metrics for a single process. To this end, the architectures provide counters for analyzing branch predictor effectiveness, cache misses due to speculative execution, coherence protocol metrics, etc.

PMCs provided by Intel are not designed to help identify bottlenecks in resources shared between cores, or to quantify the magnitude of interactions in these shared resources. A lot of counters provide measurements for time spent accessing a shared resource, but they measure average or total accumulated time, and they do not identify possible interferences caused by other cores.

For example, there is a counter called INST_QUEUE_WRITES which counts the number of instructions written to the instruction queue. There is another counter called INST_QUEUE_WRITE_CYCLES which counts the number of cycles spent writing to this same queue. As a result, we are able to obtain the average number of cycles it takes to write, but there is not any counter to hold the maximum observed latency observed.

Most counters can be configured to measure events for either from a core or all the cores, an agent[1] or all agents, and other kinds of specific qualifications (such as detection of all events/exclude prefetching events, or counts for different states for the coherence protocol used).

Many shared resources have PMCs that measure their behavior (in terms of number of accesses, average latency, number of cache misses, etc), and in some cases and with some experimentation, information about inter-processor conflicts can be guessed: for instance, we could approximate the amount of L2 cache lines evicted by another process comparing the observed number of L2 misses when running in isolation with the number of L2 misses when running concurrently with a competing process. Unfortunately, this is not possible at deployment time, because the observed events are dependent on the specific way tasks have been scheduled.

Some of the PMCs that measure shared resource behavior which could be used to detect interference between cores are described in the following sections.

3.1 PMC infrastructure

Intel names the PMC control registers as performance event select registers (shortened as `IA32_PERFEVTSELx`), and the performance monitoring counters are shortened as (`IA32_PMCx`). Intel provides some architectural infrastructure to query about the availability of PMCs, but ultimately, the amount of counters available is model dependent. The width of the PMCs is also implementation dependent. This information can be queried from the CPUID.0AH leaf.
The architectural infrastructure for PMCs, which comprises 3 different versions of the PMC facilities, can be also queried at run time. Each newer version provides upgrades

[1] Intel defines “Agent” as a component (core or other kind) which is able to cause a PMC event.

on the infrastructure: for example, the 3rd version provides improved support for multi-threaded processors.

3.2 PMC availability in each processor model

Intel provides a small set of PMCs which are available throughout all their processors. These are called Architectural Performance Monitoring Counters. The rest of the PMCs are available for some processors. Intel organizes similar processors in generations of processors. Each processor generation has the same PMCs available.
The following table summarizes which processors belong to each generation:

Table 2 *Intel processor generations*

Generation (Codename)	Processors
4th Generation (Haswell)	Xeon E3-1200 v3
3rd Generation (Ivy Bridge)	Xeon E3-1200 v2
2nd Generation (Sandy Bridge)	Core i7-2xxx Core i5-2xxx Core i3-2xxx Xeon E3-1200
1st Generation (Nehalem)	Core i7 Xeon 5500 Series
(Westmere)	Xeon E7-xxxx

The PMCs described in later sections are classified based on availability in one or more processor generations.

3.3 PMC classification

To ease the understanding of Intel's philosophy on PMCs, we have classified them in categories.

- Instruction type: number of retired instructions of a certain type (branch, load, store, etc).
- Event count: microinstructions issued, branch instructions executed, number of speculative instructions retired, number of miss-predictions, number of cache misses.
- Reference count: number of referenced lines in each MESI state, number of each level of cache references.
- Threshold exceeding event count: number of times a threshold specified in number of cycles has been exceeded for a given event. The threshold value is configured by the user.
- Number of outstanding requests per core: cache requests, all offcore requests, etc. in the moment of reading the counter.
- Busy resource cycles: number of cycles in which the caches are busy, a resource is unavailable, stalled core, etc.
- Cycle count: unhalted core cycles, unhalted thread cycles, cycles with outstanding cache misses, TLB walk duration, etc.

- Resource-specific event count: number of lines brought to each cache, data and instruction TLB misses for loads or stores, completed page walks per page size, number of split memory accesses, number of retired load/store instructions in each memory in the hierarchy, etc.

3.4 Interesting PMCs

Intel implements a high number of PMCs in their processors, but they are not focused on worst observed case, or on interference between cores detection. In this chapter a selection of the most relevant PMCs for the purpose of creating a set of WCMC is presented.

Table 3 *PMCs with possible WCMC potential*

PMC name	*Description*
LD_BLOCKS.NO_SR	Measures split load operations (i.e. data is split throughout two different cache lines) blocked because all resources for handling them are in use.
L2_LINES_OUT.*	Measures L2 clean/dirty line eviction caused by demand or by the prefetcher.
.*DEMAND_CLEAN*	Clean lines evicted by demand
.*DEMAND_DIRTY*	Dirty lines evicted by demand
.*PF_CLEAN*	Clean L2 cache lines evicted by the MLC prefetcher.
.*PF_DIRTY*	Dirty L2 cache lines evicted by the MLC prefetcher.
.*DIRTY_ALL*	Dirty L2 cache lines filling the L2.
LONGEST_LAT_CACHE.*	References and misses to the last level of cache, originating from each core.
CYCLE_ACTIVITY.CYCLES_L2_PENDING	Number of cycles with pending L2 miss loads
MEM_LOAD_UOPS_RETIRED.*	Number of load micro-operations which were served by each memory in the hierarchy.
MEM_TRANS_RETIRED.LOAD_LATENCY	Counts the amount of loads whose latency is above a user defined threshold, but only a small fraction of the overall loads are randomly sampled. This counter can help finding an approximate worst observed behavior
MEM_LOAD_RETIRED.OTHER_CORE_L2_HIT_HITM	Load instructions that hit another core's L2 cache Measure positive interaction between cores.

There are available some coherence counters as the one shown next. As control (critical) applications are expected to be single-threaded these counters are not

considered of interest at this moment in time. We acknowledge that in case of migrations of one task from one core to another, when the task start running in the new core, some events of the coherence protocol will be trigger, but we expect the impact on timing analysis of those to be low.

`UNC_CBO_XSNP_RESPONSE.*`: for snoop hits and invalidations.

`UNC_CBO_CACHE_LOOKUP.*`: for cache coherence status: number of times a line has been found in L3 in modified, exclusive, shared, or invalid on a MESI coherence protocol.

`L2_RQSTS.*`: Measures the L2 hit/miss, coherence protocol related misses, prefetcher misses, etc. (all generations).

3.5 Summary

Overall, the PMCs provided in Intel processors are designed to provide average performance metrics, but not to identify sources of inter-processor conflicts in shared resources. They are neither fit to measure worst observed behaviors.

4 ARMv7-A

The ARMv7-A architecture [ARMv7-A] provides 6 different 32-bit counters, which can count any event available. This architecture is used by several different processors: Cortex-A7, Cortex-A9, Cortex-A15, the big.LITTLE system, and others.

4.1 Performance Monitoring Unit

ARMv7-A provides a Performance Monitoring Unit (PMU) with 6 performance counters, as shown in Figure *4*.

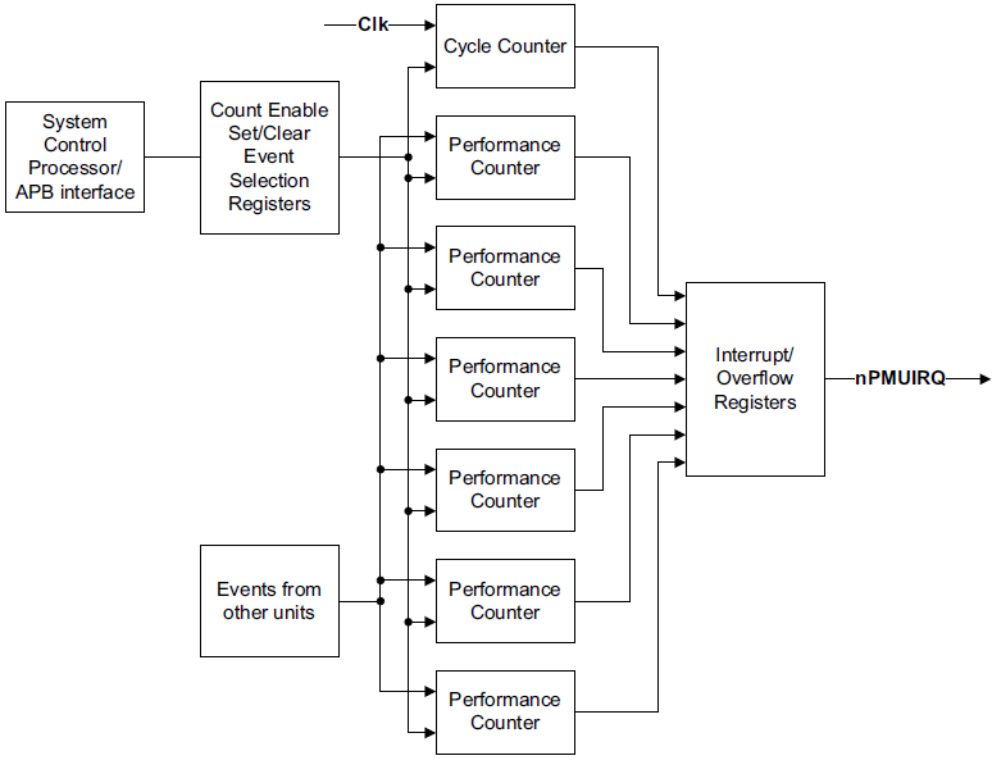

Figure *4 ARMv7 Performance Monitoring Unit block diagram [ARMv7]*

The ARMv7-A architecture includes a set of control registers to allow performance monitoring. The most important of them are:

- `PMXEVCNTR`, which holds the value of the configured PMC.
- `PMSELR`, which configures the counter that will be used when counting an event.
- `PMXEVTYPER`, which selects the event that will increment the selected counter.
- `PMCCNTR`, which counts the amount of cycles (or cycles/64).
- Counters are set and cleared using the `PMCNTENSET` and `PMCNTENCLR` registers.

4.2 PMC classification

The events counted by ARM architectures can be classified in the following types:

- Instruction type: number of retired instructions of a certain type (branch, load, store, etc).
- Event count: miss-predicted branches, number of exceptions, etc.
- Reference count: number of L1 accesses, bus accesses, data memory accesses, unaligned accesses to memory, etc.
- Cycle count: CPU cycles, bus cycles.
- Resource-specific event count: L1 write backs, number of L1/L2 refills.

4.3 Cortex-A7 and Cortex-A9

ARM processors have a small set of PMCs available, as they are designed to be power-efficient and not as focused on performance. In Figure *5* the pipeline of the Cortex-A7 is shown. It is an in order, dual-issue, 8 to 10 stage pipelines processor.

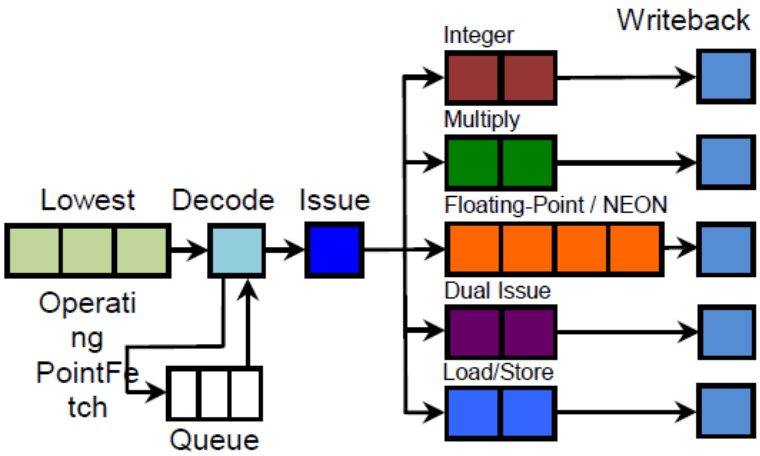

Figure *5 Block diagram of the ARM Cortex-A7 pipeline[bL]*

These processors are widely used in handheld devices which use Java based operating systems such as Android. For this reason, ARM provides PMCs to measure Java bytecode performance. In the last years, ARM has been designing CMP chips, but there is not yet any counter designed to measure the effect on the core caused by other cores. There is not any counter for worst observed behavior either.

4.4 Cortex-A15

The Cortex-A15 [ARM-A15] is a much more complex processor compared to its predecessors. It also provides a much larger set of counters compared to Cortex-A7 and Cortex-A9. As can be seen in *Figure 6*, it features a 15 to 24 stage triple issue pipeline and supports out-of-order execution. It focuses on performance rather than power-efficiency.

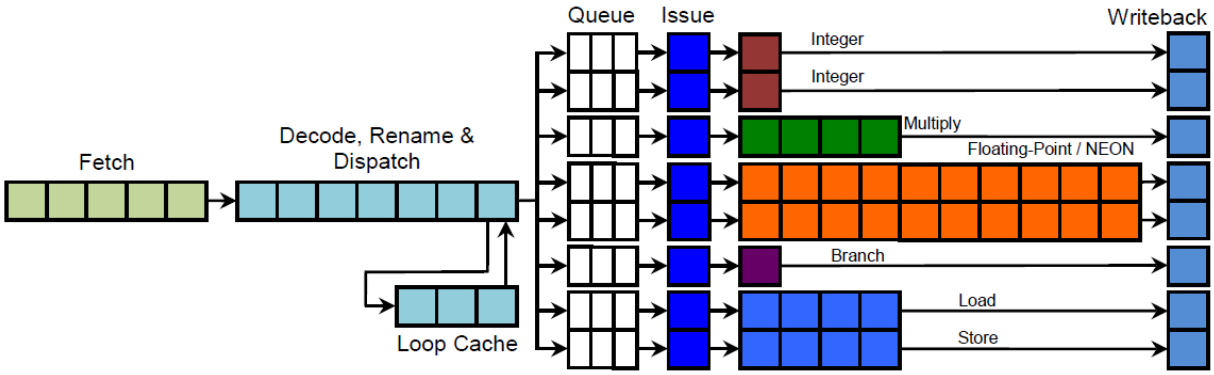

Figure 6 Block diagram of the ARM Cortex-A15 pipeline [bL]

In Figure *7*, a block diagram of the Cortex-A15 is presented, showing a 4 core multicore design. The Performance monitoring unit (one per core) is highlighted.

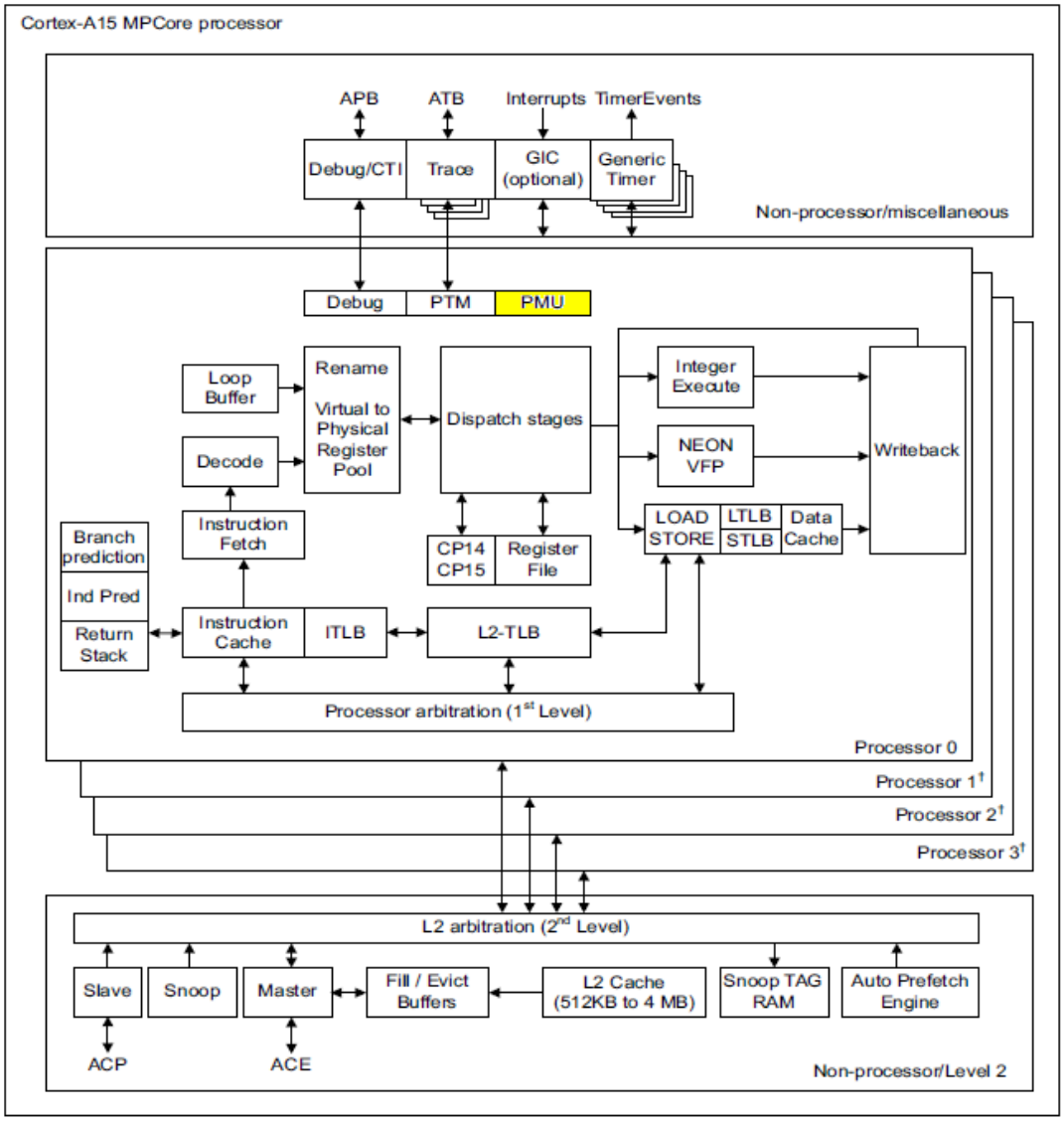

Figure *7 Block diagram of the ARM Cortex-A15 [ARM-A15]*

Cortex-A15 introduces new PMCs which take into account the multi-core hardware architecture. Several counters are introduced to model how each process use the bus:

- `BUS_ACCESS_LD`: Amount of read bus access.
- `BUS_ACCESS_ST`: Amount of write bus access.
- `BUS_ACCESS_SHARED`: Bus access, Normal, Cacheable, Shareable
- `BUS_ACCESS_NOT_SHARED`: Bus access, not Normal, Cacheable, Shareable
- `BUS_ACCESS_NORMAL`: Bus access, normal
- `BUS_ACCESS_PERIPH`: Bus access, peripheral

4.5 big.LITTLE

ARM's big.LITTLE [bL] is a system designed to combine high performance and low power consumption. It features an energy-efficient Cortex-A7 processor, and a powerful Cortex-A15, connected via a CCI-400 Network-on-Chip which provides full coherency.

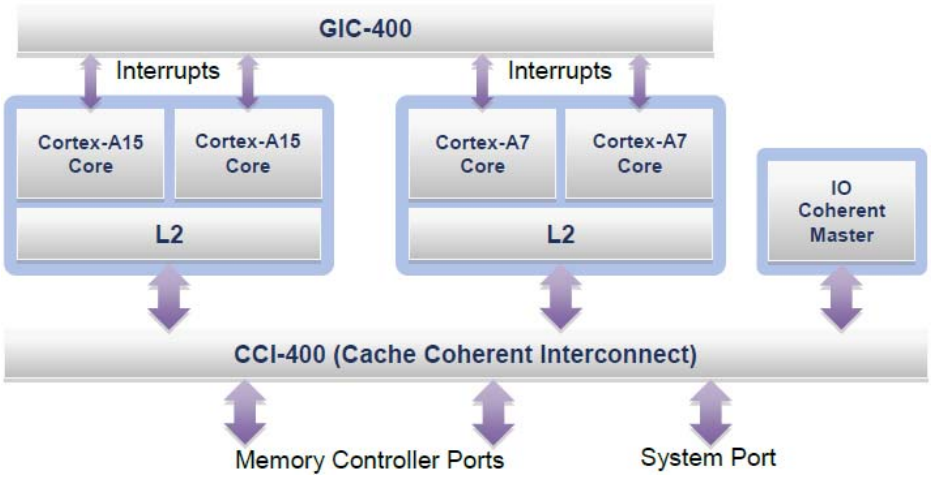

Figure *8 Block diagram of the big.LITTLE architecture [bL]*

While architecturally both processors support ARMv7-A, micro-architecturally they differ. Cortex-A7 is an in order, dual-issue, 8 to 10 stage pipelines processor, while Cortex-A15 supports out-of-order execution, triple issue and 15 to 24 stage pipelines. This allows big.LITTLE to be very power-efficient when running applications which are not focused on performance, while being able to sacrifice power-consumption when performance is required.

Cortex-A15 sacrifices energy-efficiency for performance when needed. The following table compares energy and performance of both processors for some benchmarks:

Table 4 *Performance vs energy efficiency comparative*

	Cortex-A15 vs Cortex-A7 Performance	Cortex-A7 vs Cortex-A15 Energy Efficiency
Dhrystone	1.9x	3.5x
FDCT	2.3x	3.8x
IMDCT	3.0x	3.0x
MemCopy L1	1.9x	2.3x
MemCopy L2	1.9x	3.4x

4.5.1 Corelink CCI-400 Cache Coherent Interconnect

big.LITTLE connects the cores by using the Corelink CCI-400 [CCI400] network on chip, which provides coherence, and QoS, and QoS virtual networks. Hardware managed coherence helps improve power consumption because it reduces memory accesses and it reduces software coherence mechanisms' overhead. It complies with the AMBA AXI and ACE specification. It provides a crossbar interconnect functionality between masters (2 ACE, 3 ACE-Lite) and slaves (up to three).

In Figure *9*, an example architecture is presented, using 2 ACE masters (Cortex-A15 and Cortex-A7), 3 ACE-Lite slaves, and 3 ACE-Lite masters.

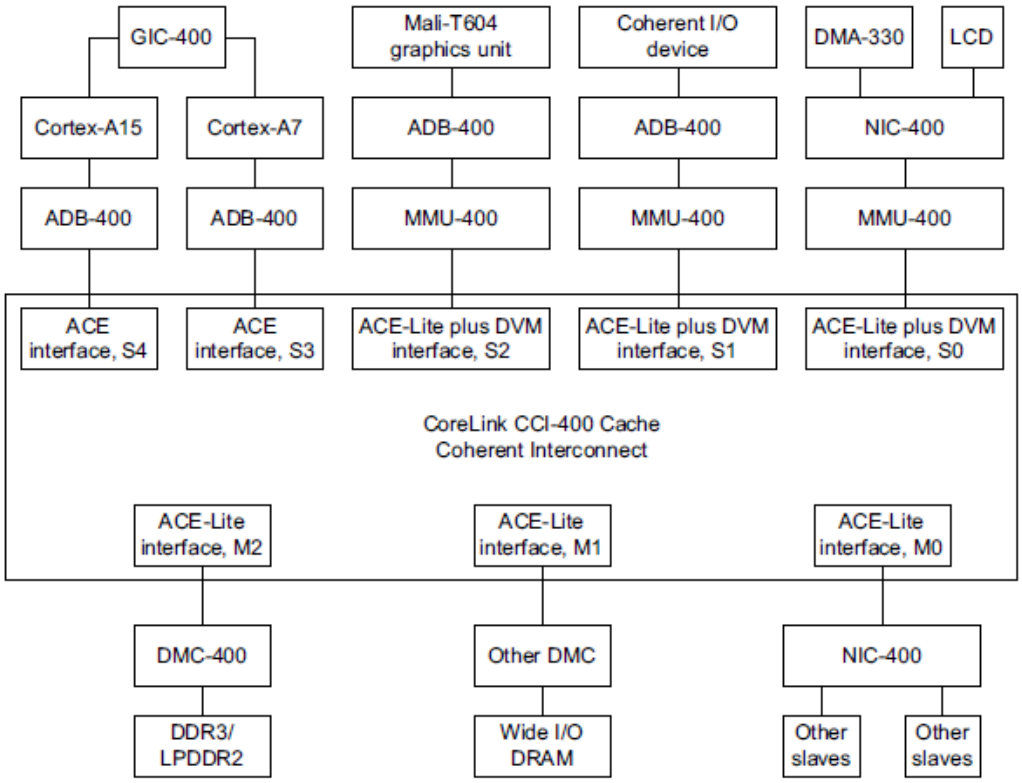

Figure *9 Example architecture with 5 masters and 3 slaves connected to the CCI-400 interconnect.[CCI400]*

The Corelink CCI-400 interconnect provides also a Quality of Service (QoS) mechanism to isolate packets in three different groups: low, medium and high priority. In every arbitration point, the highest priority request is granted access. If arbitration occurs between two requests of the same priority, a Least Recently Granted (LRG) algorithm is used for granting access.

The QoS also provides a mechanism for token based virtual networks (VN). Having VN allows that a packet in a network is able tot reach its destination even when a transaction in a different VN is blocked. This is very useful to prevent high-bandwidth transactions from blocking latency-critical tasks from using the interconnect. Up to four VN can be used.

This interconnect provides a performance monitoring unit (PMU) with four 32 bit counters which can be configured to count any available event.

Table 5 shows a list of the events that can be counted:

Table 5 *PMCs provided by the Corelink CCI-400 interconnect.*

Event name	Type
Read request handshake: any; device transaction; normal, non-shareable or system-shareable, but not barrier or cache maintenance operation; inner- or outer-shareable, but not barrier, DVM message or cache maintenance operation; cache maintenance operation; memory barrier; synchronization barrier; DVM message, not synchronization; DVM message, synchronization; and data returned from the snoop instead of from downstream. Write request handshake: any; device transaction; normal, non-shareable, or system-shareable, but not barrier; inner- or outer-shareable, WriteBack or WriteClean; WriteUnique; WriteLineUnique; and Evict. RETRY of speculative fetch transaction.	Number of requests
Read request stall cycle because the transaction tracker is full. Increase `SIx_R_MAX` to avoid this stall; `RVALIDS` is HIGH, `RREADYS` is LOW; master interface ID hazard; barrier hazard ; and because slave interface ID hazard Write request stall cycle because the transaction tracker is full. Increase `SIx_W_MAX` to avoid this stall; barrier hazard; and stalled for a cycle because the write transaction tracker is full. Increase `MIx_W_MAX` to avoid this stall. Stall cycle because of an address hazard. A read or write invalidation is stalled because of an outstanding transaction to an overlapping address.	Stall cycles
A read request with a QoS value in the high priority group is stalled for a cycle because the read transaction queue is full. Increase `MIx_R_MAX` to avoid this stall. A read request with a QoS value in the low priority group is stalled for a cycle because there are no slots available in the read queue for the low priority group. A read request with a QoS value in the medium priority group is stalled for a cycle because there are no slots available in the read queue for the medium priority group. A read request is stalled for a cycle while it was waiting for a QVN token on VN0. (There is a counter for each VN, for reads and writes)	QoS stall cycles

When enabling performance monitoring counting, a maximum duration of the test can be configured (in clock cycles). Each counter is set up providing an event number and the master or slave device for which the events will be counted (i.e. events are counted per core, not globally).

The CCI-400 interconnect provides a good amount of PMCs which are, by the nature of the component, very related to inter-core communication. They could be used to easily detect how a certain type of interference affects the behavior of a core. For

instance, we could test how an increase of one core's number of read requests affect the number of stalls cycles in different core.

Unfortunately, for all the architecture we know there are only 4 available counters, so measuring interferences would be a process which should be repeated for every type of interference which we want to measure, and for each different contender. This would require a very high amount of experimentation.

QoS counters can be very useful to measure interference based on priority groups, as they implicitly provide information about the cause of the interference:

- If a high priority QoS read request is stalled: it is because the queue for high priority requests if full. We solve this by increasing the size of the queue, or by reducing the amount of contenders using the high priority queue.
- If a medium priority QoS read request is stalled: there are high priority requests using the network, or the request is waiting for another medium priority QoS request to finish. Because of the LRG arbitration policy, we know the maximum amount of time the request will be blocked will be at most equal to the number of other tasks in the medium priority QoS queue.
- If a high priority QoS read request is stalled: there are high or medium priority requests using the network, or the request is waiting for another low priority QoS request to finish. Because of the LRG arbitration policy, we know the maximum amount of time the request will be blocked will be at most equal to the number of other tasks in the low priority QoS queue.
- If a read or write request is waiting for a QoS VN token, we know that contenders in the same VN are causing the block.

5 Freescale P4080

Freescale P4080 processor which hosts eight e500mc cores. e500mc [E500mc] is a superscalar processor that can issue two instructions and complete two instructions per clock cycle. E500mc cores comprise two simple instruction units (SFX0, SFX1), a multiple-cycle instruction unit (MU), a branch unit (BU), a floating-point unit (FPU), and a load/store unit (LSU).

Each core has a private L1 instruction and data cache. It also has a private L2 unified cache. The eight cores are connected through a proprietary CoreNet Fabric coherent interconnect with two shared 1MB L3 off-chip caches. Each L3 off-chip cache is connected to a distinct DDR memory controller as depicted in Figure *10*.

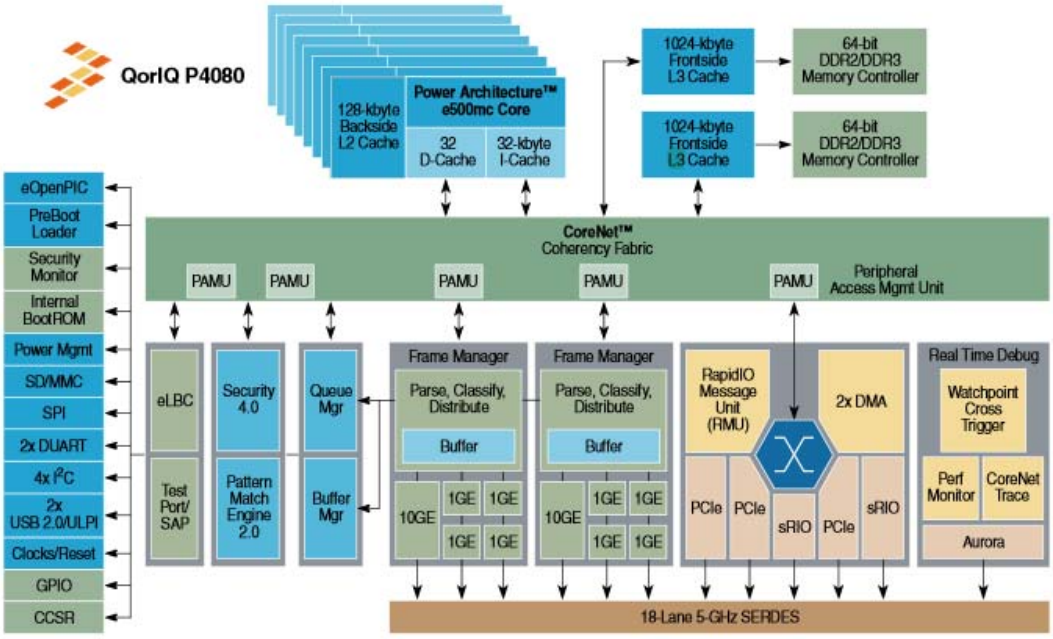

Figure *10 Block diagram of the Freescale P4080*

5.1 Performance Monitors

The performance monitor provides several performance monitor registers (PMRs). The PMR support offers dedicated core and SoC platform counters[P4080-progRef][P4080- Debug]. At the core level, the e500mc core allows us to monitor 256 different hardware events, each core being able to monitor 4 different events at a given time in 4 dedicated 32-bit registers. At the SoC level the P4080 Event Processing Unit (EPU) allows counting SoC platform events of interest.

5.1.1 Per-core monitors

The performance monitor facility provides the ability to count events and processor clocks associated with particular operations. For example, cache misses, mispredicted branches, or the number of cycles an execution unit stalls may be countable events in a particular processor implementation.

Trackable events can be grouped as follows:

- Instruction type: number of completed instructions of a certain type (branch, load, store, etc).
- Event count: Instructions completed, instructions fetched, Micro-ops decoded, etc.
- Reference count: number of referenced lines in L2, misses in L2, L2 invalidation in different coherence state, etc.
- Busy resource cycles: number of cycles in which the pipeline is stalled, the FPU input data stalls, FPU pipeline stall, MMU miss, MMU busy, Load miss with DTLB full, etc.
- Busy resource times: number of times a stalled resource has been accessed. Counts the same events as the ones described in Busy resource cycles, but instead of latency, number of times a stall is detected is counted.
- Cycle count: Processor cycles, FPU divide cycles, etc.
- Threshold exceeding event count: number of times a threshold specified in number of cycles has been exceeded for a given event, such as L1 miss, ILFB miss, interrupts, etc.

Counters are usually per unit such as the fetch unit, the L2 cache, the execution units. In this category we find events such as the number of fetches, BIU requests, Snoop requests, Snoop hits, L2cache accesses, hits (data and instr.), etc.

5.1.2 SoC monitors

While we have found information about per-core counters, we have not been able to find information about the events trackable in the EPU[2].

5.1.3 Interesting PMCs

The architecture provides two types of counters that could be useful for approximating the worst observed case and to detect interference between cores, for different types of events.

5.1.3.1 Threshold for worst observed behavior

We have found six counters that can be used to approximate worst observed behavior. This may help model the worst observed behavior, which one of the goals of the WCMCs. The following two count the number of times a threshold has been exceeded for DLFB and ILFB load miss cycles.

76: Data line fill buffer load miss cycles

77: Data line fill buffer load miss cycles

Some of the counters may be useful to approximate worst observed behavior for interrupt handling latency:

78: External input interrupt latency cycles

[2] We believe that the documentation about PMCs for the CoreNet network, L3 shared cache and beyond of the P4080 should be in the “Advanced QorIQ Debug and Performance Monitoring Reference Manual”. However it seems that this document can be accessed through an NDA.

79: Critical input interrupt latency cycles

80: External input interrupt pending latency cycles

81: Critical input interrupt pending latency cycles

Interaction between cores

There is very little support in the architecture to detect inter-core interference. We have found the following counters:

179: stwcx successes: number of times the stwcx instruction, used for locking (in conjuntion with the lwarx instruction) has successfully updated the read-modify-write lock variable.

180: stwcx unsuccessful: number of times the stwcx has failed to update the lock variable.

This is insufficient to model general case interference, as these counters only take into account a very particular case of interference: locking.

5.1.4 Summary

Inter-task interferences happen mainly in the interconnect, the L3 and the access to memory. The impossibility to find the events that can be tracked at that level, does not allow us determine to which extend SoC-level PMR can tract inter-task interferences.

Freescale PowerPC processors provide a very limited amount of PMCs, most of them inherited from single core designs. There are very few PMCs that count events originated outside of the core.

6 Aeroflex Gaisler NGMP

The NGMP is a 4 core LEON4 processor used by the ESA. It contains one or more LEON4 Statistical Unit (L4STAT). The debug driver for L4STAT provides an interface for reading and configuring the performance counters available in a L4STAT core.

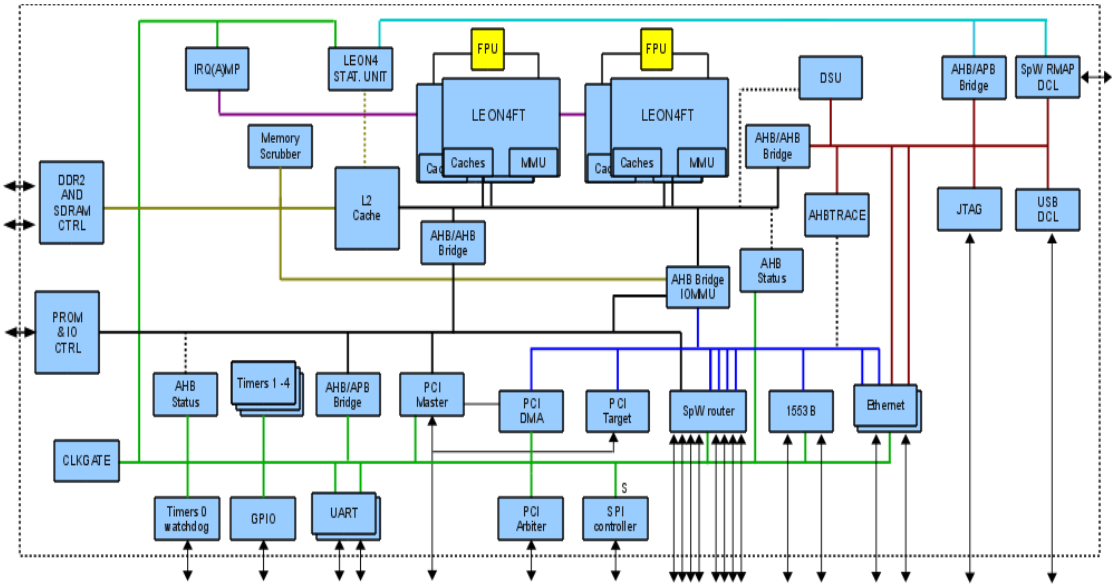

Figure *11 Block diagram of the NGMP [NGMP]*

6.1 PMC infrastructure

Each L4STAT allows configuring any available four events we want to monitor. These 4 counters are 32-bit wide and reset to 0 on overflow. Each counter has an associated control register. Both the counters and the control registers are mapped to APB address space.

6.2 Performance Monitoring Counters

The available events can be divided in three different categories, depending on the component counting the events:

- Processor events: events generated by the processor, e.g., pipeline or the L1 cache.
- AHB events: events generated by the AHB bus, e.g., AHB busy cycles or number of read accesses.
- Device specific events: events generated by other devices such as the L2 or the IOMMU.

The PMCs available in the NGMP are the following:

Table 6 Complete list of *PMCs available in the NGMP*

Processor events	AHB events	Device specific events
Instruction cache miss	AHB IDLE cycles	L2 cache hit
Instruction MMU TLB miss	AHB BUSY cycles	L2 cache miss
Instruction cache hold	AHB NON-SEQUENTIAL	L2 cache bus access
Instruction MMU hold	transfers	IOMMU cache lookup

Data cache miss Data MMU TLB miss Data cache hold Data MMU hold Data write buffer hold Total instruction count Integer instructions Floating-point unit instruction count Branch prediction miss Execution time, excluding debug mode AHB utilization (per AHB master)AHB utilization (total) Integer branches CALL instructions Regular type 2 instructions LOAD and STORE instructions LOAD instructions STORE instructions	AHB SEQUENTIAL transfers AHB read accesses AHB write accesses AHB byte accesses AHB half-word accesses AHB word accesses AHB double word accesses AHB quad word accesses AHB eight word accesses AHB waitstates AHB RETRY responses AHB SPLIT responses AHB SPLIT delay AHB bus locked	IOMMU table walk IOMMU access error/denied IOMMU access OK IOMMU access passthrough IOMMU cache/TLB miss IOMMU cache/TLB hit IOMMU cache/TLB parity error

The available counters can be classified in the following categories depending on the type of event they count:

- Busy resource cycles: the resource is unavailable because it is busy. For example AHB busy cycles
- Idle resource cycles: the resource is not being used. For example AHB idle cycles
- Cycle count: other events counting cycles. For example, CPU cycles.
- Instructions of a given type: Load, store, floating point, integer, total count.
- Event count: number of mispredictions, IOMMU errors,
- Reference count: AHB accesses L1 and L2 accesses and misses
- Maximum Count mode: see section 6.3.

The NGMP does not provide performance counters that would be suitable as WCMC. The lack of appropriate counters to model inter-core interferences has hindered research made on the NGMP. For instance, [MBENCH] could have provided more precise results if the L2 miss counter would be able to count the number of L2 misses per core, instead of system-wide misses. The PMCs provided are not prepared to identify the source of interferences either.

6.3 Maximum count mode

It is interesting noting that, if implemented by the core, the PMCs can be configured in *Maximum Count mode*. While in this mode, the counter keeps the maximum amount of time the selected event has been asserted. It is also possible in this mode to count the maximum amount of time between two event assertions. Maximum count mode may be very useful as a first step WCMC which keeps track of worst observed behaviors.

Using *Maximum Count Mode* it could be possible, for example, to count the longest burst of AHB busy cycles, or the longest amount of time the bus has been without having a read access.

Unfortunately, the availability of this counter is implementation dependant and is not available in the ML-510 board.

7 Performance Monitoring Counter classification across processors

In the current document we have classified PMCs in every processor in different categories. In this section we try to find similarities in the PMC classification across all the analyzed processors. To this end, we are presenting the categorization for each platform in a tabular form.

Intel	ARM	Freescale	IBM	NGMP
Busy resource cycles: number of cycles in which the caches are busy, a resource is unavailable, stalled core, etc.	Corelink CCI-400 stall cycles and QoS stall cycles	Busy resource cycles: number of cycles in which the pipeline is stalled, the FPU input data stalls, FPU pipeline stall, MMU miss, MMU busy, Load miss with DTLB full, etc. Busy resource times: number of times a stalled resource has been accessed. Counts the same events as the ones described in Busy resource cycles, but instead of latency, number of times a stall is detected is counted.	Number of cycles a resource is full (with this causing the stall of the processor),	Busy resource types: AHB busy cycles
			Number of cycles a resource is empty. This can be of a private	Idle resource cycles: AHB idle cycles

			resource of a shared resource. The latter meaning that none of the threads that can generate a request to that resource have done so.	
Cycle count: unhalted core cycles, unhalted thread cycles, cycles with outstanding cache misses, TLB walk duration, etc.	Cycle count: cpu cycles, bus cycles.	Cycle count: Processor cycles, FPU divide cycles, etc.		Cycle count: cpu cycles
Instructions of a type: number of retired instructions of a certain type (branch, load, store, etc).	Instructions of a type: number of retired instructions of a certain type (branch, load, store, etc).	Instructions of a type: number of completed instructions of a certain type (branch, load, store, etc).	Number of instructions of a given type	Instructions of a type: Load, store, floating point, integer, total count.
Event count: microinstructions issued, branch instructions executed, number of speculative instructions retired, number of miss-predictions, number of cache misses,	Event count: miss-predicted branches, number of exceptions, etc. Resource-specific event count: L1 write backs, number of L1/L2 refills.	Event count: Instructions completed, instructions fetched, Micro-ops decoded, etc.	Number of events of a given type (e.g. prefetch requests sent, …)	Event count: number of misspredictions, IOMMU errors,
Reference count: number of referenced lines in each MESI state, number of each level of cache references. (Note that this can be considered a subtype of the	Reference count: number of L1 accesses, bus accesses, data memory accesses, unaligned accesses to memory, etc.	Reference count: number of referenced lines in L2, misses in L2, L2 invalidation in different coherence state, etc.	Number of references to a given resource (e.g. L2 accesses, Dcache invalidates from L2	Reference count: AHB accesses L1 and L2 accesses and misses

above).	CCI-400 number of requests.			
			Quantity of data transferred	
			Stall cycles due to inter-task conflict	
Threshold exceeding event count: number of times a threshold specified in number of cycles has been exceeded for a given event. The threshold value is configured by the user.		Threshold exceeding event count: number of times a threshold specified in number of cycles has been exceeded for a given event. The threshold value is configured by the user.		Only if Maximum Count Mode is implemented
Number of outstanding requests per core: cache requests, all offcore requests, etc. in the moment of reading the counter.				

8 Discussion

This document contains a description of several commercial multi-core processors from IBM, Intel, ARM, Freescale and Aeroflex Gaisler). In particular we focus on the Performance Monitor Counter infrastructure provided by each of them.

In all the studied architectures, PMCs are used to improve average system performance by monitoring software execution, characterizing processors behavior, and/or helping system developers bring up and debug their systems. Few exceptions of counters exist that help understanding the effect of inter-task interferences. In particular, only the POWER7, Intel, and the NGMP have been identified to have PMCs that provide some information about inter-task interferences and maximum (worst) duration of a stall event (situation).

- POWER7: The only PMC we have seen that provide inter-task interference knowledge is `PMC_CMPLU_STALL_THRD`, which provides the number of cycles a task was stalled due to inter-task interferences in the SMT. Despite that, the reasons behind this stall is not provided.
- NGMP: Under the Maximum Count Mode (bit 22 of the Counter Control Register is asserted), then the counter is able to keep the maximum value the counter has achieved, or the maximum amount of time between two consecutive occurrences of the same event.
- Intel: Provides the `MEM_TRANS_RETIRED.LOAD_LATENCY` which can be configured the amount of times a memory load operations exceeds a user defined threshold. This allows the user to approximate the worst observed behavior, and to upper bound it.

However, in general, we observe lack of detailed inter-task interference PMC support. We note that some information about inter-task interference can be derived in controlled scenarios. In a first run the program under study is run in isolation recording PMCs. In a subsequent run the program under study is run again, maintaining the same input data sets as part of the workload. By subtracting the PMCs in the first run for those in the second run some inter-task interference information can be obtained. However, this complicates the analysis and requires analyzing the program with the same input data in isolation and with other tasks (deployment time), which is hard to obtain in the general case.

It is also remarkable, that only POWER7 has a CPI stack model designed by chip vendor (IBM). We consider that CPI stack provides very relevant information for the timing behaviour of an application, though for the average performance.

References

INTEL1	Intel® 64 and IA-32 Architectures Software Developer's Manual
ARMv7	ARM® Architecture Reference Manual. ARMv7-A and ARMv7-R edition
ARM-A7	Cortex™-A7 MPCore™ Revision: r0p3. Technical Reference Manual
ARM-A9	Cortex™-A9. Revision: r4p1. Technical Reference Manual
ARM-A15	Cortex™-A15 MPCore™ Revision: r3p2. Technical Reference Manual
CCI400	CoreLink™ CCI-400 Cache Coherent Interconnect. Revision: r1p1. Technical Reference Manual
ARM-A9	Cortex™-A9. Revision: r4p1. Technical Reference Manual
E500mc	e500mc Core Reference Manual. http://cache.freescale.com/files/32bit/doc/ref_manual/E500MCRM.pdf
P4080-progRef	EREF 2.0: A Programmer's Reference Manual for Freescale Power Architecture® Processors. http://cache.freescale.com/files/32bit/doc/ref_manual/EREF_RM.pdf
P4080-Debug	On-Chip Debugging of Multicore Systems. http://2008ftf.ccidnet.com/pdf/PN115.pdf
bL	Big.LITTLE Processing with ARM Cortex™-A15 & Cortex-A7
P7-PMCs	Comprehensive PMU Event Reference POWER7. Alex Mericas, Brad Elkin, Venkat Rajeev Indukuru. Release 1.0. August 30, 2011. IBM Systems and Technology Group
P7-desc	B. Sinharoy et al. IBM POWER7 multicore server processor. IBM J. Res. Dev. , 55:191–219, May 2011
P7-PMC-list	https://www.power.org/wp-content/uploads/2012/09/POWER7_PMU_Detailed_Event_Description.pdf
MBENCH	"Multicore OS Benchmark" ESA project number 4000102623
NGMP	Quad Core LEON4 SPARC V8 Processor LEON4-NGMP-QUADLION Data Sheet and User's Manual