ECE 404: Reading Assignment

(Paper Summaries)

By

Raj Parihar
Summary

Thrust for higher performance drove the architects beyond the CPI less than one which was considered a hard limit once for a while. This new avatar of microarchitecture, capable of executing more than one instruction per cycle, is popularly known as superscalar architecture. Tomasulo’s algorithm is the heart of many of the superscalar architectures and the key is to allow the instructions to flow in out-of-order manner in various stages of pipeline. Dynamic instruction scheduling allows the out-of-order execution and resolves the data dependencies by stalling the dependent instructions until the operands are not ready.

Most of the pipelined stages in superscalar are out-of-order however the completion, also called commit, should be in-order and true data dependencies along with control dependencies must be obeyed in execution stage. Another key point is that fetch and issue unit should be wider than other stages because due to dependencies not all instructions would be able to execute in same cycle. Role of branch predictors are important in order to know the right place from where the next set of instructions should be fetched.

Another care which is taken is to choose appropriate execution window and possibility of branch instruction in the middle is minimized. Instruction renaming removes the WAW and WAR dependencies. Non-blocking memory hierarchy, read bypassing write, write buffers are some of the other optimizations which are incorporated into superscalar architecture. Major concerns are binary compatibility which is taken care by keeping the same ISA, precise interrupts requires the storing of checkpoints and multiple stages of checkpoints.

The limit of available ILP would result into diminished returns from superscalar architectures. Another challenge is that complexity of data and control path increases as the number of simultaneously issued instructions increases. Due to frequent branches 8 is considered to be the limit for width of superscalar processors. One school of thought believes that VLIW would gain importance because the responsibility to generate the independent instruction stream is on the software which can do a better job. Another school of thought conceives that generation of dynamic streams from static would expose more parallelism.

In general, superscalar has become a well known and proven method to implement high performance microprocessors and it is embraced by many vendors similar to pipelining.
Title: “Alternative Implementation of Two-Level Adaptive Branch Prediction”
By: T. Yeh and Y. Patt

Summary

Branches change the normal flow of control in unexpected ways which interrupts the normal fetch and issue operation of instructions because it takes considerable number of cycles to resolve the outcome of branches. In order to maintain the high throughput branch prediction in modern high performance microarchitecture has become absolute MUST.

There are two aspects of branch predictions. First is to determine the outcome (Taken or Not taken) of branch and if taken then the knowledge of target address. A static branch prediction scheme relies on the information incorporated in prior to runtime whereas the dynamic (adaptive) branch prediction logic tries to extract the information and predicts the behavior of branches in run-time.

Branches outcome are not purely random and in fact they do exhibit some sort of temporal and spatial co-relation. The behavior of previous branches and the patterns they exhibit are used to predict the behavior of new branches. If the prediction is incorrect corrective measures are taken and this information is incorporated into adaptive predictors.

Most of the adaptive predictors use two piece of information. The history of last k branches and their specific behaviors is taken into account in order to implement the high performance branch prediction logic. The reason why dynamic branch prediction beats static is because the kind of data appears in run time is very much different from the sample (trace) data which is used for profiling. In addition to outcome, the branch address is cached with history table entry and updated in run-time.

In two-level branch prediction, first level takes care of branch address and second level Different patterns. Three flavors of two level predictors are

1. Global History Register with a Global Pattern History Table
2. Per-Address Branch History with a Global Pattern History Table
3. Per-Address Branch History with Per-Address Pattern History Table

Above three implementations also exhibit the tradeoff between performance and storage/ hardware required. Before two-level predictors, best predictors gave 94.4% accuracy whereas two-level predictors pushed it to 97%. The 3% was still substantial miss rate and current version of simplescalar gives 98.8% hit rate.
Title: “Combining Branch Predictor”
By: S. McFarling

Summary

Processor performance depends upon exploitation of ILP which is mainly restricted due to presence of conditional branches. Various predictors do well in specific cases but no predictor beats others in all the cases. The idea, presented in this report, is to choose the one which performs better than others for a particular case and keep track of it. In addition to keep the history of branches, in this approach the history of predictors is also kept which is used to make a wise decision about the predictor selection. This technique beats the previous well known by a margin of 1% and reduces the miss rate to less than 2%.

Bi-model predictor is simplest kind of predictor which uses the branch address and n-bit saturating counter based Pattern Table to predict the outcome. If miss predicted the Pattern Table entry is corrected by incorporating the actual outcome. In such predictors, the prediction accuracy is function of size of pattern table and the maximum achievable hit rate is 93 – 94%. Local branch predictors add one more level of hardware which is known as History Table in addition to n-bit saturation counter based Pattern Table. Saturation counter can be replaced by various state machines which are described into YEH 1992 paper. These predictors are good for local branches and give around 97.1% of accuracy.

Global branch prediction uses a Global History Register which keeps track of previous branches. It is found that branches do exhibit some sort of co-relation which is exploited in order to predict the behavior by use of GHR. Merging the branch address information with previous branch outcome information (GHR content) further enhances the predictions. Two possible ways to combine the branch address with GHR are either concatenate them or XOR them. Second option is generally preferred because it doesn’t increases the size of history table tremendously.

Combining the above mentioned branch predictors into one yields even better accuracy. The idea is to choose the predictor which gave maximum hit rate for a particular branch. A tournament predictor is used which based on branch address selects the most appropriate predictor. Combined branch prediction results into 98.1% accuracy.
Summary

This paper discusses the IBM’s POWER4 processor microarchitecture and techniques to build small, mid-scale SMP machines using POWER4 chips as basic building blocks. POWER4 microarchitecture has focus on high speed execution by breaking (cracking) the instructions into groups and executing them in program order as fast as possible – with speculation as well. Two levels of caches along with directory for L3 cache are on-chip whereas L3 memory array is external to the chip. Most of the buses run at the half of the CPU clock speed and scale accordingly. For chip-to-chip communication POWER4 provides 4, 16-byte wide ports and total bandwidth of ~38 GBps whereas multiple boards can be connected using 2, 8-byte ports with the bandwidth of 9.6 GBps. Four POWER4 chips can be connected in ring to make an 8-way SMP and 4 such SMP can be connected further in a ring which is 32-way SMP. L1 implements sort of point to point coherence whereas L2 uses snoopy protocol. Fabric controller is an important unit which is mainly responsible for communication between L3 and L2, and also between L3 and CPUs.

Strengths

Paper discusses the processor microarchitecture in great detail – every tiny bit is discussed. From system perspective it discusses the way to build SMP machines using POWER4 as basic elements. Future roadmap and design objectives are also discussed briefly. All the hardware to implement cache coherence is provided with chip which minimizes the overall cost as well.

Weak (Not-so-strong) Points

The discussion is more hardware oriented and very less detail of software is discussed yet it claims to discuss the system architecture. The discussion on system interconnects is not as precise and focused as it is in processor microarchitecture. Paper doesn’t provide the explicit figures of various bandwidths such as L2 BW, Memory BW and Chip-to-chip BW.

Future Work

The discussion on building SMP machine can add more details. Paper doesn’t even mention that the chip-to-chip interconnection is ring or bus which can be overcome by discussing couple of suitable topologies to build SMP machines. Some more explicit discussion on system architecture and software aspects such as consistency models would enable readers to understand it better.
Summary

The microprocessor discussed in the paper has a 20-entry instruction issue queue and a scheduler which can schedule 4-instructions per cycle to four execution units which are arranged in two clusters with completely replicated register file. In order to minimize the inter-cluster latency (which is 1 cycle in this case) the issue logic tries to schedule the instructions to the cluster which is going to produce the result as well.

There are five major components of the datapath described in this paper.

- The Register Scoreboard logic tracks data dependencies for 80 registers and is split in half to reduce circuit delays.
- The Request logic stores execution unit assignment information and combines it with the two register request signals.
- The Arbiter contains two pick-oldest-two arbiters operating in parallel to choose four instructions each cycle for execution.
- The Valid logic maintains a valid bit for each instruction and de-asserts one of the request lines when invalid.
- Finally, the Update logic accepts new valid bits each cycle and produces MUX selects which compact the instructions in the queue.

Some of the key aspects of designing high-speed microprocessor, discussed in the paper, are as following:

- Specific implementation of instruction queue which minimizes the cross-cluster latency problem and maintains very high frequency should be used. This sort of implementation would ensure improved performance in systems where execution units are clustered together and use dedicated (or replicated) resources.
- The Arbiter, described in the paper, picks the oldest instructions that can issue each cycle with a novel cascaded priority encoder.
- Compaction of the oldest instructions to the bottom of the queue each cycle simplified the arbitration circuits at an acceptable complexity and area cost.

The issue queue allows any of the 20 instructions to issue to either cluster, which is a 10% performance improvement on SPECINT95 benchmarks over a queue that statically assigns instructions to a single cluster. A previous out-of-order instruction queue allowed instructions to issue to only one execution unit, as was physically determined by the instruction’s location in the queue.
Summary

MIPS R10K is a four-way superscalar processor which fetches 4 instructions and decodes them in single cycle. It executes the instructions speculatively beyond branches with a four-entry branch stack. Execution of instructions is out-of-order of nature. Register renaming logic renames the registers in order to resolve the false dependencies i.e. WAW and WAR. In order to map architectural registers into physical register a map table is used which maintains the dependencies chain. Busy-bit table keeps track of free and busy registers. Exceptions are not precise due to out-of-order execution. However, for precise exceptions can be achieved by use of in-order graduation.

Five independent pipelined execution units include a non-blocking load/store unit (younger loads are allowed to serve under the older loads), dual 64-bit integer ALU unit, 64-bit floating-point unit, a pipelined adder and multiplier with 2-cycle latency each. The integer register file was 64 bits wide and contained 64 entries, of which 32 were architectural registers and 32 were rename registers used to implement register renaming. The register file had seven read ports and three write ports.

Memory sub-system is non-blocking of nature which means the hits are served under the misses. Level-1 INS and DATA cache is of 32KB each and two-way-associative. An external, two-way-associative secondary cache is also provided.

MIPS IV is a 64-bit architecture, but the R10000 did not implement the entire physical or virtual address to reduce cost. Instead, it has a 40-bit physical address and a 44-bit virtual address, thus it is capable of addressing 1 TB of physical memory and 16 TB of virtual memory.
Title: “Alpha 21264/ EV67 Microprocessor Hardware Reference Manual”

Summary
The Alpha architecture is a 64-bit load and store RISC architecture designed with particular emphasis on speed, multiple instruction issue, multiple processors, and software migration from many operating systems. The 21264/EV67 microprocessor is a superscalar pipelined processor. The 21264/EV67 can issue four Alpha instructions in a single cycle, thereby minimizing the average cycles per instruction (CPI). The branch predictor is composed of three units: the local, global, and choice predictors.

Other 21264/EV67 features include:
- A peak instruction execution rate of four times the CPU clock frequency.
- Two onchip, high-throughput pipelined floating-point units, capable of executing VAX and IEEE floating-point data types.
- An onchip, virtually-indexed, physically-tagged dual-read-ported, 64KB data cache.
- An onchip, 8-entry victim data buffer.
- An onchip, 32-entry load queue. An onchip, 32-entry store queue.
- An onchip, duplicate tag array used to maintain level 2 cache coherency.
- A 64-bit data bus with onchip parity and error correction code (ECC) support.

Pipeline consists of 7 stages as following.

Stage 0 — Instruction Fetch
The branch predictor uses a branch history algorithm to predict a branch instruction target address. Up to four aligned instructions are fetched from the Icache, in program order.

Stage 1 — Instruction Slot
The Ibox maps four instructions per cycle from the 64KB 2-way set-predict Icache. Instructions are mapped in order, executed dynamically, but are retired in order.

Stage 2 — Map
Instructions are sent from the Icache to the integer and floating-point register maps during the slot stage and register renaming is performed during the map stage.

Stage 3 — Issue
The 20-entry integer issue queue (IQ) issues instructions at the rate of four per cycle. The 15-entry floating-point issue queue (FQ) issues floating-point operate instructions, conditional branch instructions, and store instructions, at the rate of two per cycle.

Stage 4 — Register Read
Instructions issued from the issue queues read their operands from the integer and floating-point register files and receive bypass data.

Stage 5 — Execute
The Ebox and Fbox pipelines begin execution.

Stage 6 — Dcache Access
Memory reference instructions access the Dcache and data translation buffers. Normally load instructions access the tag and data arrays while store instructions only access the tag arrays.
Summary

Alpha21364 microprocessor integrates an Alpha 21264 core, 1.75 MB of L2 cache, two memory controllers and cache coherence hardware to implement directory-based protocol in single die. One of the key goals was to meet the requirement of communication demands which often occurs in many memory- and I/O- intensive applications.

Integrated router runs at the same clock speed as the Alpha21264 which is 1.2 GHz. Alpha21364 can sustain high network bandwidth -- 70% to 90% of the peak bandwidth which is 22.4 GB/s. 128 Alpha21364 chips can be connected together in a two-dimensional torus network to build a medium scale multiprocessor.

In order to implement the cache coherence directory-based protocol is used. Directory protocol implements separate virtual channels for different kind of packets which avoids the deadlocks and improves the performance in general. In order to avoid the deadlocks in routing three separate virtual channels – adaptive, VC0 and VC1 are used.

Each flit is 39 bit wide and per flit ECC provides the better reliability. One flit can be transferred in one cycle. Alpha21364 supports seven different packet classes – Request Class, Forward Class, Block response class, non-block response class, Write IO class, Read IO class and Special class. These packets could be either of 1, 2, 3 or 18, 19 flits long. Packets with more number of flits are the one which try to transfer the whole cache line or blocks. Alpha21361 implements virtual cut-through routing in which flits of a packet proceed through multiple routers until the header flit gets blocked at a router.
Summary

Consistency models bridge the gap between expected behavior by programmers and actual behavior supported in a system. In general, consistency models present tradeoffs between programmability, portability and performance. Simplest of all, Sequential consistency is natural extension of uni-processor programs but restricts many of the optimizations i.e. write buffers with bypassing, overlapping writes, non-blocking reads etc. One of the important tasks in any consistency models is to figure out as when a WRITE should be considered complete. Cache coherent and system with generic inter-connect network present additional degree of complexity to consistency models. By relaxing program order (W-R, W-W, and R-R/W to different locations) and write atomicity we move to relaxed consistency models. This relaxation comes at a cost of increased program complexity which requires programmers to understand the lower level details in order to maintain the “correctness”. Any relaxation in write atomicity should provide the safety nets i.e. fence instructions in order to ensure the correctness of program with system optimizations.

Strengths

Variety and complexity in consistency models is one of the challenges to understand them in order to implement the optimizations which don’t violate the specified model. This tutorial explains such details in concise and efficient manner. Two views – system-centric and programmers’ perspective – presented in the tutorial complement each other very well. Role of compilers in maintaining the “correctness” and still incorporating the “optimizations using re-ordering” is well explained for various consistency models. Programming language support for parallel program is discussed in precise manner which points out the required constructs needed to maintain the “correctness” and achieve the higher “performance”.

Weak (Not-so-strong) Points

Paper assumes that reader is familiar with cache-coherence and doesn’t explain the coherence in any detail. Another aspect which is not touched is the “optimizations possible” in sequential consistency model. Paper does talk about the way sequential consistency puts constraints to performance but no optimizations for system with sequential consistency models are discussed.

Future Work

One of the important issues left opened is to decide which model is best. It is pointed out that one of the disadvantages with relaxed consistency is increased programming complexity. It would be interesting to figure out the ways to alleviate this increased complexity.
Summary

This paper presents the architecture of the AlphaServer GS320, a cache coherent non-uniform memory access multiprocessor, which is targeted at medium scale multiprocessing with 32 to 64 nodes. Snoopy-protocol is efficient in terms of performance but doesn’t scale well for medium and high scale systems. On the other hand directory-protocol scales well but common transaction flow is burdened from rare corner cases. GS320 incorporates a number of innovative techniques for improving directory-based protocols and efficiently implement consistency models. One of the key features of the design is that it exploits the extra ordering property of switches to implement the directory-based protocol.

Strengths

AlphaServer GS320 tries to achieve the best-of-both-worlds by tailoring a directory-based protocol to eliminate inefficiencies associated with existing designs and to exploit the limited scale of the target systems. In addition to general implementation the solution provides elegant solution to deadlock, livelock, starvation and fairness issues.

Future Work

One of the desired things which can be done is to remove the inefficiency from the generic directory-based protocol for larger systems.
Title: “The Sun Fireplane System Interconnect”
By: Alan Charlesworth

Summary

This paper discusses the forth generation Sun’s system interconnect – Sun Fireplane – which is primarily intended to build large scale SMP machines. Fireplane employs 2-level of cache coherence protocol – snoopy based implementation to achieve the low latency in local accesses and directory based protocol to achieve higher bandwidth across the network. One of the notable characteristic of system is that it uses separate data and address networks to reduce the network contention and avoids deadlocks. To alleviate the high latency of critical path in address request it provides multiple address channels (virtual channels) and higher bandwidth for data transfer is achieved by wider buses. Small scale SMP (< 24 nodes) implement single snoopy domain and for larger SMP implementation multiple snoopy domains are connected together using directory based protocol. In snoopy domain, the physical network is ordered point-to-point, as oppose to bus, which provides the low latency and yet preserves the serialization property. Snoopy protocol uses MEOSI protocol and directory utilizes MSI protocol. For any operation the states of both the level should be same or else there is extra latency involved in changing the status of particular coherence layer. Maximum data bandwidth is 172 GBps whereas BW per address bus is 9.6 GBps.

Strengths

Discussion incorporates the coherence implementation in detail and various possible coherence requests and their responses are descried in precise manner. Paper also presents all the benchmark results and explicit figures for various bandwidths and latencies of the system.

Weak (Not-so-strong) Points

Paper focuses mainly on system interconnect rather than processor microarchitecture. Any system architecture discussion is not complete without the processor microarchitecture.

Future Work

Memory subsystem is an important part of SMP machines and some detail about it would enable readers to have better understanding of overall system.
Summary

The main idea presented in this paper is to achieve the high performance by decoupling the “Operand Access” and “Execution”. As oppose to array processors, where software does the most of the coordination and synchronization, in Decoupled architecture the hardware implements the decoupling which removes the burden of programmers and also improve the performance by avoiding the software overhead.

The Decoupled Architecture separates its processing into two parts: Access to memory to fetch operands and store results, and operand execution to produce the results. Decoupled architecture is separated into two major functional units, each with its own instructions stream. The communication between the two streams happens through the queues. In addition to get high performance significant memory communication can be hide by use of decoupled architecture.
Title: “The SGI Origin: A ccNUMA Highly Scalable Server”
By: James Laudon et al.

Summary

SGI Origin is MIPS R10K based large scale multiprocessor which is scalable up to 512 nodes. It is a non-uniform memory access (NUMA) based cache coherent machine which was manufactured by Silicon graphics Inc. In SGI Origin, each node consists of 2 identical R10K processors are connected to common 4 GB main memory and its directory memory. The architecture supports 1024 processors, 1 TB of main memory and the nodes can be connected together via any scalable interconnect network.

The Origin system employs SPIDER routers to create a bristled fat hypercube interconnect topology. The network topology is bristled in that two nodes are connected to a single router instead of one. For 32 nodes the system interconnect topology is hypercube based and beyond 64 processors, a hierarchical fat hypercube is employed.

Cache-coherence is similar to DASH machine with several optimizations. First, the clean – exclusive (CEX) processor cache state is fully supported which allows efficient execution of read-modify-write. Secondly, the full support of upgrade requests which move a line from a shared to exclusive without the BW and latency overhead. Origin’s protocol is fully insensitive to network ordering. It uses more sophisticated network deadlock scheme than DASH.

The key differences from other DSM based machines are high scalability, high modularity with a low-entry point and incremental costs. A bristled fat hypercube network is used to provide high bi-section bandwidth, low-latency interconnect.
Summary

Cache coherence schemes tackle the problems of maintaining the consistent copy of shared data in shared-memory multiprocessors. A typical cache coherence implementation could fully rely on software or hardware or combination of both.

Two most common hardware based protocols are snoopy protocol and directory based protocols which rely on a certain cache coherence policy. Two HW based policies are write-invalidate and write-update. Write-Invalidate policy maintains consistency of multiple copies in the following way: Read request are carried out locally if a copy of the block exists. When a processor updates a block, all other copies in system are invalidated. The write-update policy maintains consistency differently. Instead of invalidating all the copies it updates them.

Snoopy protocols mainly used for small size networks and hardly scalable for larger systems. This kind of protocol is used mainly to reduce the bus traffic, with a secondary goal to reduce the average memory access time. However, due to lot of misses there could be a huge traffic in network because of multiple invalidations. The directory based protocol tries to minimize the intense traffic which is due to misses and invalidations in snoopy protocol. Instead of local caches, in addition to them directory maintains the useful information about a block. In order to perform the read and write the communication between requester and destination Home node (the directory which maintains the information) is also involved.

In general, write-invalidate protocol perform better than write-update protocol by saving the bandwidth of network for multiple invalidation case. In directory based protocol, for larger systems the directory overhead to maintain the information increases and ultimately becomes bottleneck. As an alternative of the some software schemes have been proposed which are very cost-effective for larger systems.
Title: “Multiscalar Processors”
By: Gurinder S. Sohi et al.

Summary

Multiscalar tries to exploit the fine-grain parallelism by use of combination of hardware and software. Multiscalar processor divides a program into tasks which are distributed to a number of parallel processing units reside within a processor complex. The processor complex uses multiple program counters to sequence through the different part of program simultaneously. This ensures the execution of multiple instructions in single cycle. The appearance of a single logical register file is maintained with a copy in each parallel processing unit.

The objective of the non-sequential walk of the control flow graph (generated by compilers) taken by a multiprocessor is to establish a large and accurate dynamic window of instructions from which independent instructions can be extracted and scheduled for parallel execution. To maintain the sequential appearance two strategies are employed. First, each processing unit adheres to sequential execution semantics for the task assigned to it. Secondly, a loose sequential order over the collection of processing units, which in turn imposes a sequential order on the tasks.

The performance improvements from Multiscalar are expected to increase as the compiler evolves. The reason for this is that evolved compilers would be able to extract more level of ILP that are far beyond the reach of current generation.