Chapter 1 : Multithreaded, Parallel, and Distributed Programming

Foundations of Multithreaded, Parallel, and Distributed Programming covers, and then applies, the core concepts and techniques needed for an introductory course in this subject. Its emphasis is on the practice and application of parallel systems, using real-world examples throughout.

Clustalw analysis using distributed and parallel computing by Kuo-bin Li - Bioinformatics , " ClustalW is a tool for aligning multiple protein or nucleotide sequences. The alignment is achieved via three steps: All three steps have been parallel All three steps have been parallelized to reduce the execution time. The software uses a message-passing library called MPI Message Passing Interface and runs on distributed workstation clusters as well as on traditional parallel computers. Finally, the calculations of the forward and backward passes of the dynamic programming are also parallelized. Software Architecture for Computer Vision: Beyond Pipes and Filters by Alexandre R. This document highlights and addresses architecture level software development issues facing researchers and practitioners in the field of Computer Vision. A new framework, or architectural style, called SAI, is introduced. It provides a formalism for the design, implementation and analysis of sof It provides a formalism for the design, implementation and analysis of software systems that perform distributed parallel processing of generic data streams. Architectural patterns are illustrated with a number of demonstration projects ranging from single stream automatic real-time video processing to fully integrated distributed interactive systems mixing live video, graphics and sound. Show Context Citation Context This section first offers an overview of the classic Pipes and Filters architectural style, emphasizing its desirable properties, and highlighting its main limitations. PDES is a fine-grained parallel application whose performance and scalability are limited by communication latencies. Traditionally, PDES simulation kernels use pro Traditionally, PDES simulation kernels use processes that communicate using message passing; shared memory is used to optimize message passing for processes running on the same machine. We report on our experiences in implementing a thread-based version of the ROSS simulator. The multithreaded implementation eliminates multiple message copying and significantly minimizes synchronization delays. We study the performance of the simulator on two hardware platforms: We identify performance bottlenecks and propose and evaluate mechanisms to overcome them. Results show that multithreaded implementation improves performance over the MPI version by up to a factor of 3 for the Core i7 machine and 1. Efficient partitioning is necessary to reduce remote communication [30], [18]. However, most efforts in implementing parallel applications discover that application insights and awareness of the arc Due to the interplay between increasing chip capacity and complex applications, System-on-Chip SoC development is confronted by severe challenges, such as managing deep submicron effects, scaling communication architectures and bridging the productivity gap. Network-on-Chip NoC has been a rapidl Network-on-Chip NoC has been a rapidly developed concept in recent years to tackle the crisis with focus on network-based communication. NoC problems spread in the whole SoC spectrum ranging from specification, design, implementation to validation, from design methodology to tool support. In the thesis, we formulate and address problems in three key NoC areas, namely, on-chip network architectures, NoC network performance analysis, and NoC communication refinement. Quality and cost are major constraints for micro-electronic products, particularly, in high-volume application domains. We have developed a number of techniques to facilitate the design of systems with low area, high and predictable performance. When humans interact with computer systems, they expect the experience to meet human standards of reactiveness, robustness and, if possible, non-intrusiveness. In order for computer vision techniques to have a significant impact in human-computer interaction, the development of e cient and robust In order for computer vision techniques to have a significant impact in human-computer interaction, the development of e cient and robust algorithms, as well as their integration and operation as part of complex including multi-modal systems, must be specifically addressed. Results show that the SAI formalism is an enabling tool for designing, describing and implementing robust systems of e cient algorithms.

In the Pipes and Filters style, the components, called filters have a set of in Centralized matchmaking for minimal agents by K. In this paper we propose a simple though efficient mechanism to enable distributed processing among nodes to communicate with each other and to dynamically rebalance the workload among them. We discuss a centralized matching mechanism and investigate different matching functions, varying in the amou We discuss a centralized matching mechanism and investigate different matching functions, varying in the amount of information that is taken into account. We also show that this time efficiency does not have a cost in terms of efficient task allocation and resource utilization. They are all different instances of the same problem where competing processes are looking for resources. Simultaneous reachability analysis SRA is a recently proposed approach to alleviating the state space explosion problem in reachability analysis of concurrent systems. The concept of SRA is to allow a global transition in a reachability graph to contain a set of transitions of different proces The concept of SRA is to allow a global transition in a reachability graph to contain a set of transitions of different processes such that the state reached by the global transition is independent of the execution order of the associated process transitions. Our multi-port EFSM model assumes asynchronous message passing, which involves nonblocking send and blocking receive. We present the main features of a programming language designed to be applicable for parallel computing on a wide range of platforms, including a range of programmable hardware. Our language supports an object oriented, yet statically instantiated style of programming; parallel processing with commu Our language supports an object oriented, yet statically instantiated style of programming; parallel processing with communication and synchronization by rendezvous; and reusability via generic classes, interfaces, and connections between objects. Interobject communication is by shared memory or by rendezvous. From the point of view of a client object, rendezvous looks much like a method call in an langua Theory and practice of rapid elasticity in cloud applications

Chapter 2 : Foundations Of Multithreaded Parallel And Distributed Programming - PDF Free Download

Foundations of Multithreaded, Parallel, and Distributed Programming Cover photo taken near Flora, Norway by Philippe Colombi. Click image for a larger version.

Hence, the processors cornrnunicale with each other by sending and receiving messages. Each processor has its own cache, but because memory is not shared there are no cache or memory consistency problems. For this reason, a multicomputer is often called a fightill coupbed mnchirze. A multicomputer is used by one or at most a few applications at a time; each application uses a dedicated set of processors. The interconnection network provides a highspeed and high-bandwidth communication path between the processors. It is typically organized as a mesh or hypercube. Hypercube machines were among the earliest examples of multicomputers. Consequently, network systems are called loosely cotiplccl multiprocessors. Again processors communicate by sending messages, but these take longer to deliver tkan in a rnulticornputer and there is Inore network contention. These workstarions execute a single application, separate applications, or a mixture of the two. A currently popular way to build an inexpensive distributedmemory multiprocessor is to construct whac is called a Beovvulf rnnchine. A Beowulf machine is built out of basic hardware and free software. The name Beowulf refers to an Old Engljsh epic poem, the first masterpiece of English literature. Hybrid combinations of distributed- and shared-memory rnultjprocessors also exist. The nodes of the distributed-memory machine might be sbaredmemory multiprocessors rather than single processors. This makes the machines easier to program for many applications, but again raises the issues of cache and memory consisrency. It also provides a way to make use of lnultiple processors. There are three broad, overlapping classes of applications-multithreaded systems, distributed systems, and parallel computations-and three corresponding kinds of concursent programs. Recall that a process is a sequential program that, when executed, has its own thread of control. Every concurrent program contains multiple processes, so every concul-rent program has multiple theads. However, the term multithreaded usually means that a program contains more processes threads than there are processors to execute the threads. Consequently, the processes take turns executing on the processors. These systems axe written as rnultithreaded programs because it is much easier to organize the code and data structures as a collection of processes than as a single, huge sequential program. In addition, each process can be scheduled and executed independently. For example, when the user clicks the mouse on a persona! That process thead can now execute and respond to rhe mouse click. In addition, applications in other windows can continue to execute in the background. The second broad class of applications is distributed computing, in which components execute on machines connected by a local or global communicaeion network. Hence the processes communicate by exchanging messages. Examples are as follows: For example, a file server provides data files for processes executing on client machines. The components i n a distributed system are often themselves multitheaded programs. Parallel cornpuling is the third broad class of appljcations. The goal is to solve a given problem faster or equivalently to solve a larger problem in the same amount of time. Examples of parallel computations are as follows: They are executed on parallel processors to achieve high performance, and there usually are as many processes threads as processors. We examine these as well as other applications in this book. Most importantly, we show how to program them. The processes i n a distributed system cornmunicate by exchanging messages or by invoking remote operations. The processes in a parallet computation interact using either shared variables or message passing, depending on the hardware on which the program will execute. Part 2 covers message passing and remote operations. Although there are Inany concurrent programming applications, they employ only a small number of solution patterns, or paradigms. In particular, there are five basic paradigms: Applications are programmed using one or more of these. Iterarive parallelism is used when a program has several, often identical processes, each of which contains one or more loops. Hence, each process is an iterative program. The processes in the program work together to solve a single problem; they communicate and synchronize using shared variables or message 12

Chapter 1 The Concurrent Computing Landscape passing. Recursion is often used in imperative languages, especially to implement divide-and-conquer or backtracking algorithms. Recursion is also the fundamental programming paradigm in symbolic, logic, and functional languages. They are often organized inro a pipeline through which inforination flows. Each process in a pipeline is a. Filters occur at the appljcation shell level in operating systems such as Unjx, within operating systelns themselves, and within application programs whenever one process produces output that is consumed read by another. Clierzls and senjers are the dominant interaction pattern in distributed systems, from local networks to the World Wide Web. A client process requests a service and waits to receive a reply. A server waits for requests from clients then acts upon them. A server can be implemented by a single process that handles one client request at a time, or it can be multithreaded to service client requests concurrently. Clients and servers are the concurrent programming generalization of procedures and procedure calls: However, when the client code and server code reside on different maclnines, a conventional procedure call-jump to subroutine and later return from subroutine-cannot be used. Instead one has to use a remote procedure call or a rerzdezvous, as described in Chapter 8. Interacting peers is the final interaction paradigm. It occurs in distributed programs when there are several processes that execute basically the same code and that exchange messages to accomplish a task. Interacting peers are used to implement distributed parallel programs, especially those with iterative parallelism. They are also used to implement decentralized decision making in distributed systems. We describe several applications and communication patterns in Chapter 9. The next five sections give examples that illustrate the use of each of these patterns. The examples also introduce the programming notation that we will be using throughout the text. Many more examples are described in later chapters-either in the text or in the exercises. Matrix Multiplication 13 1. An iterative parallel program contains two or more iterative processes. Each process computes results for a subset of tbe data, then the results are combined. As a simple example, consider rhe following problem from scientific cowputing. Given matrices a and b, assume that each matrix has n rows and columns, and that each has been initialized. The goal is to compute the matrix product of a and b, storing the result in the n x n matrix c. This requires computing n 2 jnner products, one for each pair of rows and columns. The rnatl-ices are shared variables, which we will declare as fallows: By default, the indices for the rows and columns range from o to n- 1. After initializing a and b, we can compute the matrix product by the following sequential program: The inner loop with index k computes the inner product of row i of matrix a and column j of matrix b, and then stores the resu1. The line that begins with a sharp character is a comment. Matrix multiplication is an example of what is called an embnrrnssi Two operations can be executed in parallel if they are independent. Assume that the read set of an operation contains the variables it reads but does not alter and that the write set of an operation contains the variables that it alters and possibly also reads. Two operations are independent if the write set of each is disjoint from both the read and write sets of the other. Informally, it is always safe for two processes to read variables that do not change. However, it is generally not safe for two processes to write into the same variable, or for one 14 Chapter 1 The Concurrent Computing Landscape process to read a variable that the other writes. We will examine this topic in detail in Chapter 2. For matrix multiplication, the computations of inner products are independent operations. The innermost loop in the program reads a row of a and a column of b, and reads and writes one element of c. Hence, the read set for an inner product is a row of a and a column of b, and U1e write set is an element of c. Since the write sets for pairs of inner products are disjoint, we could compute all of then1 in parallel. Alternatively, we could con-lpute rows of results i n pacallel, columns of results in parallel, or blocks of rows or colurnns i n parallel. Below we show how to program these parallel computations. First consider cornpudng rows of c in parallel. This can be program. However, there is an important semantic difference: A different way to parallelize matrix lnultiplication is to compute the columns of c in parallel. It is safe- to interchange two loops as long as the bodies are independent and hence compute the same results, as they do here. Matrix Multiplication 15 We can also compute all inner products in parallel. This can be programmed in several ways.

DOWNLOAD PDF FOUNDATIONS OF MULTITHREADED, PARALLEL, AND DISTRIBUTED

Chapter 3 : CiteSeerX â€” Citation Query Foundations of Multithreaded, Parallel and Distributed Programm

Foundations of Multithreaded, Parallel, and Distributed Programming and a great selection of similar Used, New and Collectible Books available now at racedaydvl.com - Foundations of Multithreaded, Parallel, and Distributed Programming by Gregory R Andrews - AbeBooks.

Chapter 4 : Download [PDF] Foundations Of Multithreaded Parallel And Distributed Programming â€" Fodr

It also provides a way to make use of lnultiple processors. There are three broad, overlapping classes of applications-multithreaded systems, distributed systems, and parallel computations-and three corresponding kinds of concursent programs. Recall that a process is a sequential program that, when executed, has its own thread of control.

Chapter 5 : Pearson - Foundations of Multithreaded, Parallel, and Distributed Programming - Gregory R. A

DOWNLOAD NOWÂ» Foundations of Multithreaded, Parallel, and Distributed Programming covers, and then applies, the core concepts and techniques needed for an introductory course in this subject.

Chapter 6 : [PDF] Download Foundations Of Multithreaded Parallel And Distributed Programming â€" Free

Add tags for "Foundations of multithreaded, parallel, and distributed programming". Be the first.