Chapter 1 : Parallel computing - Wikipedia

Note: Citations are based on reference standards. However, formatting rules can vary widely between applications and fields of interest or study. The specific requirements or preferences of your reviewing publisher, classroom teacher, institution or organization should be applied.

The single-instruction-multiple-data SIMD classification is analogous to doing the same operation repeatedly over a large data set. This is commonly done in signal processing applications. Multiple-instruction-single-data MISD is a rarely used classification. While computer architectures to deal with this were devised such as systolic arrays , few applications that fit this class materialized. Multiple-instruction-multiple-data MIMD programs are by far the most common type of parallel programs. According to David A. Patterson and John L. Hennessy , "Some machines are hybrids of these categories, of course, but this classic model has survived because it is simple, easy to understand, and gives a good first approximation. It is alsoâ€"perhaps because of its understandabilityâ€"the most widely used scheme. Bit-level parallelism From the advent of very-large-scale integration VLSI computer-chip fabrication technology in the s until about , speed-up in computer architecture was driven by doubling computer word size â€"the amount of information the processor can manipulate per cycle. Historically, 4-bit microprocessors were replaced with 8-bit, then bit, then bit microprocessors. This trend generally came to an end with the introduction of bit processors, which has been a standard in general-purpose computing for two decades. Not until the early s, with the advent of x architectures, did bit processors become commonplace. Instruction-level parallelism A canonical processor without pipeline. A canonical five-stage pipelined processor. A computer program is, in essence, a stream of instructions executed by a processor. These processors are known as subscalar processors. These instructions can be re-ordered and combined into groups which are then executed in parallel without changing the result of the program. This is known as instruction-level parallelism. Advances in instruction-level parallelism dominated computer architecture from the mids until the mids. These processors are known as scalar processors. The canonical example of a pipelined processor is a RISC processor, with five stages: The Pentium 4 processor had a stage pipeline. Most modern processors also have multiple execution units. These processors are known as superscalar processors. Instructions can be grouped together only if there is no data dependency between them. Scoreboarding and the Tomasulo algorithm which is similar to scoreboarding but makes use of register renaming are two of the most common techniques for implementing out-of-order execution and instruction-level parallelism. Task parallelism Task parallelisms is the characteristic of a parallel program that "entirely different calculations can be performed on either the same or different sets of data". Task parallelism involves the decomposition of a task into sub-tasks and then allocating each sub-task to a processor for execution. The processors would then execute these sub-tasks concurrently and often cooperatively. Task parallelism does not usually scale with the size of a problem. Distributed shared memory and memory virtualization combine the two approaches, where the processing element has its own local memory and access to the memory on non-local processors. Accesses to local memory are typically faster than accesses to non-local memory. A logical view of a non-uniform memory access NUMA architecture. Computer architectures in which each element of main memory can be accessed with equal latency and bandwidth are known as uniform memory access UMA systems. Typically, that can be achieved only by a shared memory system, in which the memory is not physically distributed. A system that does not have this property is known as a non-uniform memory access NUMA architecture. Distributed memory systems have non-uniform memory access. Computer systems make use of caches â€"small and fast memories located close to the processor which store temporary copies of memory values nearby in both the physical and logical sense. Parallel computer systems have difficulties with caches that may store the same value in more than one location, with the possibility of incorrect program execution. These computers require a cache coherency system, which keeps track of cached values and strategically purges them, thus ensuring correct program execution. Bus snooping is one of the most common methods for keeping track of which values are being accessed and thus should be purged. Designing large, high-performance cache coherence systems is a very difficult problem in computer architecture. As a result, shared memory computer architectures do not scale as well as distributed memory systems do. Parallel computers based on interconnected networks need to have some kind of routing to enable the passing of messages between nodes that are not directly connected. The medium used for communication between the processors is likely to be hierarchical in large multiprocessor machines. Classes of parallel computers[edit] Parallel computers can be roughly classified according to the level at which the hardware supports parallelism. This classification is broadly analogous to the distance between basic computing nodes. These are not mutually exclusive; for example, clusters of symmetric multiprocessors are relatively common. Multi-core processor A multi-core processor is a processor that includes multiple processing units called "cores" on the same chip. This processor differs from a superscalar processor, which includes multiple execution units and can issue multiple instructions per clock cycle from one instruction stream thread ; in contrast, a multi-core processor can issue multiple instructions per clock cycle from multiple instruction streams. Each core in a multi-core processor can potentially be superscalar as wellâ€”that is, on every clock cycle, each core can issue multiple instructions from one thread. A processor capable of concurrent multithreading includes multiple execution units in the same processing unitâ€”that is it has a superscalar architectureâ€”and can issue multiple instructions per clock cycle from multiple threads. Temporal multithreading on the other hand includes a single execution unit in the same processing unit and can issue one instruction at a time from multiple threads. Symmetric multiprocessing A symmetric multiprocessor SMP is a computer system with multiple identical processors that share memory and connect via a bus. Distributed computing A distributed computer also known as a distributed memory multiprocessor is a distributed memory computer system in which the processing elements are connected by a network. Distributed computers are highly scalable. The terms " concurrent computing ", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them.

Chapter 2 : Parallel Computer Architecture Models

Story time just got better with Prime Book Box, a subscription that delivers hand-picked children's books every 1, 2, or 3 months â€” at 40% off List Price.

However, they are beatable. Some common advice for those who have trouble finishing the LR section on time is to just skip parallel reasoning questions. They are typically among the more time consuming questions, so skipping them frees up time to get easier points elsewhere. You can always come back to them at the end if you have extra time do not, however, forget to mark a guess answer on your first pass. Because they are planning to skip them, some prep students choose to just stop practicing PR questions entirely. Parallel reasoning questions are terrific practice for honing your understanding of arguments. Even if you are using the skip and come back strategy, you should attempt to master these questions. Always do them before you look at the answers to a section. Here, we help you figure out how to crush them. You may find that you get your skill level to the point where you feel comfortable attacking them right away on your practice tests. While that often can eliminate two or three answer choices, we want you to be able to solve the damn problem! To do that, it helps to have a firm foundation in conditional reasoning. If you are shaky, check out our FREE conditional reasoning lesson. Rather, you are just trying to find another argument that contains the same structure as the stimulus. Going Block By Block: Each block is one piece of the argument. A pretty typical argument on the LSAT will contain a couple premises and a conclusion. Now these blocks come in different shapes. What you need to do is find an answer that has exactly the same blocks as the stimulus. That means that both arguments have the same number of blocks, and they match in size and shape. Now, forget the content. The stimulus can be about lions and the correct answer can be about aliens. The same blocks that you had in the stimulus can be all jumbled up in the correct answer. The pattern of reasoning in the argument is most similar to that in which of the following arguments? What is the first one? If Tom Brady is healthy, It would be highly unlikely the Colts will win their match-up with the Patriots. Now, that is all we need to know about this block. When we look at the answer choices we are going to be searching for this same kind of block. Now for Block 2. We need to find something just like this in the answer choice. A correct one and one that look close but is not right. If the soccer tournament was not rigged, the team that won it would have been highly unlikely to win the whole thing. Thus, since this team was highly unlikely to win it, the tournament was probably rigged. So far, this question is on the right track. Now, I look at the second sentence. On to answer Choice B. Same conditional statement with unlikely language. Fox did cover the event. That is just like the second block, when the Colts did win. The unlikely did in fact occur. Note the the block is flipped around, but it still makes the same block. You could even chop block 2 up, and still have a valid correct answer. Fox covered a recent event. It might help for you to think of block 2 as actually two separate blocks. The point is merely that these pieces can move all over the place. Comparing Individual Premises And Conclusion Note that in our example the Kaplan and Princeton Review method of comparing conclusions would have actually worked. In our example it was pretty easy. A big element of our Block 2 conclusion, something unlikely actually happening, was just not there. Think about our example correct answer. What if it said: Easier parallel reasoning questions can be done in your head a lot of the time, but you should not hesitate to rely on diagramming, especially for questions that use conditional reasoning. Practice often and you will develop a feel for when you can do it in your head and when it might help to do some diagramming. On the above question example, which is a little harder than the average PR question, I would have definitely written some stuff down. I would probably have written something like: Diagramming is always a little idiosyncratic. The important thing is that you understand the logical relationships expressed by the diagrams. Getting Better Now, you should be ready to approach these problems. Even if you skip them while taking timed section, you should always do the PR questions afterward. You should not worry too much about whether they take a long time to solve. Even some very high scoring test takers skip these problems and only hit them at end. Taking the pressure off and learning how to do them right will naturally build speed. If you find you are doing them in about 2 minutes, which about typical for a PR question, you might be able to start trying them on the first approach on your practice tests rather than skipping them. This gives you plenty of real LSAT LR questions of each type to practice on, along with explanations that apply the correct techniques to the problems. It tests all the various logical reasoning skills that you learn in the LR bible. Good luck and let us know in the comments if you are having trouble with any specific parallel reasoning question. This lesson is excerpted from our Mastermind Study Group. Access us through the private forum or in the live office hours.

Chapter 3 : CiteSeerX â€" Citation Query Reasoning about parallel architectures

Reasoning About Parallel Architectures Epub Book Reasoning About Parallel Architectures Full Online Buy Reasoning About Parallel Architectures On.

Hill , " A memory model for a shared memory, multiprocessor commonly and often implicitly assumed by programmers is that of sequential consistency. This model guarantees that all memory accesses will appear to execute atomically and in program order. An alternative model, weak ordering, offers greater perfor An alternative model, weak ordering, offers greater performance potential. Weak ordering was first defined by Dubois, Scheurich and Briggs in terms of a set of rules for hardware that have to be made visible to software. The central hypothesis of this work is that programmers prefer to reason about sequentially consistent memory, rather than having to think about weaker memory, or even write buffers. Following this hypothesis, we re-define weak ordering as a contract between software and hardware. By this contract, software agrees to some formally specified constraints, and hardware agrees to appear sequentially consistent to at least the software that obeys those constraints. We illustrate the power of the new definition with a set of software constraints that forbid data races and an imple-mentation for cache-coherent systems chat is not allowed by the old definition. In highly-pipelined machines, instructions and data are prefetched and buffered in both the processor and the cache. This is done to reduce the average memory access la-tency and to take advantage of memory interleaving. Lock-up free caches are designed to avoid processor blocking on a cache miss. Write buffers are often included in a pipelined machine to avoid processor waiting on writes. In a shared memory multiprocessor, there are more advantages in buffering memory requests, since each memory access has to traverse the memory- processor interconnection and has to compete with memory requests issued by different processors. Buffering, however, can cause logical problems in multipro-cessors. These problems are aggravated if each processor has a private memory in which shared writable data may be present, such as in a cache-based system or in a system with a distributed global memory. In this paper, we analyze the benefits and problems associated with the buffering of memory requests in shared memory multiprocessors. We show that the logical problem of buffering is directly related to the problem of synchronization. A simple model is presented to evaluate the performance improvement result-ing from buffering. Show Context Citation Context In this case, the cachesscould be connected by a point-to-point interconnection suchsas a ring or a mesh. A software distributed shared memory DSM system allows shared memory parallel programs to execute on networks of workstations. This thesis presents a new class of protocols that has lower communication requirements than previous DSM protocols, and can consequently achieve higher performance. The lazy release consistent protocols achieve this reduction in communication by piggybacking consistency information on top of existing synchronization transfers. Some of the protocols also improve performance by speculatively moving data. We evaluate the impact of these features by comparing the performance of a software DSM using lazy protocols with that of a DSM using previous eager protocols. As part of this comparison, we show that the cost of executing the slightly more complex code of the lazy protocols is far less important than the Supporting SC in the presence of non-atomic memory transactions is even more difficult. If the relative orderings of sub-operations of competing memory operations do not agree, then SC is violated. This problem is especially severe in systems The memory consistency model for a shared-memory multiprocessor specifies the behavior of memory with respect to read and write operations from multiple processors. As such, the memory model influences many aspects of system design, including the design of programming languages, compilers, and the u As such, the memory model influences many aspects of system design, including the design of programming languages, compilers, and the underlying hardware. Relaxed models that impose fewer memory ordering constraints offer the potential for higher performance by allowing hardware and software to overlap and reorder memory operations. However, fewer ordering guarantees can compromise programmability and portability. Many of the previously proposed models either fail to provide reasonable programming semantics or are biased toward programming ease at the cost of sacrificing performance. Furthermore, the lack of consensus on an acceptable model hinders software portability across different systems. This dissertation focuses on providing a balanced solution that directly addresses the trade-off between programming ease and performance. To address programmability, we propose an alternative method for specifying memory behavior that presents a higher level abstraction to the programmer. Adve , " The memory consistency model or memory model of a shared-memory multiprocessor system influences both the performance and the programmability of the system. The simplest and most intuitive model for programmers, sequential consistency, restricts the use of many performance-enhancing optimizations The simplest and most intuitive model for programmers, sequential consistency, restricts the use of many performance-enhancing optimizations exploited by uniprocessors. For higher performance, several alternative models have been proposed. However, many of these are hardware-centric in nature and difficult to program. Further, the multitude of many seemingly unrelated memory models inhibits portability. We use a 3P criteria of programmability, portability, and performance to assess memory models, and find current models lacking in one or more of these criteria. This thesis establishes a unifying framework for reasoning about memory models that leads to models that adequately satisfy the 3P criteria. The first contribution of this thesis is a programmer-centric methodology, called sequential consistency normal form SCNF , for specifying memory models. This methodology is based on the observation that performance enhancing optimizations can be allowed without violating sequential consistency if the system is given some information about the program. An SCNF model is a contract between the system and the programmer, where the system guarantees both high performance and sequential consistency only if the programmer provides certain information about the program. Myreen, Jade Alglave " Multiprocessors are now dominant, but real multiprocessors do not provide the sequentially consistent memory that is as-sumed by most work on semantics and verification. Instead, they have subtle relaxed or weak memory models, usually described only in ambiguous prose, leading to widespread confus Instead, they have subtle relaxed or weak memory models, usually described only in ambiguous prose, leading to widespread confusion. We develop a rigorous and accurate semantics for x86 multiprocessor programs, from instruction decoding to re-laxed memory model, mechanised in HOL. We test the se-mantics against actual processors and the vendor litmus-test examples, and give an equivalent abstract-machine charac-terisation of our axiomatic memory model. For programs that are in some precise sense data-race free, we prove in HOL that their behaviour is sequentially consistent. We also contrast the x86 model with some aspects of Power and ARM behaviour. This provides a solid intuition for low-level programming, and a sound foundation for future work on verification, static analysis, and compilation of low-level concurrent code. Related Work Reasonably precise definitions of relaxed memory models were first studied in the Computer Architecture community, e. We refer the reader to the surveys by Adve and Gharachorloo [8], Luchango [26], and Higham, Kawash, and Verwaal [25] for an overview; the latter Testing shared memories by Phillip B. Sequential consistency is the most widely used correctness condition for multiprocessor memory systems. This paper studies the problem of testing shared-memory multiprocessors to determine if they are indeed providing a sequentially consistent memory. It presents the first formal study of It presents the first formal study of this problem, which has applications to testing new memory system designs and realizations, providing run-time fault tolerance, and detecting bugs in parallel programs. A series of results are presented for testing an execution of a shared memory under various scenarios, comparing sequential consistency with linearizability, another well-known correctness condition. Linearizability imposes additional restrictions on the shared memory, beyond that of sequential consistency; these restrictions are shown to be useful in testing such memories. Modern shared-memory multiprocessors use complex memory system implementations that include a variety of non-trivial and interacting optimizations. In particular; large-scale Distributed Shared Memo In particular; large-scale Distributed Shared Memory DSM systems usually rely on a directory cache-coherence protocol to provide the illusion of a sequentially consistent shared address space. Verifying that such a distributed protocol satisfies sequential consistency is a dificult task. In this papes we examine a new reasoning technique that is precise and we find intuitive. Such total orderings can be used to verify that the requirements of a particular memory consistency model have been satisjed. We apply Lamport clocks to prove that a non-trivial directory protocol implements sequential consistency. We have conducted extensive experiments on several generations of processors: Based on these, on published details of the microarchitectures, and on discussions with IBM staff, we give an abstract-machine semantics that abstracts from most of the implementation detail but explains the behaviour of a range of subtle examples. Our semantics is explained in prose but defined in rigorous machine-processed mathematics; we also confirm that it captures the observable processor behaviour, or the architectural intent, for our examples with an executable checker. While not officially sanctioned by the vendor, we believe that this model gives a reasonable basis for reasoning about current POWER multiprocessors. Our work should bring new clarity to concurrent systems programming for these architectures, and is a necessary precondition for any analysis or verification. We focus on models for the major current processor families that do not have sequentially consistent behaviour: It also introduces PSO Verification Techniques for Cache Coherence Protocols. Because cache protocols are essentially composed of component processes such as memory and cache cont Specifically, we focus on verifying cache protocols where the behavior of an individual protocol component C is modeled as a finite state machine [FSM. Inputs to these machines are processor-generated events and messages for maintaining data consistency. In general, the protocol models are abstracted representations. They are often kept simple to make the complexity of verification manageable, while preserving properties of interest. It is clear that the quality of a ve

Chapter 4 : LSAT Parallel Reasoning Questions: Your Logical Reasoning Nemesis

Seungjoon Park, David L. Dill, An executable specification, analyzer and verifier for RMO (relaxed memory order), Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures, p, June , , Santa Barbara, California, United States.

All the processors have equal access time to all the memory words. Each processor may have a private cache memory. Same rule is followed for peripheral devices. When all the processors have equal access to all the peripheral devices, the system is called a symmetric multiprocessor. When only one or a few processors can access the peripheral devices, the system is called an asymmetric multiprocessor. Here, the shared memory is physically distributed among all the processors, called local memories. The collection of all local memories forms a global address space which can be accessed by all the processors. Here, all the distributed main memories are converted to cache memories. In this case, all local memories are private and are accessible only to the local processors. Multivector and SIMD Computers In this section, we will discuss supercomputers and parallel processors for vector processing and data parallelism. Vector Supercomputers In a vector computer, a vector processor is attached to the scalar processor as an optional feature. The host computer first loads program and data to the main memory. Then the scalar control unit decodes all the instructions. If the decoded instructions are scalar operations or program operations, the scalar processor executes those operations using scalar functional pipelines. On the other hand, if the decoded instructions are vector operations then the instructions will be sent to vector control unit. All the processors are connected by an interconnection network. PRAM and VLSI Models The ideal model gives a suitable framework for developing parallel algorithms without considering the physical constraints or implementation details. The models can be enforced to obtain theoretical performance bounds on parallel computers or to evaluate VLSI complexity on chip area and operational time before the chip is fabricated. Fortune and Wyllie developed a parallel random-access-machine PRAM model for modeling an idealized parallel computer with zero memory access overhead and synchronization. This shared memory can be centralized or distributed among the processors. These processors operate on a synchronized read-memory, write-memory and compute cycle. So, these models specify how concurrent read and write operations are handled. To avoid write conflict some policies are set up. Nowadays, VLSI technologies are 2-dimensional. The size of a VLSI chip is proportional to the amount of storage memory space available in that chip. We can calculate the space complexity of an algorithm by the chip area A of the VLSI chip implementation of that algorithm. If T is the time latency needed to execute the algorithm, then A. For certain computing, there exists a lower bound, f s , such that A.

Chapter 5 : Architectures for reasoning in parallel - CORE

Find helpful customer reviews and review ratings for Reasoning About Parallel Architectures at www.nxgvision.com Read honest and unbiased product reviews from our users.