The IBM Power Micro-architecture

Report for COMP9244: Software View of Processor Architectures

Godfrey van der Linden

2006-08-01

Abstract

The IBM PowerPC instruction set architecture and the implementations of it have powered many different computer systems. It is a second generation RISC design that incorporates many instruction extensions designed to ease the generation of quality code by modern compilers. The RISC design lends itself to scaling from very small implementations designed for embedded applications, through super–computers, to standalone desktop and server machines. Although the design is fundamentally RISC, the IBM designers have explored much of the landscape of modern computer architectures, from large super–scalar processors down to tiny single issue pipelined processor cells, suitable for use in custom designed ASICs. This paper explores the history of the architecture and some of the unusual instruction set choices that the original PowerPC designers made, before investigating some of the PowerPC implementations. Modern PowerPC processors, such as the POWER4, have very powerful and general supervisor and hypervisor mode functionality. The later sections of this paper discusses operating system design and some of the some of the issues posed by the POWER4 architecture.

1 Introduction

Continuing with IBM's long history of computer design innovation, the PowerPC architecture and the various implementations of it, such as the POWER5, are worthy representatives of their world class design. IBM lead the way with many performance innovations in the 1950s and '60s; it is interesting that so many of the technologies that they invented have been of direct use in their RISC architectures some 40 years later.

Apple Computer shocked the personal computer world when they announced their intention of switching to the Intel Core Duo line of processors. IBM and Freescale seem to be unconcerned, certainly IBM will be shipping as many of these processors as their fabricators can make attempting to satisfy the insatiable gaming console market. In a remarkable coup both MicroSoft's XBox 360 and Sony PlayStation 3 will be powered by PowerPC chips. Freescale is concentrating their design efforts with low power signal processing chips aiming for the handheld embedded market, such as mobile phones.

In this introduction we shall briefly discuss the development history of the PowerPC(PPC) architecture and a quick survey of the PPC extensions to traditional RISC architectures. In two middle sections will discuss the large mainframe desktop implementations and followed by the smaller embedded processor variations. Finally, in Section 4, issues and design features will be discussed from the viewpoint of an operating system designer.

In this document it is assumed that the reader is familiar with both modern RISC processor architectures and also with operating system design issues.

1.1 History of the PowerPC

In 1985 IBM started working on a second generation RISC architecture, known as the "AMERICA architecture", [CM90]. In 1986 the development on the RS/6000 was started, which was the first implementation of the AMERICA architecture, in February 1990, the first RS/6000 machines shipped, [IBM01] with a renamed POWER architecture. At the end of development of the RS/6000 series IBM had reduced the chip count of the CPU from eleven chips down to a single chip in the low end machines.

In the early '90s Apple and IBM were collaborating on a number of projects. During this time IBM approached Apple to collaborate on the development of a family of single–chip processors based on the POWER architecture. Soon after, Apple asked Motorola to join the collaboration to leverage Motorola's experience in manufacturing high–volume processors. This collaboration was known as the AIM[1] alliance; the result was a modification of POWER, known as the PowerPC instruction set architecture*(ISA)*. PowerPC added single–precision floating point, general register–to–register multiply and divide and removed some POWER instructions, such as MQ register based multiplies. PowerPC also added a parallel 64–bit instruction set mode.

The 1993, the POWER 2 was released and the POWER3 was released in 1997. The POWER3 was the first implementation of the full 32 and 64–bit PowerPC ISA. The POWER4 was released in 2001 and has become the basis for later development of relatively large systems. The PowerPC 970(2002) and POWER5(2005) extending the POWER4 micro-architecture.

1.2 PowerPC instruction set architecture

The PowerPC instruction set architecture is part of a group of second generation RISC ISAs that were developed in the late '80s and early '90s. This document will concentrate on those parts of the PowerPC ISA that are unconventional. In most ways the PowerPC ISA is conventionally RISC in that only load and store operations are used to access memory; that is, there are no memory operand addressing modes. However unlike many RISC instruction sets, the PowerPC has some extended instructions that are capable of modifying multiple target registers or are inherently multi-cycle in nature.

[1](A)pple, (I)BM and (M)otorola

1.2.1 Branch registers: Link and Count

The PowerPC ISA uses a dedicated link register, `lr`, to save the return address for a function call. It is the callees responsibility to save the `lr` if it isn't a leaf function. Using a special register for the link register can make the return jump faster since the hardware may not need to go through a register read pipeline stage for function returns.

The ISA also has a dedicated register for controlling loops, the count register `ctr`. This register is automatically decremented and tested against 0 by the branch conditional instructions. By using this special register the branch processor can determine the result of branch early in the pipeline.

In the PowerPC architecture the branch and instruction fetch units are closely associated, in fact he branch unit is split into two with the branch prediction unit very close — in terms of clock cycles — to the fetch unit. The `lr` and `ctr` registers can be implemented in the part of the branch unit that is near the fetch unit and the instruction set can the use these registers to implement indirect function calls. See [HP03, Appendix C] for details.

1.2.2 Load and Store extensions

Atomic update The *Load and Reserve*(`lwarx`) and *Store Conditional*(`stwcx`) instructions provide the PowerPC shared–storage atomic–update mechanism. `Lwarx` instruction loads a value from a memory location and marks that location as reserved. The subsequent `stwcx` stores a new value, if the reservation is still valid. A reservation is invalidated when another processor successfully commits a `stwcx` instruction. Using this mechanism a general atomic operation suite can be built such as *atomic increment*, *compare and swap* and *test and set*. See [WSM$^+$5b].

With update Indexed load and store instructions have a 'with update' variant, which updates the source index register with the computed effective address after the load or store is initiated. The load instruction will modify two registers; something that is unusual for RISC instruction sets. This variation is essentially a convenience to reduce the number of instructions executed inside a loop iteration. See [WSM+5a].

Multiple Another pair of convenience instructions are the load and store multiple register instructions `lmw`/`stmw`. These instructions are intended for the preamble/postamble of functions to save and restore non-volatile registers, as defined by the application binary interface (*ABI*). These instruction do not really increase program performance directly as they still take one cycle per register to implement but rather are used to increase the instruction stream density. See [WSM+5a].

String/misaligned A variation of the Load/Store Multiple, the Load/Store string instruction. These instructions load a series of words from an arbitrary byte address in to or out of multiple registers. See [WSM+5a].

Byte reversed The load/store word/halfword instructions all support a byte swapped variant, where the (half–)word at the specified location can be byte–swapped on the way to or from memory. As many of the POWER implementations have switchable endianess these instructions are used to efficiently work with cross–endian integers. See [WSM+5a].

1.2.3 Shared Storage

The PowerPC specifies a weakly consistent storage model. This means that each processor's hardware on the memory bus are not responsible for maintaining coherence. The design takes the approach that a programmer must maintain a consistent view of memory with the appropriate use of synchronisation operations in code, see section 1.7 Shared Storage of [WSM+5b].

The strength, and weakness, with the weakly consistent memory model is that the programmer is responsible for maintaining coherence across the processors by appropriate use of memory barriers. However the details are so complex and difficult to explain that the average programmer is left behind. Which means that some system programmers may utilise sophisticated synchronisation techniques internally in the operating system, but probably needs to publish quite conservative, i.e. slow, services to external non-kernel programmers.

In PowerPC ISA there are four data synchronisation instructions: `isync`, `eieio`, `lwsync` and `sync`.

`isync` *Instruction synchronise* This instruction insures that all preceding instructions are completed before the `isync` completes and stops all subsequent instructions from scheduling until completion. It also invalidates any instructions in the I-cache with respect to any outstanding instruction cache block invalidations, `icbi`, from the bus. In lock acquisition, this instruction is used immediately after the `stwcx` complete test to guarantee that no later instructions are executed speculatively until the results are known. In general `isync` can be thought of as a load barrier.

`eieio` *Enforce in–order execution of I/O* The behavior of this instruction depends on the type of caching enabled in preceding loads and stores. Memory mapped device registers are mapped cache–inhibited/guarded (see section **??**). For this type of mapped register operation, this instruction guarantees load and store ordering such that all preceding loads and stores complete to main memory before the `eieio` completes. This only effects loads and stores, if ordering of all instructions is required then the `sync` must be used. For the default cache–enabled case the instruction guarantees store ordering but its use is deprecated in favour of the `lwsync` instructions, when available, see below.

`lwsync` *Light–weight synchronise* This instruction guarantees order of store instructions, hence a memory barrier, to system memory. However it doesn't work predictably on memory mapped device memory, in which case the, much slower, `sync` should be used. This instruction precedes the atomic update for mutex unlocking and guarantees that all of the code in a critical section is complete *before* the lock is dropped. Only implemented in the POWER4 derived micro-architectures.

`sync` *Heavy–weight synchronise* When ordering against a device register is required it is necessary to issue a `sync` instruction to guarantee ordering. `Sync` extends the guarantee of `lwsync` to include a round trip to the system memory controller so that outstanding bus snoops can complete.

1.2.4 Miscellaneous instructions

The designers of the PowerPC ISA added a number of unusual instructions. These instructions seem to be provided for the use of C compilers to make some of the more difficult C semantics easier to implement.

32–bit immediate values An immediate 32–bit value is generated using an `ori rt, r0,<low>` followed by a `oris rt, rt, <high>`. The first instruction loads the low 16 bits of the value into the `rt` register and the second instruction, *or immediate shifted* shifts an 16bit immediate to the top of the register and ors it into the target register.

Rotate with mask A set of instructions that can extract and insert C bitfields in a single RISC operation. They are used to rotate bits in a word to the bottom of the register and mask a specified number of bits out. Or the inverse operation, of rotating bits and inserting them in a target based on the specified mask.

Complemented logical operations It is possible to complement one of the operands for all of the bit wise logical operations: AND, OR and XOR. The XOR complemented instruction is also known as equivalent `eqv`.

`sraw/addze` This pair of instructions — *shift right algebraic/add zero bit extended* — is used by a compiler to optimise an immediate power of two division of signed or unsigned integers as an algebraic shift operation.

`cntlz` *Count leading zeros* Very useful instruction for computing the log2 of a value. Very useful when implementing power of two allocators or logarithmic instrumentation.

Move program counter (*PC*) In the Apple dynamic library system, the program counter is required to be passed as an argument to the dynamic code patcher. This is an unfortunate design as RISC architectures rarely make the PC directly available in the ISA. Apple has been using a branch and link to the next instruction,`BCL 20, 31, $+4`, which stores the PC in the link register. The PPC970 branch predictor has been specially modified to not update the link register stack for this particular instruction.

2 POWER4 based micro-architectures

The first machine with the POWER4*(P4)* processor was released in 2001, [TDF+01]. The basic architecture has been extended a number of times for later processors, such as the PowerPC970 family and POWER5. All P4 based processors are all 64–bit processors with a massive out–of–order execution engine, that is typified by long pipelines and high clock rates.

The driving design principles for the P4 design was a combination of IBM server class machine requirements, such as: SMP optimisation, full system design approach, very high clock frequencies, reliability, availability and serviceability and finally binary compatibility with the 32 and 64–bit PowerPC ISA.

Each P4 chip has two processor cores each with their own L1 cache, a large L2 cache which is split into three independent and concurrent caches. The chip also has a dedicated intra-chip and inter-chip communication fabric to support a maximum of 32 processors in a SMP

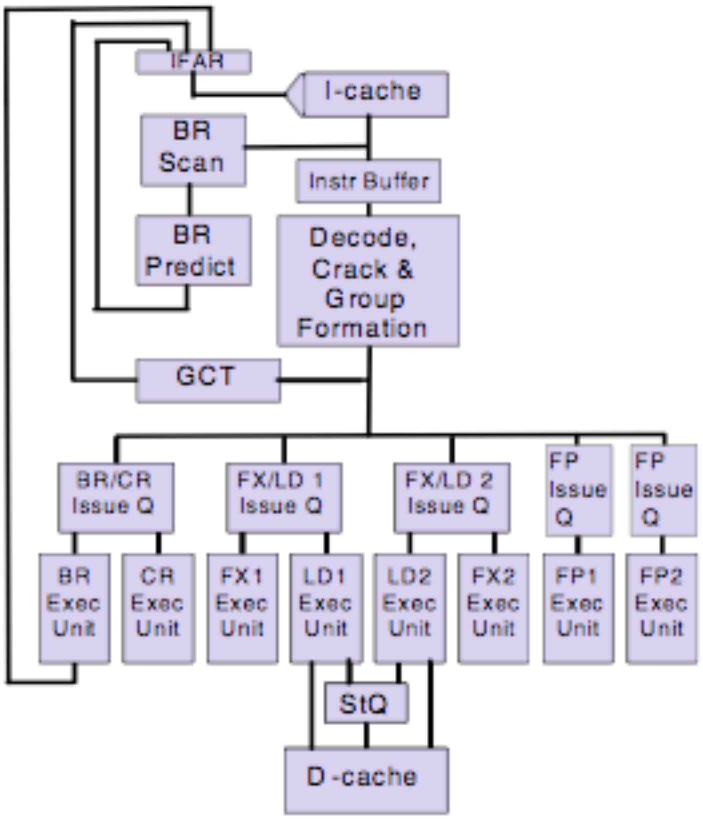

Figure 1: Block diagram of the POWER4 processor. [BBF+01]

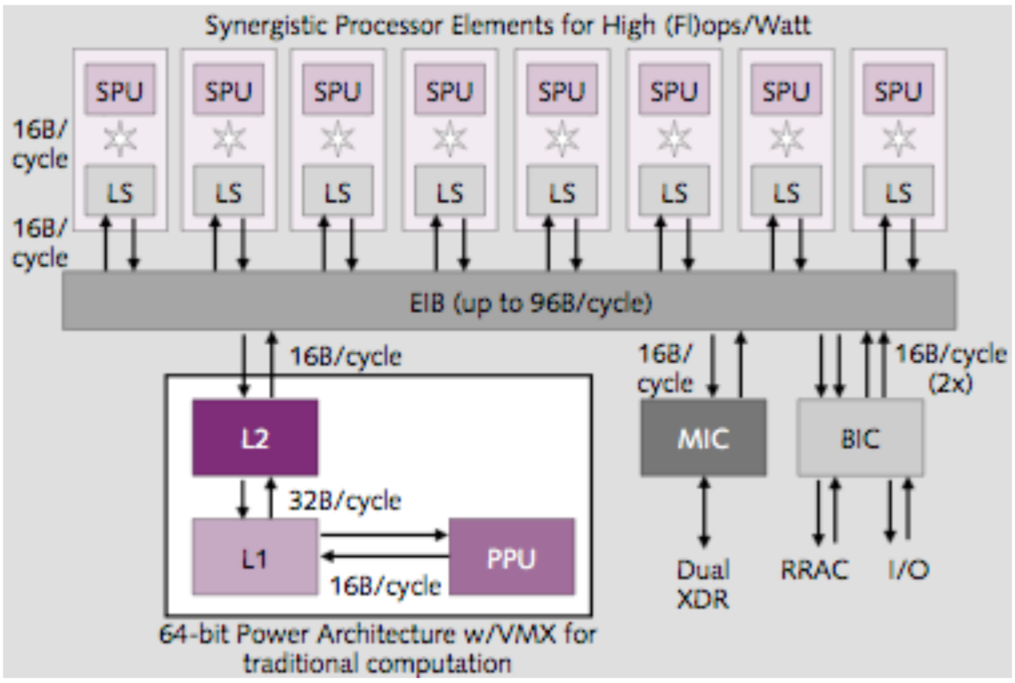

Figure 2: Block diagram of the Cell processor. Where: SPU means synergistic processor element, LS is local store, EIB is element interconnect bus, PPU is PowerPC processing unit, MIC is memory interrupt controller and BIC is I/O bus interface controller. [Kre05]

system. The fabric is used by the L2 cache to communicate to an on–chip local L3 controller and directory, with a memory controller built in.

The internal micro-architecture of the core is a speculative, superscalar, out–of–order *(OOO)* execution design, see figure 1; it can fetch, decode and crack up to eight instructions per cycle, issue up to five operations per cycle and maintain a sustained completion rate of five operations per cycle — most instructions *crack* to one operation, a few crack to more than one operation. With large register rename pools, other OOO resources and long pipelines, the P4 can have over two hundred instructions in flight. To exploit instruction level parallelism there are eight execution units each of which can have an instruction issued each cycle, though this rate can not be sustained due to the formation of five instruction groups.

To minimise the resources used to keep track of such a large number of instructions in flight, the P4 decodes and schedules instructions into *instruction groups* of five. The group is encoded such that each slot in the group can only be scheduled on specific execution unit pipelines, and the fifth slot may *only* contain a branch instruction or a no–op. This group then has rename and other OOO resources associated with it and is tracked through the entire execution pipeline. Finally when every instruction in the group has completed the entire group is committed, hence the sustained completion rate of five instructions per cycle.

With this micro-architecture's power memory bandwidth could become a bottleneck. The P4 has very large caches to minimise memory access latencies with: a 64KiB directly mapped I-cache, 32KiB two–way set associative D-cache, 3 x 480KiB (1.41MiB total) independent 4–8 way set associative L2 caches and 8–way set associative L3 cache directory for up to 32MiB of off–chip L3 cache. The L1 and L2 caches are 128b per line and the L3 cache has 512b per line. Finally the memory hardware is capable of recognising up to eight software initiated contiguous data streams and will start scheduling pre-fetch of contiguous data.

To achieve very high potential clock rates the P4 has a sixteen cycle instruction pipeline, which can cause very long miss–predicted branch stalls, so the designers introduced a very large branch prediction unit with sophisticated subroutine and other indirection logging. The P4 uses three branch history tables: the local predictor table has 16k 1–bit predictors, the global predictor table also has 16k 1–bit predictors — indexed by hashing the result of the last 11 branches with the current branch address and finally a selector table of 16k 1–bit predictors — indexed the same as the global branch table. According to [TDF+01], "This combination of branch prediction tables has been shown to produce very accurate predictions across a wide range of workload types." Certainly a large number of transistors have been dedicated to branch prediction. In addition to the branch prediction unit the P4 architecture maintains a stack of the last link register values for target prediction of return addresses. Also the POWER4 has a 32 entry direct mapped cache for count register target address prediction.

To keep the memory flowing through the caches of the large SMP systems, the PowerPC ISA only specifies a weakly consistent storage model. However a rich set of synchronisation primitives are provided for a programmer to optimise shared storage accesses. In subsection 4.3 the operating system issues of weakly consistent model will be explored more completely.

As a part of IBM's commitment to reliability, availability and serviceability, the designers have introduced into the P4 the ability to logically partition an SMP system into a number of logical subsystems. Each processor has a logical partition ID, a real mode offset register (*RMOR*), a real mode limit register (*RMLR*) and a hypervisor RMOR. These registers and a few others can only be modified in a new privileged mode called the hypervisor mode. Once the RMOR/RMLR registers are set by the hypervisor the processor will be incapable of accessing memory out of that range of physical memory. With these tools the operating system can divide a large SMP into a series of logical partitions that can be individually failed — without bringing the entire system down – until a service technician can be scheduled. This technique has other uses too, such as processor virtualisation.

2.1 PowerPC 970

The PowerPC970 family of chips are all 64–bit PowerPC ISA microprocessors, based upon the POWER4 micro-architecture with vector — single–instruction, multiple–data (*SIMD*) — instruction extensions known as *VMX*, [ppc05]. The processor is designed for desktop and low–end servers applications for uni-processors up to four way SMP system configurations.

Apple Computer was a big customer for the 970 design, which they marketed as the G5. They requested a few changes in the basic P4 design, specifically the requested support for Motorola's SIMD instruction set, support for a binary compatibility 32–bit execution mode is supported for 32–bit process support and a couple of smaller changes to support the Mac OS X application binary interface, see Move program counter in section 1.2.4 above. Apple had wanted to design a 970/G5 laptop system but the power consumed and heat dissipated presented insurmountable problems to Apple's system designers. The 2.5GHz 970MP dual core chip has a peak power consumption of 100W.

The primary limitation of the 970 over the P4 architecture are in the caching hierarchy. The 970 does not have a L3 cache and only has a single 512KiB L2 cache, though the low–power 970FX and dual–core 970MP expanded it to 1MiB. Otherwise the caches are the same structure as the P4's.

Probably the biggest architectural difference between the P4 and the 970 family is the addition of the VMX, *Vector/SIMD Multimedia eXtension*, instructions. Two new execution units were added to the architecture: the vector permutation (*VPERM*) and the vector ALU (*VALU*). The VPERM has a 19 stage pipeline. The VALU is further divided into 3 units for fixed point, complex–fixed and floating point instructions with 19, 22 and 25 pipeline stages respectively. The AltiVec/VMX ISA seems to be very highly respected in the industry in comparison to the SIMD extensions available on other processors.

2.2 POWER5 micro-architecture

The key goal of the POWER5, was to maintain both binary and structural compatibility with the POWER4, [SKT+05]. The POWER5(*P5*) is a two–way simultaneous multithreaded(*SMT*) dual–core chip that has a very similar architecture to the P4. P5 systems are capable of running up to 64 processors or 32 chips, twice the number of processors for P4s.

The P5 designers realised that the addition of SMT and 64 processors SMP would present tremendous problems to the P4 memory design where the L3 cache was between the inter–processor fabric and the memory bus. Thus when a processor core had an L3 hit it would issue a memory cycle on the processor fabric, with twice as many processors on this bus and more fetches issuing from each chip this caused an unacceptable degradation in memory performance. The P5 has moved the L3 to directly connect to the L2 controller, so that L2 fills no longer cause transactions on the fabric. In addition to moving the L3 cache, IBM designers added an on–chip memory controller to improve the speed of main memory access. These two changes greatly decrease the latency to main memory of a cache miss and also enhances system reliability by reducing system chip count.

The caches were also somewhat reorganised from the P4 architecture. The P5's L1 caches have been doubled in size by increasing the I-cache and D-caches to 2–way and 4–way respectively. The L2 cache is now a 10–way 128b line with 640KiB for each of the 3 cache slices for a total of 1.875MiB of L2 cache. The L3 directory is very different from the P4. The new 36MiB cache is divided into three slices where each slice is organised as 12MiB, 12–way set associative with 256b per line.

Too maintain structural compatibility the P5 has an identical instruction pipeline to the P4 processor. So code that is optimised to run on the P4 will still run optimally on the P5. However with the addition of SMT it should be possible to maintain higher utility of the chips execution units, the P4 usually achieved on 25% utilisation of each of its eight execution units. Other resources where also increased to better support SMT for instance the various register files where increased in size to support much more register renaming. In IBM benchmarks these design changes have led to a 40%

improvement in processor throughput over a large test base.

To deal with potential resource starvation for a single thread the processor supports eight priority levels per thread. These are used to control the relative number of decode cycles each thread gets and hence the number of instructions executed. Also the processor monitors various recourse such as the global completion table and the load miss queue, to detect significant resource starvation. In the case that resource starvation is found the offending resource hunger thread is throttled back by lowering its priority.

The P5 is capable of running in a single threaded(ST) mode as some applications that are execution unit or bus bandwidth limited. In this mode all of the additional resources needed to support SMT are devoted to the execution of a single thread, for instance the full 120 rename registers become available to a single thread. The processor can dynamically switch between ST and SMT execution modes.

Chip power is becoming an important metric and such a complex chip as the P5, with its 237 million transistors, would be very power hunger. To offset this power consumption problem, the P5 chip has implemented a dynamic power management system that dynamically shuts–down inactive parts of the chip. Each unit can be clock gated into a low power state, on a cycle by cycle bases, with no performance loss.

2.3 POWER6 micro-architecture — what is known

According to reports from the 2006 International Solid–State Circuits Conference in San Francisco, IBM presented many papers on the up–coming POWER6. It is expected to be shipping in 2H2007 and will clock in the 4–4.5 GHz range, with speeds of 5.6GHz achieved in the lab. Simultaneously IBM claims to have kept a lid on power consumption; "Despite the speeds, it will have a lower power density than in some chips found in today's desktops" said Bernard Meyerson, chief technologist of IBM's Systems and Technology Group, [Ass06, Feb. 07].

Although we do not yet know many details of the POWER6 micro-architecture, we suspect that many of the technologies that have been explored for the very high speed Cell broadband engine will make their way into the POWER6, see subsection 3.3.

3 PowerPC based micro-architectures

In this section the smaller embedded and set–top implementations of the PowerPC ISA will be described. In general most of these processors continue to use the 32–bit instruction set. The PowerPC 440 will be discussed first, followed by the Blue Gene Compute Chip and the Cell Broadband Engine.

3.1 PowerPC 440

The PowerPC 440 (*PPC440*) core is a relatively high–performance, superscalar processor core intended for embedded applications. The PPC 440 core is also available as part of the IBM ASIC library of processor cores. The target market is the usual list of embedded applications such as digital cameras, laser printers, switches and network cards, [ppc99]. In an unexpected twist the PPC440 core has become the basis of the of the Blue Gene/L Compute Chip, see section 3.2 below.

The core features a two–way superscalar design, with 3 execution pipelines and OOO issue, execution and completion. The pipeline itself is 7 stages long. In addition to the standard 32–bit RISC PowerPC ISA some 24 DSP operations are also available including a single–cycle throughput 16x16+32 to 32 multiply accumulate instruction.

The basic design is quite modular and the library core con be configured with a number of different subsystems including 0 - 64KiB caches. The processor is very low power with 2.5mW/MHz or about 1.4W at 555MHz. At the peak of 555MHz, the processor delivers 1000 MIPS using the Dhrystone 2.1 benchmark.

The 3 execution units are: a load/store pipeline, simple integer pipeline and complex integer pipeline. The pipeline itself has seven stages: instruction fetch, pre-decode, decode/issue, read registers, execution stage 1/EA address compute, execution stage 2/data cache access and register writeback. The lack of renaming registers is a serious limitation for exploiting any ILP that may be available.

The PPC440 has separate instruction and data caches, factory configurable up to 64KiB and are highly associative, the 64KiB cache is 128–way set associative, the associatively varies with cache size. With the high associatively advanced cache partitioning is possible. The caches can be separated into normal, transient and locked regions, where: normal regions have traditional cache replacement, transient regions are used temporarily then not used again — typical of data streams and locked regions is used for code that is not to be cast out of the cache.

As with most embedded system designs, the PPC440 is designed with a number of external interfaces:

Processor local bus (*PLB*) Three separate PLB interfaces are used to access system resources: one for instruction fetches, one for data reads and the last fro data writes. Each PLB controller is a 128–bit bus master.

Device control register bus (*DCR*) The DCR bus is a configuration bus for components external to the core. The DCR bus is used to manage status and configuration registers and reduces PLB traffic. System resources on the DCR bus are protected to some extent as they are not part of the system memory map.

Auxiliary processor unit (*APU*) Instructions within the PowerPC ISA have been reserved for APUs to execute instructions in the stream. The APU interface provides a dual–issue pipeline design and can use a full 128–bit load/store path to the D-cache. Some uses of the APU is to implement a full PowerPC Floating Point Unit, multimedia macros, DSP or other custom functions.

External interrupt control (*EIC*) The EIC can extend the PPC440 interrupt system to external interrupt systems. These external interrupts are level sensitive and can be wired to critical or non-critical interrupts internally.

Debug There are two debugging interfaces on the PPC440 core, the JTAG and instruction trace ports.

3.2 Blue Gene/L Supercomputer

The fully populated IBM Blue Gene/L system installed at Lawrence Livermore National Laboratories in Livermore, CA topped the November 2005 TOP500 Supercomputer list with a sustained 280.6 TFlops. At the time no other system exceeded 100 TFlop/s and the Blue Gene/L is expected to top the next few TOP500 lists. The Blue Gene/L system is an unorthodox entry into the supercomputer field, in that it uses slow unsophisticated processors as its workhorse compute engine. Each Blue Gene/L compute chip is a 4MiB DRAM with a pair of 700MHz PowerPC440 processor cores sharing the DRAM as a L3 cache. Each core has 2 FPUs, 32KiB I and D-caches, a very small 2KiB L2 cache and 16KiB of local scratch SRAM. They also share some I/O related subsystems for inter-processor communication. Each chip can be thought of as a 4MiB DRAM cache with a small compute core in one corner of it, [IJEBP+05].

In Blue Gene/L two chips are packaged with 2x512MiB of RAM onto a single processor board. These boards are collected together into a processor 'card'. These cards are collected into a cabinet with 1024 nodes, or 2048 processors, drawing a total of 29kW and has a performance of 2.9/5.8 TFlops. A fully configured Blue Gene/L system has 64k processing nodes, draws a tiny 1.8MW and achieves an astounding 76 MFlops/W, [cF05].

In comparison the ASC Purple using 10240 POWER5 processors draws 7.5MW and achieves 63.3TFlops or 8.4 MFlops/W. The ASC Purple machine is typical of the high–power compute node based high performance systems and also typical has required quite carefully designed cooling systems. Of course the choice of slow processing nodes means that only problems that are massively parallel will benefit, in fact it is easy to conceive of workloads that would run slower on the Blue Gene/L than more traditional high–performance computers even though they do not compare to the Blue Gene/L in processing power.

3.3 Cell broadband engine

The Cell broadband engine is the product of a collaboration between IBM, Sony and Toshiba. The design goal of the chip is to support the next generation of Sony PlayStation gaming console. Sony required substantial performance of the design and it seems that IBM has achieved this goal, the chip has a design frequency of 4.0GHz and can achieve a startling, though theoretical, peak single–precision performance of 192GFlops at 3.0GHz[2]. The memory interface has also been redesigned and is capable of an I/O bandwidth of some 25.6GB/s, provided by two Rambus XDR memory controllers, [Kre05].

A Cell chip is IBM's first instantiation of a architecture known as the '*Broadband Processor Architecture*' (*BPA*). The new architecture was designed for specific workloads such as: cryptography, graphics and lighting, physics, fast–Fourier, matrix computations and other scientific tasks. This instantiation is a multi–processor on a chip, with a completely redesigned 64–bit PowerPC core and eight '*Synergistic Processor Elements*(*SPEs*)', which are SIMD signal processors. It would seem that IBM had an additional design goal of maximising floating point performance per watt even though the architecture runs at very high frequency. The block diagram, figure 2, from [Kre05], shows the structure of the Cell chip; as you can see the Cell is a multi–SIMD processor, or a MIMD.

With the Cell IBM has reworked the 64–bit PowerPC core, which is the controlling processor or (*PPU*). It implements the 64–bit PPC instruction set, though internally the PPU has a much simpler design than the super–scalar POWER4 designs with their complex multi–functional–unit out–of–order issue architecture. Instead the Cell implements two thread SMT with dual, in–order, instruction issue to a three computation functional units. Like the POWER4, the Cell was designed for a very fast clock, some 4.0GHz for the Cell and has a long, 21 stage, pipeline. However as the Cell is not super–scalar to any extent IBM has limited instruction hazard and cache miss penalties: a branch misprediction takes eight cycles to unwind and data loads have a four cycle latency. Like the 970 the Cell's PPU does implement the complete VMX/AltiVec SIMD instructions set. The PPU is intended to run the operating system and control the work flow through the SPUs and the new memory flow controller.

Each SPE has a vector processing unit, known in IBM inimitable three letter acronym way as an *SPU*, and 256KiB of local static RAM. The design of the SPU The SPUs, are derived and 'inspired' from and by the VMX/AltiVec implementation and Sony PlayStation 2's 'Emotion Engine'. The vector instruction set mostly traditional with perhaps an unusual number of registers. The SPU supports 128 128–bit wide registers. With this large register set and the fast local store the designers have not implemented caches in the processor. The designer hope that a carefully designed algorithm should not need a cache. The local store is populated using dedicated SPE local DMA controllers. The DMA engines can access both main memory and other processors local stores. Although the DMA transactions are coherent, once the data is cached in an SPE's local store its coherence is not maintained, leaving coherence and synchronisation to the programmer, see section 4.3.

Intra-chip communications use the *element interconnect bus* (*EIB*). The EIB consists of four busses, two clockwise and two anticlockwise. Each bus is sixteen bytes wide and data clocks every two processor clocks, or a theoretical maximum bandwidth of 128BiB/second. The EIB is organised as a token ring where each processor on the ring is no more than five steps away from any other processor. However, the bus will automatically partition itself if necessary when two non-overlapping processors are communicating. For example if processor A is moving data to B and C is communicating with C they can share the same processor bus as B's DMA engine does not transmit packets targeted for itself on the bus.

In conclusion, the Cell's potential is enormous but its unconventional architecture will make severe demands on tool and software designers to try to make use of this potential.

[2]IBM has not stated the power consumption of the Cell processor yet, but Microprocessor Report reckons that it will be 80W or peak 2400MFlop/W

4 Software considerations for operating systems on the POWER4

While researching the POWER architecture several features of the design suggested themselves as being of interest or concern to operating system designers. In this section some of those areas will be explored. The 32–bit PowerPC440 based architectures are traditional pipelined RISC processors and do not implement many of the features here discussed. Hence this section will concern itself with the 64–bit POWER family, such as the POWER4, PowerPC970 and Cell processors.

In subsection 4.1 we will discuss the enhanced memory management options available using the new two level effective–address to real–address translation using segment tables. Subsection 4.2, will explore a new API for context dependant pre-fetching of cache data or alternatively structuring data in such a way that automatic pre–fetching is triggered. Subsection 4.3, explores the implications of the POWER families weakly consistent memory models. Subsection 4.4, covers issues resulting from large branch misprediction penalties and interactions with typical operating system designs. The hypervisor facilities of the POWER architecture are the subject of subsection 4.5. Subsection 4.6, discusses the internal implementation of the POWER architecture and how structuring code carefully may greatly enhance performance. Finally in subsection 4.7, we will briefly introduce the reader to some of the performance tools available to the reader for analysing PowerPC970 performance.

4.1 Segmented Memory

A 64–bit virtual address spaces is too huge to maintain a traditional 4KiB page table for each process. The 64–bit PowerPC architecture specifies a virtual storage model where the user address space (*effective address* or *EA*) is a subset of an OS maintained virtual memory address space. An effective address maps into the virtual address space using 256MiB segments, where each segment maps an effective address ID *ESID* and effective address to specify a virtual memory address. The segment table is backed by a small fully–associative SLB cache and the virtual memory has the traditional TLB. Finally there are small cache's for performing direct EA to real address conversion.

This structure allows the operating system to maintain a single much larger 65–bit virtual address space which combines segment table provided virtual specifier IDs (*VSIDs*) with effective address to specify a real address. The designers intended the segmentation to allow for large portions of the virtual address space to be shared between different user tasks. For instance to mapping large shared libraries, an OS kernel, a micro-kernel and rootserver combination or even large shared memory pools created and manipulated directly by client processes through such APIs as `mmap`. A single segment can map single use effective address space into a VSID, but an entire segment table will map into many different VSID specified parts of the VM. It should be possible to limit the number of TLB interactions using this design.

The VM page table itself has been enhanced to support multiple page sizes, on POWER4 they can be 4KiB and 16MiB, but the Cell can support 64KiB, 1MiB and 64MiB pages. Use of these new page sizes should be explored for the operating system as a whole, though it is obvious that the kernel/rootserver and large user shared libraries can make immediate use of them.

The POWER family has had a long standing problem with slow TLB interactions. Any operating system engineer designing an implementation on the POWER architecture should spend considerable time characterising the performance of the many levels of look–aside buffers implemented in this segmented memory model.

We believe that it will be worthwhile to experiment with the new feature to determine if TLB sharing between processes is practical and performant. With judicious uses of the VSID identifier an operating system may be designed that will only infrequently require a TLB invalidation.

4.2 Cache pre-fetching

An argument could be made [3] that an operating system is a massive multiplexer/demultiplexer engine. Good system call design usually limits the number of entry points into an OS and each entry point is as general as possible. Hardware interrupts are also demultiplexed and routed to appropriate drivers. Traditionally all of this demultiplexing is implemented using C's very efficient pointer to function. Once a demultiplex has been decided the layer of code probably has some context that needs to be re-established. If the decision point could also arrange for the context to be pre-loaded, later cache misses may be avoided.

We propose a new API allows a system engineer to register not only a pointer to function and data context pointer, but also the context size with a parent data provider. The parent could decide whether it implemented the decision or not, but it is responsible for passing the request onto the ultimate decision maker. The decision maker would associate the set of addresses with a particular result and pre-load context data. These pre-load requests would be made before the data is requested by the normal stream of instructions, and thus should shorten the data cache delays.

Consider as a concrete example of the use of this API a PCI device driver. At the time when the driver registers its interrupt handler and data context, it would also associate the context length with the interrupt handler. The interrupt demultiplexer would store a handler function, a context data pointer and the context's length. When an interrupt for the card occurs, the interrupt controller would quickly spin over the data context once per cache line touching the entire context into data cache, before branching to the handler function. By the time the indirect function call is resolved at the context data should be closer to the caches than would otherwise occur.

It would appear at first glance that the proposed API constitutes an unnecessary layer violation, after all each interrupt handler could easily pre-load its own data directly. But that means that all well designed interrupt handlers would need to add this functionality. If every client needs the functionality then good design demands that the functionality must be pushed up to the service provider. This mechanism of associating a function pointer with the complete context data could be made a general service provided by the operating system and coded as efficiently as possible.

Many processors that conform to the later 64–bit POWER design also have hardware detected data and instruction streaming. When the cache controller notices that a series of cache–misses have been contiguous, controller automatically issues pre-fetch requests. With this facility in mind we should investigate mechanisms to identify associated data and attempt to group all of the data required sequentially. This is easy within one module but if we can identify opportunities across several API layers the rewards may be significant. To identify such cases we may be able to run an instrumented version of software to identify separate layer data structures, by profiling perhaps, then try to arrange this memory to live in a single allocated data structure. Interestingly the hardware can identify cache–missing streams in either direction, which means that both up calls and down calls through a stack would benefit.

Extensive profiling would be required to prove that the pre-fetching technology discussed in this section do not have a detrimental impact on the system due to an increase in worthless cache invalidation. Further research could define probability bounds where a cache–line that is pre-fetched is sufficiently likely to be used.

4.3 Shared Storage

The POWER architecture specifies a weakly consistent storage model. Later processors do extensive instruction and write buffer reordering. Data–hazards are dealt with by the internal instruction dispatcher and the cache snooping will deal with cache–line inconsistencies. However store re-ordering is independent of the caching specified on the memory, the CPU never guarantee's the order of stores.

Although it is possible to describe this is-

[3] But not here!

sue in only a few sentences, the implications of it are wide ranging and performance critical. A system designer, considering POWER, should devote some time experimenting with the weakly consistent storage model, researching interactions and performance impact on a particular platform/memory controller combination. This research should be carefully documented and distilled, ideally in such a way that an average programmer can grasp enough of the subtleties to seek expert advice when required.

Given weakly consistent memory, the programmer is expected to implement sufficient synchronisation between different entities on a modern computer system; for instance between processes that can run on different cores but on the same caches, on different processors on the same memory, or on a NUMA machine. Communicating with devices on the system bus also presents different but related synchronisation issues.

Optimising synchronising performance is the real issue, IBM engineers having ducked consistency, have left consistency to the programmer and system designers. If a naive, but safe, consistency model is used, such as using the `sync` instruction exclusively it would utterly destroy the performance of the system; the POWER core would be reduced to little better than early non-pipelined CPU performance without even a cache as all memory operations must complete. Hence it is natural for system designers to choose the weakest, that is highest performance, of the many synchronisation options provided be the architecture as possible. However the experience at Apple Computer (conversations with Apple [Eng06]) with these barrier operations `lwsync`, `sync`, `eieio` and the instruction order barrier `isync` has been painful. Each revision of the CPU and memory controller combination has had subtly different semantics. Apple spent significant amounts of time and effort to achieve a high–performance, consistent, set of barrier operations. The task is so complex that it is unlikely even now that Apple achieved either cross processor generation portability or optimal performance.

It may be impossible to design a simple set of rules that the general programming community could follow and that are optimal. To certain of a safe, near optimal consistency solution would require a team of application, operating system and platform hardware engineers. The solution developed with would only be narrowly optimal, another revision of the processor or memory system would probably invalidate the synchronisation and may even lead to inconsistent data.

At Apple Computer, we had the Mac OS X kernel running for some six months, before we discovered the cause of a rare bug where two threads could both modify lock protected data. We knew that each thread took the lock, yet the corruption still occurred. It was discovered that write re-ordering could unlock a lock *before* some memory stores are completed, thus defeating the lock. At that time we decided to use the `sync` operation before mutex unlock. `sync` before unlocking a mutex. Years later, after gaining more experience, we determined that a `sync` was unnecessary and the higher performance `lwsync` instruction would be sufficient.

4.4 Indirect function performance

The design goal of achieving a very high processor frequency, required the POWER4 to implement very long instruction pipelines. This long pipelines implies that branch misprediction will be expensive. They are, take twelve cycles to resolve, [BBF+01]. As the POWER4 architecture is super–scalar, with up to ten functional units, it is possible that a hundred instructions must be unrolled[4]. To limit these penalties the branch prediction unit, on POWER4 derived architectures, is one of the most sophisticated such units implemented on any RISC processor, see section 2.

PowerPC implements pointers to functions by using the count register(CTR) with the `bcctr` instruction. The CTR and, related, link register(LR) can be located in the instruction fetch unit. The fetch unit can use these registers to determine its path through the instruction stream. On the POWER4, these architectural registers are backed by sixteen physical rename registers and so some care must be

[4]This does not occur due to the instruction grouping, see subsection 4.6.

taken by the fetch unit to use the appropriate physical register.

In subsection 4.2, we noted that much of the demultiplexing performed by an operating system is implemented using C function pointers. The performance of these instructions is critical. Software designers on the POWER4 platforms should be aware of the very high cost of branch misprediction, however they may not know that *all* `bcctr` indirect branches use the branch predictor's target address predictor. The nature of demultiplexing will often result in high miss rates on branch target predictions. This is a shame; if the designers required allowed the actual value to be used rather than the predicted value it is often possible to order instructions so as to make the target available early. Consider that many indirect functions have two or more arguments. Just marshalling the arguments would give sufficient time for the CTR value to become available.

As the CTR register cache is direct mapped and indexed by the instruction address of the branch the indirect function call will often have different target addresses and will be mispredicted. This is especially problematic for C++ virtual function calls which all indirect through a class's virtual table.

Given the deep pipeline in the POWER4 micro-architecture, mispredicted branches can take twelve cycles to resolve; it would be better to use an extension to the `bcctr` instruction that *disables* the branch target prediction. Although we will pay the twelve cycle penalty and stall the pipeline, we are no worse off and the processor has not spent a lot of energy computing values that it will then discard.

4.5 POWER family hypervisor

IBM were probably the first company to explore virtualisation, their 370 series of computers which introduced the concept of totally independent virtual machines in the VM/370 operating system in 1972. In recent years Intel processors have also become so powerful that it has become practical to run multiple instances of an operating system on one computer. The POWER4 derived hardware implements virtualisation enhancements, known collectively as hypervisor functions. A hypervisor is a sort of a small 'operating system' system. In other words it does not deal with user tasks directly but rather controls and services various operating systems on the platform, [AAB+05].

With the POWER4 IBM introduced the first phase of their virtualisation solution, which culminated in the POWER5. The POWER4 only had limited hypervisor functionality, in the form of logical partitioning of a system. A system's memory could be partitioned into smaller contiguous chunks. The processor's *memory management unit (MMU)* enforces a mapping of real addresses onto this partition. While a process is executing in a partition, the MMU is incapable of generating real addresses that are not in the partition. The hypervisor kept track execution state and switched the MMU mapping when switching to a new state in a different partition. This required a total TLB invalidation, as the previous virtual memory mapping would be useless. Hence the performance of switching partitions would probably be very slow. A scheduler which has a concept of processor/partition affinity is required. Or, alternatively, a multiprocessor system could be mapped such that processors are assigned to logical partitions and only occasionally would an administrator switch the configuration around.

With the POWER5 the limitations of the POWER4's virtualisation have been addressed. The design criteria was to enable efficient and flexible virtualisation to hypervisor aware operating systems, this design technique is sometimes known as *paravirtualisation*. The client operating systems be assigned exclusive access to particular PCI slots or can share I/O resources with other clients.

IBM has an implementation of a hypervisor on their iServer and pServer systems. They needed to design a solution that could efficiently share I/O systems across multiple logical partitions. A traditional design would locate drivers in hypervisor space, but that would lead to a less robust hypervisor and could potentially compromise security. It was decided to dedicate a logical partition to hardware drivers and to cede sufficient privilege to this partition to allow it to issue remote logical partition DMAs. The approach was to allow a partition to instantiate virtual I/O adapters in its local partition. The virtual adapter has such definitions as DMA windows, interrupts and other adapter information such as real physical hardware would have. In addition hypervisor operations allows partitions to pass their virtual adapter details to the I/O service in a secure way, which passes a capability to the I/O partition to perform the physical I/O.

The design that IBM developed has many similarities with a modern micro-kernel such as L4. L4 could be a hypervisor that controls the various logical partitions. Certain enhancements would have to be made to L4 to support the POWER5 completely, such as address translation for remote partition DMA and scheduling with processor/partition affinity. A root-server task would be used to control the assignment of processor, memory and I/O resources to each logical partition all of which would be mediated by the micro-kernel.

4.6 Instruction grouping

Large super–scalar processors such as the POWER4 derived designs have some unexpected similarities with VLIW and EPIC architectures, in that they both benefit greatly from appropriate instruction scheduling. Otherwise it is unlikely that the processor could discover sufficient instruction level parallelism to keep the functional units as saturated. Although the typical operating system engineers will not be impacted by such details, they should be aware that a compiler's code generation quality will impact significantly the performance of their OS. Most operating systems have a small amount of assembler for particularly time critical functions, and hand-coding these optimally is difficult..

POWER4 derived processors have up to ten functional units, see figure 1, which are fed by several issues queues. If every operation is treated seperately then processor control would become too complex and expensive. The IBM designers decided to combine operations into groups of five, with a single group issued on each cycle and then it is tracked until entirely completed. We have been careful to refer to operations rather than instructions, as some PowerPC instructions need to be broken up into multiple operations, for example the store multiple registers instructions is issued as a separate operation per register.

A group can not be formed arbitrarily, only certain operation types may appear in each of the five slots, this simplifies the routing of operations among the different issue queues. As only a single group can be issued per cycle, it is very important for instructions to be scheduled in such a way as to maximise the utility of each group, however these instructions must also conform to the group formation limitations. For a compiler to do a good job of instruction scheduling it will need to model the complex instruction cracking and group formation architecture.

This is very difficult as the processor is so complex and is probably out of reach for hand coding of large amounts of assembler. In section 4.7 we shall discuss some of the tools available to the programmer for optimising their code. With the aid of the performance tools it should be possible for an assembler engineer to iteratively approach optimal code, in much the same way that [GCC+05] did for the Itanium, we suspect that very similar improvements can be achieved over the code generated by the GCC group of compilers.

IBM are reported to have made extensive changes to the GCC compilers to better support POWER instruction scheduling but they discovered that the GCC compiler's internal model was not powerful enough to accurately model the POWER family. The IBM XL family of compilers do not suffer from this limitation. If high performance is critical to your application then the use of the IBM provided compilers will generate much better code than GCC and is highly recommended.

4.7 PowerPC performance tools

As just mentioned the easiest way of gaining high performance is to use one of the IBM XL family of compilers, these compilers accurately model the instruction break up and operation grouping on POWER processors. IBM have also published an excellent book, [BBF+01], that describes the architecture of the processors and its caches, and has many code examples optimising performance for particular tasks. One of the nicest features of this book is a small table of optimal implementations for certain numerical kernels. This table can allow a developer to determine where they can achieve real performance improvements in their algorithm. Apple has provided a less detailed guide to optimising for the PowerPC970 in [App].

In addition to the standard Unix tools such as `prof` and `gprof` to determine hotspots in applications, an operating system may choose to publish the PowerPC performance registers. AIX makes them available through the `pmcount` infrastructure. This allows a program to record details of its own performance for a programmer to analyse. Unfortunately it takes a lot of skill to interpret these raw numbers in a meaningful way.

Apple has developed a highly regarded suite of software, *Computer Hardware Understanding Developer (CHUD)*, which they provide free as part of the Apple development suite. It is used to optimise applications running on the PowerPC970 or 'G5' processors that power their machines. CHUD has an extension to the MacOSX kernel that collects statistics in the kernel's privileged space. A performance monitoring application interfaces with the CHUD extension to start and stop data collection. These data are then combined with fully symbolled object code by the Shark application to identify problematic lines of source code. Shark can even be configured to collect statistics within the kernel itself and provided with kernel symbols it can be used to identify expensive kernel operations — although it finds that the kernel seems to spend a lot of time in the enable interupt function much to the confusion to non-kernel programmers. Other performance application in the suite can collect complicated call graphs.

Apple has also made a cycle accurate G5 simulator, [App], known as Amber, available for developers. Although not useful to identify problem code in a large application, once an area of concern has been identified it is invaluable in analysing and optimising especially critical sections of code. A combination of Amber and hand–coded assembler will allow a talented engineer to iteratively approach an optimal solution for the POWER4 derived architectures.

It should be noted that optimal code is processor architecture dependant, changing such things as the size of the caches or the internal architecture will invalidate most of the optimisations. For instance highly performant PowerPC7400 code — the previous generation of chips that Apple called the G4' — will not perform optimally on the 970. This means that it is impossible for a single app. to be optimal over many generations of a CPU, despite the fact that the CPUs are binary compatible with each other.

5 Conclusion

Like other RISC platforms, the PowerPC architecture has been found to be flexible and adaptable; from the small PowerPC440 ASIC core up to the massive super–scalar POWER5 and everything in between. The POWER4 derived architectures are particularly well designed, it is interesting that IBM managed to compress what is essentially a mainframe processor and ship it as the PowerPC970 microprocessor for use in desktop computers.

Given the 20 year history of the architecture, it is likely that IBM will continue to evolve the PowerPC for the foreseeable future. It is ironic that Apple computer has dropped the PowerPC at the same time as MicroSoft and Sony have committed to developing their respective third generation of gaming consoles based with it.

The instruction set architecture has stood up to the test of time remarkably well, IBM's designers obviously realised that C would be the most important programming language to support and tailored the instruction set to suit this language well. The bit masking instructions demonstrate their commitment to optimising C, these instructions are obviously tailor made for the C bit-field manipulation.

If the PowerPC architecture has fault it would be the weakly consistent memory model. Even talented programmers get lost in the complexity and design trade-offs required to write highly performant and correct code. This is a failure of design, the decision to leave synchronisation up to software requires an extensive and difficult training period for the average engineer. If, as Hennesey claims, it is possible to develop a high performance processor without store re-ordering and strongly consistent memory this would be ideal. In IBM's defence however it should be possible to achieve greater code efficiency with their flexible synchronisation model, if a software engineer knows precisely the context under which the code will run.

It is interesting that IBM is continuing with the development of very high clock frequencies processors after their main desktop computer competitors Intel and AMD have both retreated from it and are trying to change the marketing ground to performance/Watt. IBM will be shipping the Cell broadband engine chips with an astounding 4.0GHz clock frequency, and should be available early in the third quarter of this year, 2006. The POWER6 is also due to be released soon and, given the performance of the Cell, it seem likely that IBM will achieve their very aggressive goal of 4.5 to 5.0GHz frequencies. Even at this incredibly high frequency the Cell processor should only draw some 80 Watts, which is reasonable for a desktop system. Cooling the PlayStation 3 will be quite a dilemma for the Sony designers.

A final observation on all modern processors, is the trend towards very complicated super–scalar processors with deep pipelines and other VLIW style limitations. The importance of a good compiler can not be minimised, scheduling instructions to avoid pipeline stalls has never been so difficult, nor so essential. Code that is extensively scheduled, however, will not be optimal on different instantiations of the hardware even with 100% instruction set compatibility. Logically, a developer should optimise for the slowest processor, as the slowest processor needs the most help compared to later, faster processors. However this usually leaves some of the new generation hardware capability unexploited, not something that product marketers ever like selling. The only alternative is unpalatable, to ship multiple object code with each image scheduled for a particular processor implementation. There are obvious limits to this approach, should a developer release different images depending on each cache's size too?

References

[AAB+05] W. J. Armstrong, R. L. Arndt, D. C. Boutcher, R. G. Kovacs, D. Larson, K. A. Lucke, N. Nayar, and R. C. Swanberg. Advanced virtualization capabilities of POWER5 systems. *IBM Journal of Research and Development*, 49(4/5):523–532, July 2005.

[App] Apple Developer Connection. G5 performance programming.

[Ass06] Associated Press. IBM unveils super-fast microprocessor, 2006.

[BBF+01] Steve Behling, Ron Bell, Peter Farrell, Holger Holthoff, Frank O'Connell, and Will Weir. *The POWER4 Processor Introduction and Tuning Guide.* IBM Redbooks, 2001.

[cF05] Wu chun Feng. The importance of being low power in high performance computing. *CTWatch Quarterly*, 1(3), August 2005.

[CM90] John Cocke and V. Markstein. The evolution of RlSC technology at IBM. *IBM Journal of Research and Development*, 34(1):4–11, 1990.

[Eng06] Apple CoreOS Engineers. Conversations with Apple CoreOS engineers over an eight year period. Private communications, 1997-2006.

[GCC+05] Charles Gray, Matthew Chapman, Peter Chubb, David Mosberger-Tang, and Gernot Heiser. Itanium—a system implementor's tale. In *Proceedings of the USENIX Annual Technical Conference*, 2005.

[HP03] John L. Hennessy and David A. Patterson. *Computer Architecture — A Quantitative Approach*, chapter Appendix-C, pages C1–C44. Morgan Kaufmann, third edition, 2003.

[IBM01] IBM. A brief hstory of RISC, the IBM RS/6000 and the IBM eServer pSeries. IBM Archives 2416RS01, IBM, 2001.

[IJEBP$^+$05] S. S. Iyer, Jr. J. E. Barth, P. C. Parries, J. P. Norum, J. P. Rice, L. R. Logan, and D. Hoyniak. Embedded dram: Technology platform for the blue gene/l chip, 2005.

[Kre05] Kevin Krewell. Cell moves into the limelight. *Microprocessor Reports*, February 2005.

[ppc99] *The PowerPC 440 Core.* IBM Microelectronics Division, Research Triangle Park, NC, 1999.

[ppc05] *IBM PowerPC 970FX RISC Microprocessor User's Manual.* IBM, version 1.6 edition, December 2005.

[SKT$^+$05] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. POWER5 system microarchitecture, 2005.

[TDF$^+$01] Joel M. Tendler, Steve Dodson, Steve Fields, Hung Le, and Balaram Sinharoy. POWER4 system microarchitecture. White paper, IBM Server Group, 2001.

[WSM$^+$5a] Joe Wetzel, Ed Silha, Cathy May, Brad Frey, Junichi Furukawa, and Giles Frazier. *PowerPC User Instruction Set Architecture Book 1.* IBM, version 2.02 edition, January 2005a.

[WSM$^+$5b] Joe Wetzel, Ed Silha, Cathy May, Brad Frey, Junichi Furukawa, and Giles Frazier. *PowerPC Virtual Environment Architecture Book II.* IBM, version 2.02 edition, January 2005b.