Current Architectures for Parallel Processing

António Lira Fernandes

MICEI, Universidade do Minho
4710 - 057 Braga, Portugal
alf@nortenet.pt

Abstract: The purpose of this tutorial is to provide the basic concepts related to parallel processing. We will first discuss the fundamentals of parallel machine architectures and parallel programming, including a short view of Flynn's taxonomy with some implementation examples. An overview of some programming techniques supported in parallel environments follows, namely MPI-2 and OpenMP. Finally, a presentation of the top fastest computer systems worldwide based on the TOP500 closes this tutorial.

1 Introduction

The starting point of this tutorial is to understand why parallel computing is needed. The model to describe a problem is often too complex, so we need more and more computational power to find a solution. If we think about weather forecast, communication interceptions, folding of proteins or search for extraterrestrial intelligence, then tens of TFlops are needed to get a closer approach. This requires some sort of concurrency and we ought to think on using several of the existing computing machines in a joined manner…

The challenge in parallel systems is the development of code models to use the capabilities of the available hardware to solve more and larger problems in less time. But parallel programming is not an easy task, since there are large varieties of architectures and programming techniques.

This tutorial is structured into five main sections. We will first discuss the current parallel approaches in terms of parallel environments, with a focus on single computers with tightly-coupled multi-processors and on clusters; then we will revisit the Flynn taxonomy, which helps to understand the several types of architectures. On the following section we will present memory models, getting a closer view of shared memory systems (UMA and NUMA) and distribute memory systems (Clusters). We will present programming models for shared model systems and for distributed memory systems, namely the message-passing and the threads paradigms, on the following section. Finally, we will briefly review the number one parallel processing system on the top 500[1].

[1] TOP500. Top500 supercomputer sites. http://www.top500.org, 2003

2 Current Parallel Approaches

Three basic platforms for parallel environments are currently popular:

- a single computer with multiple processing elements, known as a tightly-coupled system, such as a SMP system or a ccNUMA system; inter-process communication is through concurrent programming with shared memories;
- a cluster of connected computers, where each computer node performs a set of tasks and uses message passing to exchange messages with the other nodes, no matter of the physical interconnection structure; also known as loosely-coupled systems;
- hybrid-coupled systems, consisting on a fusion of the first two models; usually, a cluster of SMP systems.

3 Taxonomy

To classify parallel systems Flynn [1] developed a taxonomy that helps to understand how the instruction stream (I) and the data stream (D) flows in a computer system. There are four different types of system organization:

- **Single Instruction-stream, Single Data-stream (SISD)** - is the standard workstation/PC type. A SI of a single processor is performing a task on a SD.
- **Single Instruction-stream, Multiple Data-streams (SIMD)** - is the array/vector computer type. One processor controls the others processors, which may have a private memory unit. There is only a single instruction stream using multiple data streams.
- **Multiple Instruction-streams, Single Data-stream (MISD)** - There are no computer systems using this scheme, because the data control is too hard to perform.
- **Multiple Instruction-streams, Multiple Data-streams (MIMD)** - is the most common type of a parallel computer. MI performs their task on their individual data stream. Most modern parallel systems belong to the MIMD category.

This taxonomy actually does not differentiate well tightly-coupled systems, which is due to the state-of-the-art in parallel computing in the seventies.

4 Memory Models

To clarify the MIMD models [2] we need to understand the memory architectures of a system. The way we see the memory in a system might be different from the real architecture, it is important for the distribution of a problem in the system.

Two memory models are possible: distributed memory and shared-memory. If in the system each processor has its own private memory then we are talking about a distributed memory system, common in a cluster. If the memory is understood as unique for all system then we are talking about shared memory system. Shared memory systems are further divided into UMA (Uniform Memory Access) and into NUMA (Non-Uniform Memory Access) systems. In the shared memory the system is controlled by an integrated operating system that

provides interaction between processors and interaction at job, task, file and data element levels.

4.1 Shared Memory Systems

The two types of shared memory systems are UMA and NUMA. In an UMA system all main memory is in a block, sharing a bus, that connects all processors, IO devices and the main memory (Fig.1). If all processors are similar then they have proximally the same access time to memory, all processors can perform the same functions, then we are in presence of a Symmetric Multiprocessor system (SMP [3]). SMP system has practical limit to number of processors, because the bus traffic limit is between 16 and 64 processors.

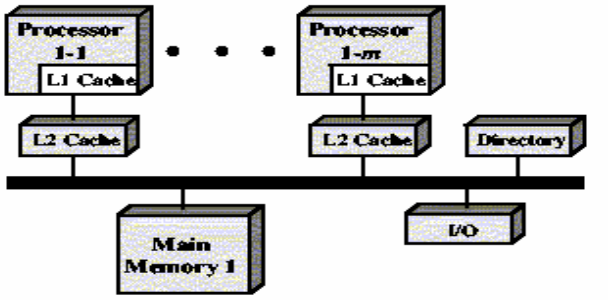

Fig. 1: Uniform Memory Access

An alternative to SMP and clustering is a non-uniform memory access – NUMA - (Fig. 2). A NUMA system combines two or more UMA sub-systems, connected by a fast interconnection, all processors have access to all parts of memory (shared memory) and each UMA sub-system is called a node, and each node has a memory. Access time of processor differs depending on the region of memory, because the main interconnection can be slower than the one on UMA sub-system. If a cache coherent is maintained among the caches of the various processors, usually in hardware, then we call it a ccNUMA system [4].

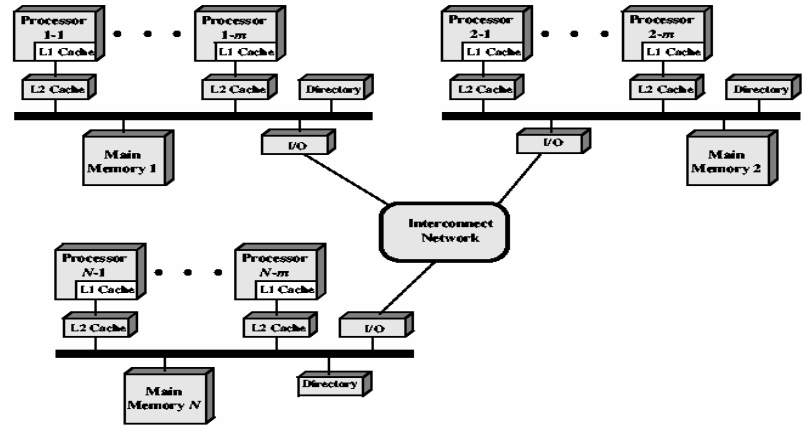

Fig. 2: Non-Uniform Memory Access

An UMA system has better communication but is hardly scalable. At some point the complexity of the interconnection will virtually rise into infinity, or it will not be powerful enough to provide sufficient performance. NUMA retains SMP essence while giving large scale multiprocessing.

The purpose is to maintain transparent system with large memory and permitting multiprocessor nodes, each with individual bus or internal interconnection.

4.2 Distributed Memory Systems

The other memory model is the distributed memory, where the memory is assigned to each individual processor. In this memory category the communication between processors is expensive (longer latency), and should be reduced, so data and tasks should be distributed at the beginning of the code execution.

The processing starts with the distribution of tasks and data through the network to processors. Each processor processes data and communicates with other processor to exchange results and get new tasks. In the end the results are sent to one appropriate receiver.

Clusters fit in this model, where each computer has its own memory, working together as unified resources, giving the idea of being one single machine. Each computer is called a node.

Thomas Sterling and Don Becker, NASA researchers, developed Beowulf[2], the first high-performance open cluster, in 1994.

4.3 Cluster vs. SMP

Both carry on multiprocessor capability for high demand applications. SMP is available commercially for longer, and is easier to manage and control, because it is integrated with a single operating system and it is closer to single processor systems. SMP uses less physical space and lower power consumption

Clustering has superior incremental and absolute scalability with superior availability and redundancy.

4.4 NUMA vs. Cluster Computing

We can see a NUMA like a very tightly couple architecture of a cluster. Using virtual memory paging in a cluster architecture we can implemented a NUMA entirely in software, with high inter-node latency.

Clustering is cheaper and has superior incremental scalability with superior availability and redundancy.

[2] Beowulf.Org The Beowulf Cluster Site , http://www.beowulf.org, 2004

5 Programming Models

In terms of different hardware approaches of parallelization we have seen: loosely-coupled and tightly-coupled processing, and memory access architectures. In this section, we will see two different paradigms for the programming of parallel environments.

To use the available hardware, exploring the parallel processing capability, we need to choose the most efficient software solution. The existing hardware and the type of problem that we face are both important for the paradigm we choose.

5.1 Message-Passing

The message-passing paradigm is a programming solution that connects processor elements, distributing tasks and data structures, by exchanging messages. We need to manually split the problem in small blocks, with tasks and data structures, sending them to each process (concurrent programming). Writing is like sending a message and reading is like receiving. We can even have more processes than CPUs, but normally we try to match these two facts together.

It works on clusters and on parallel computers, the topology is transparent to the user (can be implemented on systems with diverse architectures).

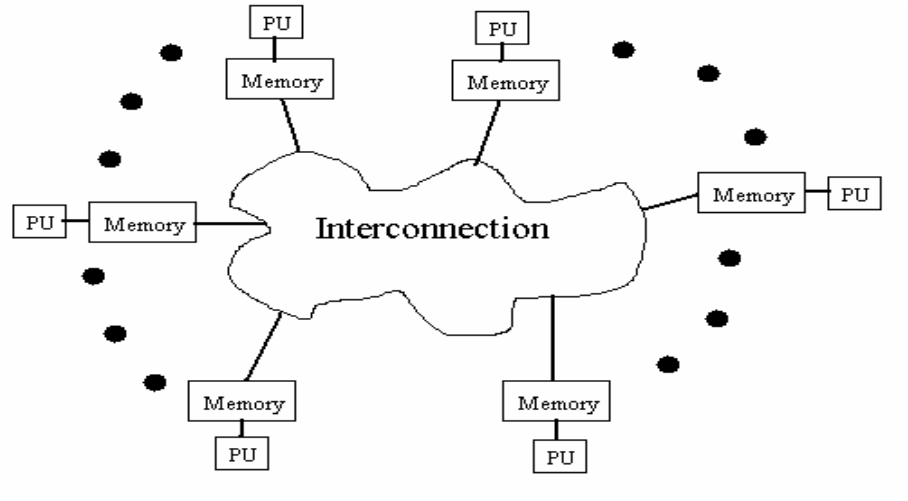

Fig. 3 - The message-passing programming paradigm is suited for distributed memory systems.

There are several message-passing libraries available, being the most popular the PVM library (Parallel Virtual Machine [5]) and the one supplied by the MPI standard (Message Passing Interface [6]; [7]). G. Geist et al.[8] compare PVM with MPI.

5.1.1 Message Passing Interface – MPI

MPI is designed as a communication API for multi-processor computers. The first version (MPI 1) was presented in 1994 and an early version of MPI 2 in 1997.

It is implemented using a communication library specification for message-passing of the vendor of the machine. It adds an abstraction level between the user and the hardware, to guarantee the portability of the program code. It can work on heterogeneous workstation clusters or on massively parallel machines, with high-performance communication on large multi-processors. It has a rich variety of communication mechanisms but it lacks dynamic resource management, which is required in fault tolerant applications.

5.2 Threading

An early programming paradigm is the thread model. It is efficient only on shared memory systems. To get concurrent control flow one process provides the environment for multiple threads, sharing memory (data) with a private stack and programming counters. A common address space is created (context), that needs synchronization mechanisms for communication between threads via shared memory. Usually, we have one thread per processor

To understand how threads work we can do an analogy with a person who has to do two things at a particular point of time. He may divide his time with the two activities to complete them as soon as possible. In a multiprocessing system the two (or more) activities are divided by the operating system into threads running in parallel.
Fundamentally, there are three different kinds of implementations for threads. There is a user thread model (library model), a kernel thread model, and a mixed model (hybrid model).

5.2.1 OpenMP

OpenMP[3] is designed as an API for multithreaded applications, with a set of compiler directives, library routines and environment variables. It enables basic loop-based parallelism for Fortran (77 and up), C, and C++. The MP in OpenMP means Multi Processing and Open means that hardware and software industry, government and academia provide specifications for Multi Processing

The idea is a fork-join model of parallel execution (Fig. 4). OpenMP is usually used to parallelize loops, finding the most time consuming loops, and splitting them up between threads.

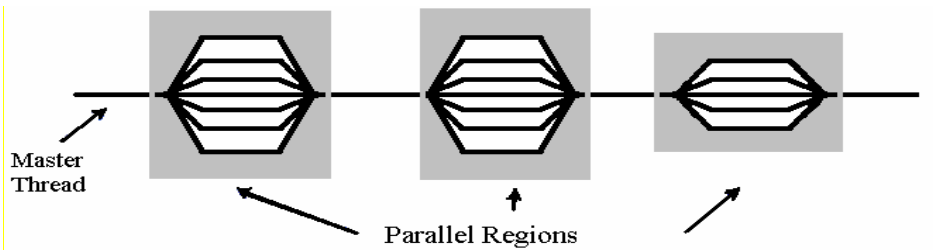

Fig. 4 - The fork-join model

[3] OpenMP Architecture Review Board. OpenMP. www.openmp.org.

A program written with the OpenMP API begins execution as a single thread of execution called the master thread. The master thread runs in serial until the first parallel directive (parallel construct) is found. Subsequently, the master thread spawns a group of threads as needed and parallelism is added incrementally, so the sequential program is transformed into a parallel program. Next, the group of threads join into a master thread and the program continues in a single thread execution until the next parallel construct.

OpenMP is a shared memory model. Threads communicate by sharing variables, leading to race problems. To control race conditions we use synchronization to protect data conflicts. Synchronization is expensive, so we must change the way data is stored to minimize the need for synchronization.

Currently, OpenMP is available for a variety of platforms. Some platforms are supported by vendor-native efforts (i.e., SGI), others by third party products.

Information on how to use OpenMP with C or C++ can be found in the OpenMP Tutorial at SuperComputer 1998 conference [9].

6 Top 500

In 1993 Hans Meuer, Erich Strohmaier, Jack Dongarra and Horst D. Simon started a project - TOP500 - that provided, twice a year, a list of the 500 most powerful computer systems. They use the Linpack benchmark to get the systems performance and order the list. The top system in November, 2003 was a Japanese earth simulator, while the top European system is at the Commissariat a l'Energie Atomique (CEA) – France (placed in 15th).

6.1 The first in Top500 – Earth Simulator (ES)

The system configuration for ES[4] is a distributed memory type with highly parallel vector supercomputer system, implemented with 640 processor nodes (PN) connected by 640x640 single-stage crossbar switches. Each node has a shared memory (16GB), 8 vector-type arithmetic processors (APs), a remote access control unit (RCU), and an I/O processor. Each AP has a peak performance of 8Gflops. The ES has a total of 5120 APs with 10 TB of main memory and the hypothetical performance of 40Tflops.

[4] Earth Simulator Center, http://www.es.jamstec.go.jp/, 2004

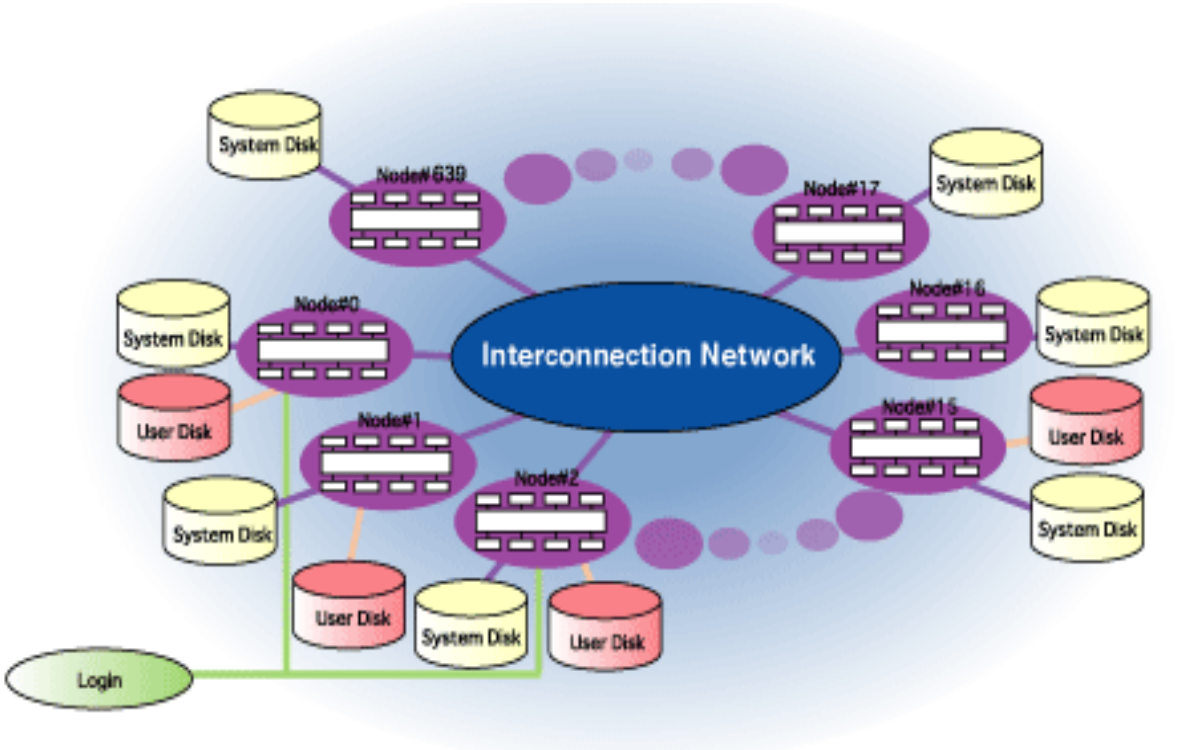

Fig. 5 - System Configuration

The ES hardware has a 3-level hierarchy of parallelism: vector processing in an AP, parallel processing with shared memory in a PN, and parallel processing among PNs via IN. They develop parallel programs that get the best performance for the system. They used two approaches: hybrid and flat parallelization. In the hybrid parallelization, the inter-node parallelism is expressed by HPF or MPI, and the intra-node by microtasking or OpenMP. In the flat parallelization, both the inter-node and the intra-node parallelism use HPF or MPI. The hybrid parallelization is superior to the flat in performance, but it is more complex to program.

7 Conclusions

Two mains factors concurred to improve the importance of paralleling computing. First, the need for a more perfect model of reality – imposing more Flops to reach solutions. Second, the fact that the capability of computing technology has steadily increased for more than four decades, resulting in machines with very large memories, greater computing performance, larger disk caches, and higher-performing network interfaces.

The more common architecture of the systems is a cluster (cluster of SMP systems), with some TFlops. But the global distribution of processing capacity tends to be a fair platform to develop none-critical applications (time and security), with a lot of processing tasks. The distribute programming techniques can give an important help.

In terms of programming languages, the choice between OpenMP and MPI 2 has pros and cons. The choice depending on: type of applications (problems), requirements (labour costs, etc) and availability (software/hardware); so, it is encouraging to adopt hybrid paradigms in your own application. There are more positive experiences of developing hybrid MPI/OpenMP parallel paradigms now.

The parallel processing is also a wide field of research, with a large variety of topics to improve.

References

[1] Michael J. Flynn, Some computer organizations and their effectiveness, IEEE Transactions on Computers, Vol.C-21, No.9, 1972, pp. 948-960.

[2] Johnson, E.E., Completing an MIMD multiprocessor taxonomy, Computer Architecture News 16, 1988, 44-47.

[3] Brooks III, E.D., Warren, K.H., A study of performance on SMP and distributed memory architectures using a shared memory programming model, SC97: High Performance Networking and Computing, San Jose, 1997.

[4] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy. The directory-based cache coherence protocol for the DASH multiprocessor. In Proceedings of the 17th Annual International Symposium on Computer Architecture, pages 148-159, May 1990.

[5] Geist, A. Beguelin, J. Dongarra, W. Jian, R. Machek, and V. Sunderam. PVM: Parallel Virtual Machine. MIT Press, 1994.

[6] MPI Forum. MPI: A message-passing interface standard. International Journal of Supercomputer Applications, 8(3/4):165–416, 1994.

[7] MPI Forum. MPI-2: Extensions to the Message-Passing Interface. Technical Report MPI 7/18/97, Message-Passing Interface Forum, 1997.

[8] G. Geist, J. Kohl, and P. Papdopoulos. PVM and MPI: A Comparison of Features. In Calculateurs Paralleles, Volume 8(2), 1996.

[9] R. Eigenmann, B. Kuhn, T. Mattson, and R. Menon. Introduction to OpenMP. In Supercomputing, 1998.