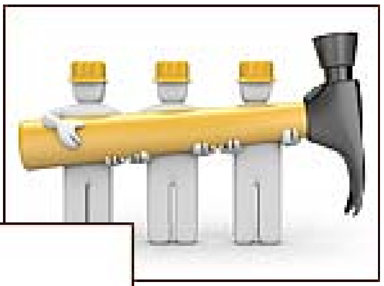
Parallel Programming



Programming languages, tools, and environments for parallel computers are necessary instruments for the design and implementation of high performance applications. Indeed, parallel programming emerged during the late 1960s and 1970s, attempting to express parallel processes in programming terms, and to improve performance of computer systems.

During recent decades parallel computers ranging from tens to thousands of processing elements have become commercially available. However, these systems require concurrent software and applications and thus a challenge arises: how can shift software design from its historic, inherently-serial approach, to one which incorporates concurrency?

Ideas like: "If one is good, then 10 or 1,000 should be better" originated parallel programming. Even though such a statement is not necessarily true, it proposes a primary attraction towards parallelism. Several competitive methods were proposed for organizing parallel programming, but there was neither enough evidence as to which design was superior, nor sufficient knowledge on which to make a careful evaluation. Therefore, presently, the high-performance programming community is, in general, not working with a rigorous discipline of programming.

The combined work of Dijkstra, Tony Hoare, and Brinch Hansen (they can be considered as pioneers in the development of parallel programming languages) on programming notation for operating system concepts led to the initial development of parallel programming languages. Hoare’s work in 1978 has been fundamental for the development of parallel programming during the last years. His main contribution has been the definition of a language for the formal specification of parallel algorithms, known as CSP (Communicating Sequential Processes). Also in 1978, and with the development of network communication among computers, Brinch-Hansen wrote a paper in which he proposed another kind of parallelism. His work has originated what is known as remote procedure call (RPC), which is the base of distributed programming today; for instance, it forms the basis of distributed communication in CORBA.

Based on the concepts, properties and characteristics proposed by Dijkstra, Hoare, and Brinch-Hansen, other authors have taken the task to develop further such ideas in formal terms and different languages for concurrent, parallel and distributed programming (Hoare, 1985; Andrews, 1991; Lynch, 1996; Hartley, 1998; Andrews, 2000). In the area of Software Engineering, some authors have developed different methods for parallel programming (Foster, 1994; Culler, 1997).

Parallel programming is complex activity, aiming for developing specifications of parallel processes which execute simultaneously and non-deterministically. Commonly, parallel programming is developed in order to obtain performance gains about execution time and increase the size of computational problems that can be solved. Nevertheless, one of the major factors that affect the performance obtained when applying parallel programming is the hardware platform. In fact, based on various parallel machine architectures models we have different parallel programming patterns. As an instance, programming a parallel computer depends on the way in which the memory of the hardware platform is “organized” or “divided” among the processors.

The “parallel programming problem” has been addressed, in high performance computing, for at least 25 years, with the result that – still - only a small number of specialized developers write parallel code. Worse yet, Programmers still find it difficult to reason about multiple simultaneous tasks in the parallel model, which is much harder for the human brain to grasp than writing “plain old” algorithms. In other words, Parallel computer programs are more difficult to write than sequential ones, because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different subtasks are typically one of the greatest obstacles to getting good parallel program performance.

From the birth point of parallel programming until now, many aspects of this technology have been discussed in researchers work. some issues in the area of parallel computing can be listed as follows: finding concurrent tasks in a program, scheduling tasks onto the processors of a parallel machine, supporting scalability, tools, API's and methodologies to support the debugging process.

Unfortunately, the large majority of today's software is written for a single processor, and there is no technique known to "auto-parallelize" these programs. Due to the difficulties in automatic parallelization today, people have to choose a proper parallel programming model or a form of mixture of them to develop their parallel applications on a particular platform.

Throughout the 1990s, hundreds of parallel programming technologies were created. However, the adoption rate was very low – why? A part of the problem was choice overload: the tendency of a consumer, when presented with too many choices, to walk away without making a choice. Choice overload is real. When we present application programmers a myriad of parallel programming environments, we overwhelm them, and decrease the adoption of the technology.