

Computer Architecture 101

The **cluster computer** is a hot topic today. What it is, and how it evolved is the subject of this paper. One kind, proposed by Kai Li over two decades ago, runs sophisticated software in order to emulate a true multiprocessor, trading execution speed for exorbitant cost to run existing parallel applications on a budget. This is called a distributed shared memory multiprocessor, or DSRM. Another kind forgoes the special software as *middleware*, but expects each application to be parallelized to take full advantage of the several independent nodes in the cluster. This is the simplest multiprocessor; it is what we commonly call a cluster computer.

Uniprocessors:

The simplest computer architecture is called a *uniprocessor* because it contains just one processor, which is also called the central processing unit, or CPU. The processor executes instructions that act upon data, and both the instructions and their data must be in memory in order for them to be executed by the processor.

If the instructions are not already in memory they must be read into it from a disk file, for example. If the data that the instructions act upon are not already in memory, they too must be read into it, from a data file on disk or from some other peripheral device. These are important concepts because getting the instructions and their data into memory in the more advanced computer architectures can be quite complex.

The uniprocessor design is not a new one, dating back to the original concept of a computer proposed by John Von Neumann. The book *The Computer and the Brain*, published by Yale University Press in 1958, is a series of his lecture notes describing his early work in the field.

Multiprocessors:

In contrast, a multiprocessor is a computer architecture that employs more than one CPU. It is often very expensive and it is usually only available commercially from a computer manufacturer.

There is another kind of multiprocessor, however, called a loosely coupled multiprocessor, or multicomputer. It is the **cluster computer** that's become such a hot topic, and it's the one type of sophisticated computer that can be assembled from a few (high-end or salvage) PCs and some networking hardware.

A few PCs and some networking hardware are not quite enough, however. You must configure the operating system files so that each PC is *aware* of the others on the network, and you must also add some custom software that gives the cluster its *personality* – its team focus. A properly tuned multicomputer can outperform a uniprocessor of equivalent power by exploiting any parallelism that may be available in an application.

Where did the cluster architecture come from; how did it evolve?

Alternative Computing Architectures:

To examine the architecture of a computer is to find out “how it is built.” To see how computer architectures can differ, it is necessary to understand the building blocks inside a computer, such as the typical desktop computer, which is an ordinary uniprocessor.

The three functional components of a typical computer are the instruction processor (CPU), physical memory (RAM), and its input-output (I/O Port) devices:

- The processing function includes instruction execution, arithmetic or logic operations, and memory data access. These functions are grouped into the central processing function because the central processing unit, or CPU, usually performs them all.
- The memory function, *remembering* the instruction and data values, is performed by the physical memory and its associated hardware. Memory is where instructions and data may be stored after bringing them in from the outside world through some I/O port. It is where instructions and data may be created and altered by a running process, and where data may be picked up for publication to the outside world.
- The input-output function is the process of transferring data between memory and the outside world through peripheral devices like a modem, hard drive, floppy disk; bring data in from the keyboard, or send it out to the display.

Different kinds of computer architectures can be created by designing a computer with more than one of these basic components. This allows for *overlap*, the ability to do two or more of the same kinds of things at the same time. Instruction-execution overlap can be achieved by having more than one CPU. Memory-access overlap is achieved by having more than one memory, and input-output (I/O) overlap is achieved by having more than one I/O port. Let’s focus on multiple CPUs first, because this is the essence of a multiprocessor.

Overlap with Multiple CPUs:

The number of CPUs inside the computer and how they are interconnected form the basis for one multiprocessor naming convention. Michael J. Flynn described four basic kinds of computer architectures that represent the four ways that CPUs and physical memories can combined inside a single computer. These are known today as the uniprocessor, multiprocessor, array processor, and the pipeline processor.

A *uniprocessor* (UP) is a single-processor computer that contains just one CPU. It is described in detail at the beginning of this chapter (Figure 2.1).

A *multiprocessor* (MP) has two or more processors, but it has only a single, shared memory which may be accessed by any or all of its CPUs, simultaneously.

Multiple CPU architectures allow more than one program to run at the same time. These machines must be programmed very carefully, however, because when two programs share data, and they can be running at the same time, there is a danger of them making simultaneous updates to that data. We’ll cover these kinds of problems and how to avoid them in Chapter 6, in “An Introduction to Network Programming in C.”

An *array processor* is a special-purpose computer containing a *master* processor and several slave processors, each with its own physical memory area. The benefit of an array processor is its ability to execute the same instruction simultaneously on any or all of its slave processors. The master processor sends an instruction to selected slave processors, but each instruction acts only upon the data in that processor's local memory; but that's a *good* thing!

The ADD instruction, for example, usually contains the memory locations of three variables named A, B, and C.

ADD A,B,C

When this instruction executes, the two numbers A and B are added together, and their sum is placed in location C. (A, B, and C are specified as offset addresses in each processor's memory.) The master sends the same instruction to each of its slave processors, but each of their memories has a different pair of numbers at its A and B addresses. The result is that each member in a *list* of A values is added to a corresponding member of a *list* of B values, and their sums are stored in a *list* of C locations. A list of numbers is called an *array* in most programming languages; hence the name array processor. (Mathematicians and engineers also refer to a list of numbers as a vector, so this architecture is also referred to as a vector processor.)

One early implementation of an array processor was the ILLIAC IV, designed and built at the University of Illinois in Champaign-Urbana. Note that modern array processors may employ more sophisticated internal hardware that serves the same purpose as the separate processors described above.

Last, and perhaps the most interesting of all is the *pipeline* processor, a highly specialized machine that exploits overlap at the instruction level using only one CPU. It does this by loading many instructions into its instruction *pipe* and performing the steps necessary to execute them all at the same time, in parallel, where possible. Some instructions can depend on the results of others, however, reducing the effectiveness of executing a pipe-full of instructions.

Tightly vs. Loosely Coupled Multiprocessors:

A tightly coupled MP has all of its CPUs and a single shared memory inside the same box. A loosely coupled MP is a lot of individual computers, interconnected via a network. These two variants of the same basic structure are called UMA and NUMA, respectively.

UMA (Uniform Memory Access) refers to a tightly coupled MP, where every CPU has direct access to the same central memory. Each CPU can access memory data in the same (uniform) amount of time.

NUMA (Non-Uniform Memory Access) implies a loosely coupled MP. Each CPU has direct access to its own local memory but not directly to those of the other CPUs. Access to their own memory data is very fast compared to the time it takes to get data from another CPU's memory. Remote memory access is done over an external data network (like a backplane or a local area network). Access times are non-uniform across CPUs.

NUMA can be further separated into NUMA and NORMA architectures. (NORMA stands for NO Remote Memory Access through hardware. Not all textbooks agree on this point, however.)

In a NUMA system, each computer *board* has its own CPU and memory. It is slid along rails into a rack mount, where it connects to a combination power supply and high speed communications bus, called a backplane. When a needed memory address is on the same board, its data appears in the local memory data register. When a memory address refers to a location on some other board, the backplane hardware electronically transfers the data into the local memory data register from the remote board, over the backplane. Remote memory access appears to be identical to local memory access, but backplane transfer time, fast as it is, is much longer than the time it takes to access a local memory.

In NORMA systems, each node is a standalone computer. When one computer needs memory data from another, software catches the remote memory access and must send an inter-process communication to request that data from the remote machine. All application software stops until the remote machine sends that data back to the requesting system. When the data is moved into its expected location, management software resumes all the application processes again. These kinds of computers are also called a distributed shared memory computer, or DSM.

All this alphabet soup may seem unnecessary to the outside observer, but these systems display very different properties, and computer people often need to talk about these differences. These simple alphabetic labels make our conversations easier!

Summary:

A uniprocessor has three basic components, which may be duplicated to develop a new kind of computer, called a multiprocessor, or MP. A variant of the MP is an array, or vector processor; another variant is called a *pipeline* processor.

The MP was such a successful architecture that it evolved into two types, called tightly coupled and loosely coupled MPs.

A tightly coupled MP is also called a UMA, for uniform memory access, because each CPU can access memory data at the same (uniform) amount of time. This is the true multiprocessor.

A loosely coupled MP is called a NUMA. Each of its node computers can access their local memory data at one (relatively fast) speed, and remote memory data at a much slower speed.

When the hardware in a backplane, for example, moves remote data from a remote node into the memory data register of the requesting node, we call it a NUMA. If some sophisticated software must manage the data movement, positioning the remote data in the local memory area, we call it a NORMA.

Remember that a NORMA is just a lot of independent computers tied together over a network. Special software runs on each node to *emulate* a true multiprocessor by resolving each local reference to a remote memory location.

Even without this special software, however, a network of PCs can serve as a parallel processor when pieces of a large application are sent across the network to remote nodes for simultaneous processing in parallel. It is also called a loosely coupled MP, or multicomputer, but we often call this kind of computer a **cluster computer** to distinguish it from a multiprocessor emulator.