Instruction Level Parallelism

Introduction

- Instruction Level Parallelism
-
- Instruction level parallelism is obtained primarily in two ways in uniprocessors: through pipelining and through keeping multiple functional units busy executing multiple instructions at the same time. When a pipeline is extended in length beyond the normal five or six stages (e.g., I-Fetch, Decode/Dispatch, Execute, D-fetch, Writeback), then it may be called Super pipelined. If a processor executes more than one instruction at a time, it may be called Superscalar. These two techniques can be combined into a Super-Super architecture.

- Super pipeline

- Given a pipeline stage time T, it may be possible to execute at a higher rate by starting operations at intervals of T/n. This can be accomplished in two ways:

- Further divide each of the pipeline stages into n sub stages.

- Provide n pipelines that are overlapped.

- The first approach requires faster logic and the ability to subdivide the stages into segments with uniform latency. It may also require more complex inter-stage interlocking and stall-restart logic.

- The second approach could be viewed in a sense as staggered superscalar operation, and has associated with it all of the same requirements except that instructions and data can be fetched with a slight offset in time. In addition, inter-pipeline interlocking is more difficult to manage because of the sub-clock period differences in timing between the pipelines.

- Even so, staggered clock pipelines may be necessary with superscalar designs in the future, in order to reduce peak power and corresponding power-supply induced noise. Alternatively, designs may be forced to shift to a balanced mode of circuit operation in which logic transitions are balanced by reverse transitions -- a technique used in the Cray supercomputers that resulted in the computer presenting a pure DC load to the power supply, and greatly reduced noise in the system.

- Inevitably, superpipelining is limited by the speed of logic, and the frequency of unpredictable branches. Stage time cannot productively grow shorter than the interstage latch time, and so this is a limit for the number of stages.

- The MIPS R4000 is sometimes called a superpipelined machine, although its 8 stages really only split the I-fetch and D-fetch stages of the pipe and add a Tag Check stage. Nonetheless, the extra stages enable it to operate with higher throughput. The UltraSPARC's 9-stage pipe definitely qualifies it as a superpipelined machine, and in fact it is a Super-Super design because of its superscalar issue. The Pentium 4 splits the pipeline into 20 stages to enable increased clock rate. The benefit of such extensive pipelining is really only gained for very regular applications such as graphics. On more irregular applications, there is little performance advantage.

Super scalar

- Superscalar processing has its origins in the Cray-designed CDC supercomputers, in which multiple functional units are kept busy by multiple instructions. The architecture community today tries to justify the use of the Superscalar buzzword, and the resulting connotation that superscalar implementations of today are somehow novel, by pointing out that the Cray designs could issue just one instruction at a time whereas "true Superscalar" machines can issue multiple instructions at once. In fact, the CDC machines could pack as many as 4 instructions in a word at once, and these were fetched together and dispatched via a pipeline. Given the technology of the time, this configuration was fast enough to keep the functional units busy without outpacing the instruction memory.

- Current technology has enabled, and at the same time created the need to issue instructions in parallel. As execution pipelines have approached the limits of speed, parallel execution has been required to improve performance. As this requires greater fetch rates from memory, which hasn't accelerated comparably, it has become necessary to fetch instructions in parallel -- fetching serially and pipelining their dispatch can no longer keep multiple functional units busy. At the same time, the movement of the L1 instruction cache onto the chip has permitted designers to fetch a cache line in parallel with little cost.
-
- In some cases superscalar machines still employ a single fetch-decode-dispatch pipe that drives all of the units. For example, the UltraSPARC splits execution after the third stage of a unified pipeline. However, it is becoming more common to have multiple fetch-decode-dispatch pipes feeding the functional units.

- The choice of approach depends on tradeoffs of the average execute time vs. the speed with which instructions can be issued. For example, if execution averages several cycles, and the number of functional units is small, then a single pipe may be able to keep the units utilized.

- When the number of functional units grows large and/or their execution time approaches the issue time, then multiple issue pipes may be necessary.

- Having multiple issue pipes requires
- being able to fetch instructions for that many pipes at once

- inter-pipeline interlocking

- reordering of instructions for multiple interlocked pipelines

- multiple write-back stages

- multiport D-cache and/or register file, and/or functionally split register file

- Reordering may be either static (compiler) or dynamic (using hardware lookahead). It can be difficult to combine the two approaches because the compiler may not be able to predict the actions of the hardware reordering mechanism.
-
- Superscalar operation is limited by the number of independent operations that can be extracted from an instruction stream. It has been shown in early studies on simpler processor models, that this is limited, mostly by branches, to a small number (<10, typically about 4). More recent work has shown that, with speculative execution and aggressive branch prediction, higher levels may be achievable. On certain highly regular codes, the level of parallelism may be quite high (around 50). Of course, such highly regular codes are just as amenable to other forms of parallel processing that can be employed more directly, and are also the exception rather than the rule. Current thinking is that about 6-way instruction level parallelism for a typical program mix may be the natural limit, with 4-way being likely for integer codes. Potential ILP may be three times this, but it will be very difficult to exploit even a majority of this parallelism. Nonetheless, obtaining a factor of 4 to 6 boost in performance is quite significant, especially as processor speeds approach their limits.

- Going beyond a single instruction stream and allowing multiple tasks (or threads) to operate at the same time can enable greater system throughput. Because these are naturally independent at the fine-grained level, we can select instructions from different streams to fill pipeline slots that would otherwise go vacant in the case of issuing from a single thread. In turn, this makes it useful to add more functional units. We shall further explore these multithreaded architectures later in the course.
-
- Hardware Support for Superscalar Operation
-
- There are two basic hardware techniques that are used to manage the simultaneous execution of multiple instructions on multiple functional units: Scoreboarding and reservation stations. Scoreboarding originated in the Cray-designed CDC-6600 in 1964, and reservation stations first appeared in the IBM 360/91 in 1967, as designed by Tomasulo.

Score Board

- A scoreboard is a centralized table that keeps track of the instructions to be performed and the available resources and issues the instructions to the functional units when everything is ready for them to proceed. As the instructions execute, dependences are checked and execution is stalled as necessary to ensure that in-order semantics are preserved. Out of order execution is possible, but is limited by the size of the scoreboard and the execution rules. The scoreboard can be thought of as preceding dispatch, but it also controls execution after the issue. In a scoreboarded system, the results can be forwarded directly to their destination register (as long as there are no write after read hazards, in which case their execution is stalled), rather than having to proceed to a final write-back stage.

- In the CDC scoreboard, each register has a matching Result Register Designator that indicates which functional unit will write a result into it. The fact that only one functional unit can be designated for writing to a register at a time ensures that WAW dependences cannot occur. Each functional unit also has a corresponding set of Entry-Operand Register Designators that indicate what register will hold each operand, whether the value is valid (or pending) and if it is pending, what functional unit will produce it (to facilitate forwarding). None of the operands is released to a functional unit until they are all valid, precluding RAW dependences. In addition , the scoreboard stalls any functional unit whose result would write a register that is still listed as an Entry-Operand to a functional unit that is waiting for an operand or is busy, thus avoiding WAR violations. An instruction is only allowed to issue if its specified functional unit is free and its result register is not reserved by another functional unit that has not yet completed.

Reservation Stations

- The reservation station approach releases instructions directly to a pool of buffers associated with their intended functional units (if more than one unit of a particular type is present, then the units may share a single station). The reservation stations are a distributed resource, rather than being centralized, and can be thought of as following dispatch. A reservation is a record consisting of an instruction and its requirements to execute -- its operands as specified by their sources and destination and bits indicating when valid values are available for the sources. The instruction is released to the functional unit when its requirements are satisfied, but it is important to note that satisfaction doesn't require an operand to actually be in a register -- it can be forwarded to the reservation station for immediate release or to be buffered (see below) for later release. Thus, the reservation station's influence on execution can be thought of as more implicit and data dependent than the explicit control exercised by the scoreboard.

Register Renaming

- The storage of operands resulting from instructions that completed out of order is done through renaming of the registers. There are two mechanisms commonly used for renaming. One is to assign physical registers from a free pool to the logical registers as they are identified in an instruction stream. A lookup table is then used to map the logical register references to their physical assignments. Usually the pool is larger than the logical register set to allow for temporary buffering of results that are computed but not yet ready to write back. Thus, the processor must keep track of a larger set of register names than the instruction set architecture specifies. When the pool is empty, instruction issue stalls.
-
- The other mechanism is to keep the traditional association of logical and physical registers, but then provide additional buffers either associated with the reservation stations or kept in a central location. In either case, each of these "reorder buffers" is associated with a given instruction, and its contents (once computed) can be used in forwarding operations as long as the instruction has not completed.
-
- When an instruction reaches the point that it may complete in a manner that preserves sequential semantics, then its reservation station is freed and its result appears in the logical register that was originally specified. This is done either by renaming the temporary register to be one of the logical registers, or by transferring the contents of the reorder buffer to the appropriate physical register.

Out of Order Issue

- To enable out-of-order dispatch of instructions to the pipelines, we must provide at least two reservation stations per pipe that are available for issue at once. An alternative would be to rearrange instructions in the prefetch buffer, but without knowing the status of the pipes, it would be difficult to make such a reordering effective. By providing multiple reservation stations, however, we can continue issuing instructions to pipes, even though an instruction may be stalled while awaiting resources. Then, whatever instruction is ready first can enter the pipe and execute. At the far end of the pipeline, the out-of-order instruction must wait to be retired in the proper order. This necessitates a mechanism for keeping track of the proper order of instructions (note that dependences alone cannot guarantee that instructions will be properly reordered when they complete).

- One scheme that is simple, but reasonably effective is to artificially create a dependence between instructions. For example, lets say that we are about to dispatch a group of instructions, We could choose the one that is going into the integer pipeline as the marker for this group, and tag all of the instructions in the group as being dependent on it. At the completion end of the pipelines, all instructions must resolve their dependences before committing their results, and so every instruction in the group must wait for the integer operation to finish (this assumes the integer pipe is the longest, or that dependences are mutual).

- The added dependence can, however, cause independent operations to stall earlier in the pipeline. Thus, another approach is to tag reservation stations with a retirement order field. The retirement order is a circular count, and there must be a global register that keeps track of the head of the list. At the retirement end of the pipeline, any set of instructions that have a sequential retirement order fields may be retired in a single cycle. The network to test the sequentiality of instructions that are scattered across reservation stations for multiple pipelines can be a bottleneck due to wire and gate delays, so we often see processors that can retire fewer instructions at once than can be issued.

- While it might seem to be a further bottleneck to retire fewer instructions than the issue limit, in practice the impact is minimal. On most cycles, we cannot issue to every pipe anyway. It is even less likely that a group of instructions, that are initially issued out-of-order (but under constraints that are necessary to preserve precise exceptions), are going to complete in a manner such that every pipe is regularly outputting an instruction that meshes perfectly with the output of all of the other pipes so that every one can be retired. In those rare cases, the retirement limit causes the pipes to stall and there is some loss of efficiency. Typically, however, the actual number of instructions that are ready for retirement doesnt approach the hardware limit.

- The extra complexity of reservation stations comes at a price: the 360/91 was far less reliable than the 6600. Even today, it is a serious concern to design a new processor using reservation stations with general out-of-order issue capability, due to the difficulty of validating the design and the extra design cost involved. The designers of the UltraSparc I, for example, found that it would be more economical and nearly as effective to invest in better compiler scheduling than to delay delivery of the processor by trying to support out-of-order issue.

Superscalar-Superpipeline

- Of course we may also combine superscalar operation with superpipelining. The result is potentially the product of the speedup factors.
-
- However, it is even more difficult to interlock between parallel pipes that are divided into many stages. Also, the memory subsystem must be able to sustain a level of instruction throughput corresponding to the total throughput of the multiple pipelines -- stretching the processor/memory performance gap even more. Of course, with so many pipes and so many stages, branch penalties become huge, and branch prediction becomes a serious bottleneck (offsetting this somewhat is the potential for speculative branch execution that arises with a large number of pipes).
-
- But the real problem may be in finding the parallelism required to keep all of the pipes and stages busy between branches. Consider that a machine with 12 pipelines of 20 stages must always have access to a window of 240 instructions that are scheduled so as to avoid all hazards, and that the average of 40 branches that would be present in a block of that size are all correctly predicted sufficiently in advance to avoid stalling in the prefetch unit. This is a tall order, even for highly regular code with a good balance of floating point to integer operations, let alone irregular code that is typically unbalanced in its operation mix.

- As usual, scientific code and image processing with their regular array operations often provide the best performance for a super-super processor. However, a vector unit could more directly address the array processing problems, and what is left to the scalar unit may not benefit nearly so greatly from all of this complexity. Scalar processing can certainly benefit from ILP in many cases, but probably in patterns that differ from vector processing.

- Looking into the Future

- The current focus of the microprocessor community on boosting floating point and image processing performance is neglectful of the basic RISC approach, which takes from Amdahl's law the principle that accelerating the common case is the most important. However, many of the RISC processors are being targeted at high performance computing because they have been unable to penetrate the personal computing market that is so dominated by the Intel CISC architecture. As a result, they have concentrated on the part of the market that Intel has not emphasized (and, not entirely coincidentally, the market that the RISC developers are most familiar with). Of course, Intel is now emphasizing graphics heavily, which boxes the RISC processors further into a corner. Most of them have been retreating into the 64-bit server market, where Intel hopes to crush them with the Itanium.

- In some ways, the current path being taken in the RISC community is to replicate the scalar supercomputers of the 1960's and 1970's in a single chip, with some updating due to technology (remember that a new generation of technological capability begins with reimplementation of the familiar). Those supercomputers have departed the market because of cost/performance that was overcome by advances in first superminis and then microprocessors.

- The shifting of the microprocessor market to commodity economics, however, has compressed the price/performance scale such that it is now even more difficult to introduce custom supercomputers. But beyond that, it is driving the high-performance oriented RISC architectures into the same blind alley as the supercomputers that preceded them. As all of the processors approach the hard speed limits, it becomes more difficult to justify a system based on a processor that costs $1000 more (= $5000 to $10,000 more in system retail cost) in order to obtain a small factor (< 10) boost in performance, especially when it isn't compatible with the majority of available software.

Some viable forward paths are:

- A RISC performance family that ranges from embedded to PC to workstation to supercomputer, with the high volume low-end supporting the upper end. MIPS has developed low-end versions of the 4400 for embedded use, there is a family of PowerPC processors that includes a derivative set for embedded applications.
- An extremely fast RISC scalar integer processor that exceeds the Intel CISC performance by a factor of 10 (a RISC PC engine), with reasonable FP performance and a family of accelerators for FP, vector, graphics, DSP, some of which could be combined on a chip, to address the upper end.

- Intel CISC or clones in a performance family, and/or with accelerators will be the biggest competition in that they will eat away at the high end market while continuing to benefit from the high volume PC performance level. Intel has to be careful not to drive up the cost of its processors in going after the high end, since that will open up an opportunity for the competition to steal their major market by bringing out less expensive clones that accelerate typical PC applications while ignoring the modest market for FP intensive performance. Hence, Intel may have to develop a performance family by design, rather than the current approach of having a defacto family due to overlapping generations -- which seems to be the recent strategy of distinguishing different Pentium models, while keeping them all current.

- It is interesting to note that the Itanium is struggling to stay ahead of the Pentium. The Pentium, meanwhile is driven to outperform the AMD clones. Thus, Intel is seeing internal competition and feeling the effects of the squeeze among its own lines.

Assignment

- Explain Instruction level parallelism.