

# **Architectures of Parallel Computers**

*(Parallel Programming and Parallel Algorithm)*

Daniel Schulze Zumkley

Supervisor: Prof. Dr. Herbert Kuchen  
Advisor: Philipp Ciechanowicz  
Information Systems  
Practical Computer Science

---

## Contents

1	Introduction.....	3
2	Basics .....	4
2.1	Flynn’s Taxonomy: The Concept.....	4
2.2	TOP500 .....	4
3	Network Connections.....	5
3.1	Shared-Media Networks.....	5
3.2	Switched-Media Networks.....	6
4	The three most popular parallel computer architectures.....	9
4.1	Processor Arrays .....	9
4.2	Multiprocessors .....	11
4.2.1	UMA Multiprocessors .....	11
4.2.2	NUMA Multiprocessors .....	14
4.3	Multicomputers .....	15
4.3.1	Asymmetrical Multicomputers .....	16
4.3.2	Symmetrical Multicomputers.....	16
4.3.3	Asymmetrical vs. Symmetrical Multicomputers .....	17
4.4	Flynn’s Taxonomy: Allocation of the Systems.....	18
4.4.1	SISD .....	19
4.4.2	SIMD .....	19
4.4.3	MISD .....	19
4.4.4	MIMD .....	19
5	Shared-Memory and Distributed-Memory Programming .....	21
5.1	OpenMP .....	21
5.2	Message-Passing Interface (MPI) .....	22
6	Outlook .....	24
	Bibliography .....	25

## 1 Introduction

It is not only since global warming took center stage in public interests that the most powerful computers in the world are used for highly calculation-intensive tasks. For several decades major universities, scientific research laboratories and military agencies have been profiting from the performance of so called “supercomputers”. Molecular modeling, weather forecasting, physical simulations of airplanes and wind tunnels as well as simulations of detonation of nuclear weapons are only a few examples of the many operational areas for these systems. Since the beginning of these computers, the users have been looking for more and more computational power to tackle increasingly complex problems. [Hu08]

In the 1960s the first real attempt was made to implement a system that allows parallel processing. The technique of parallel computing was so expensive that it was only used for commercial systems for a long time. While the same is still true for the current high-end systems, the development of parallel processing has become so advanced, that the majority of commodity computers contain some of these techniques today. For example, dual-core or quad-core processors are already irreplaceable in our everyday life.

There are many approaches to realizing parallel processing. This paper will give an overview of the most common architectures that provide these skills. We will not only name the methods but also see how they are built, what their development is and how they are used today.

In the second chapter we will learn more about Flynn’s taxonomy and the TOP500. With these ideas in mind, we can classify all the architectures we will get to know in chapter four. Furthermore, we can assign the different approaches to today’s fastest supercomputers. Before we do this in chapter four, we will take a look at two methods of creating a network between multiple processors in chapter three. These are called shared-media network and switched-media network. Chapter four contains the previously mentioned three most common architectures for parallel computers. With the understanding of these, we can fill in and explain Flynn’s Taxonomy afterwards. In chapter five we will see two application programming interfaces (OpenMP and MPI), which can be used to create parallel programs. The paper will close with a short outlook on the parallel computer architectures in chapter six.

## 2 Basics

### 2.1 Flynn's Taxonomy: The Concept

The best-known and most used classification of parallel computer architectures is Flynn's taxonomy. It categorizes the different approaches by the number of available instruction streams and data streams. These can be distinguished into single or multiple streams. An instruction stream is the sequence of instructions coming from one of the computers. A data stream is a sequence of data that is needed to process the instructions. Knowing these details about a parallel system gives us the potential to assign it into one of the four following categories [Ro90]:

SISD	single instruction stream, single data stream
SIMD	single instruction stream, multiple data stream
MISD	multiple instruction stream, single data stream
MIMD	multiple instruction stream, multiple data stream

Table 2.1: Flynn's taxonomy

In the following chapter we will learn more about different architectures of parallel computers. Knowing this taxonomy, we can assign all the upcoming approaches to one of these categories.

### 2.2 TOP500

Since 1993 the TOP500 project publishes a list of the 500 most powerful computer systems in the world. Twice a year this list is compiled by Hans Meuer of the University of Mannheim, Germany, Jack Dongarra of the University of Tennessee, Knoxville, Erich Strohmaier and Horst Simon of NERSC/Lawrence Berkeley National Laboratory. High-performance computer experts, computational scientists, manufacturers, and the Internet community in general help to maintain the facts.

The computer systems are ranked by their performance on the LINPACK Benchmark, a measure of a system's floating point computer power. To avoid scams with systems that are just built to create a good LINBACK performance, they only accept computers on their list which can be used to solve a range of scientific problems. [To09]

### 3 Network Connections

If a computer system has more than one processor, these processors need to interact. They need a way to communicate and coordinate the processes which need to be executed. There are two types of interconnections that can be used to realize the correlation. We have to distinguish between systems in which the processors share the primary memory and systems in which all processors can just access their own local memory. In the first case we talk about shared media and in the second we need switched interconnection media. [Qu04]

#### 3.1 Shared-Media Networks

The interconnection network with the easiest structure is the shared-media network (see figure 3.1). All processors use one single transmission medium to communicate to each other. Therefore only one of them can use the network at a time. [DYN03]

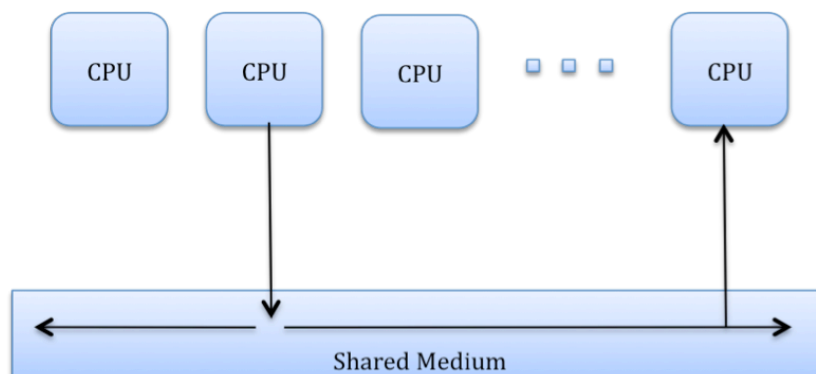


Figure 3.1: Shared-Medium Network

The coordination of the messages that go through the bus is not the task of the network itself. In this approach it stays passive and doesn't send any messages. The processors in contrast use requester, driver and receiver circuits to manage the exchange of data and addresses.

A unique performance of this approach is the feature to support atomic broadcast. The ability of all processors to monitor the shared-medium network and receive all messages sent through this is very important for the efficient support of many applications as well as for preventing the cache coherence problem (see chapter 4.2.1). [DYN03]

If all processors have to use the same transmission medium to communicate, the bandwidth becomes the bottleneck. To keep the performance of the system on a level

according to the performance that the connected processor can reach, this bandwidth is one major point to consider.

While there are many different approaches for this situation, Ethernet has become the standard over the last years. There are several specifications of this technology that vary in bandwidth. [SK09]

Ethernet	10 Megabit/s
Fast Ethernet	100 Megabit/s
Gigabit Ethernet	1 Gigabit/s
10 Gigabit Ethernet	10 Gigabit/s
40 Gigabit Ethernet <sup>1</sup>	40 Gigabit/s
100 Gigabit Ethernet <sup>1</sup>	100 Gigabit/s

Table 3.1: Ethernet standards

In the TOP500 more than 50% of the ranked computers work with Gigabit Ethernet as their networking technology. Infiniband is with over 35% the second most used one. A few years ago the Myrinet system added it's networking approach for many supercomputers in the list. Since 2004 however, this number decreases to only 7 systems in the current TOP500. [To09]

### 3.2 Switched-Media Networks

When you build a computer system with more than one processor, of course the scalability is an important part. Since the bus becomes a bottleneck in shared-medium network, the switched-media network reverts back to a different idea. [DYN03]

We can distinguish between direct and indirect networks. In a direct network every switch node is connected to exactly one processor node and one or more other switch nodes. In an indirect network a switch node can be connected to more than one processor node and switch node or just to switch nodes. [Qu04]

A few years ago these two designs where handled as very different approaches. However, the difference between these two classes of networks has been neglected

---

<sup>1</sup> These technologies are still in development.

more and more due to the development of router architectures, algorithms and switch designs which makes the disguising a question of topology. [DYN03]

Before we take a look at some switch-media network topologies, we need some parameters to evaluate the different approaches. [Qu04]

Diameter	Largest distance between two switch nodes in the network. Lower diameters are preferred. (Complexity)
Bisection width	Minimum number of edges between switch nodes that must be removed in order to divide the network into two halves. High bisection widths are preferred. (Complexity)
Edges per switch node	A constant network-independent size is preferred. (Scalability)
Edge length	A constant network-independent size is preferred. (Scalability)

Table 3.2: Parameters of switch-media network topologies

Switch-media network topologies that are found very often in commercial parallel computers are 2-D mesh, hypertree, butterfly and hypercube. Here you can find a short overview over the different designs: [Qu04]

Switch-media network topology	Processor nodes	Switch nodes	Diameter	Bisection width	Edges/nodes	Constant edge length
2-D mesh	$n = d^2$	$n$	$2(\sqrt{n} - 1)$	$\sqrt{n}$	4	Yes
4-ary hypertree	$n = 4^d$	$2n - \sqrt{n}$	$\log n$	$n / 2$	6	No
Butterfly	$n = 2^d$	$n(\log n + 1)$	$\log n$	$n / 2$	4	No
Hypercube	$n = 2^d$	$n$	$\log n$	$n / 2$	$\log n$	No

Table 3.3: Switch-media network topologies

When comparing the parameters the following facts stand out:

- With the exception of 2-D mesh, all designs have a logarithmic diameter
- Only 2-D mesh does not have a bisection width of  $n/2$
- Except for hypercube, all designs have constant edges per node
- Only 2-D mesh has a constant edge length

## 4 The three most popular parallel computer architectures

There are many approaches to realizing a parallel computer architecture. This paper narrows the options down to the three most common ones.

### 4.1 Processor Arrays

“A processor array is a vector computer implemented as a sequential computer connected to a set of identical, synchronized processing elements capable of simultaneously performing the same operation on different data.” [Qu04]

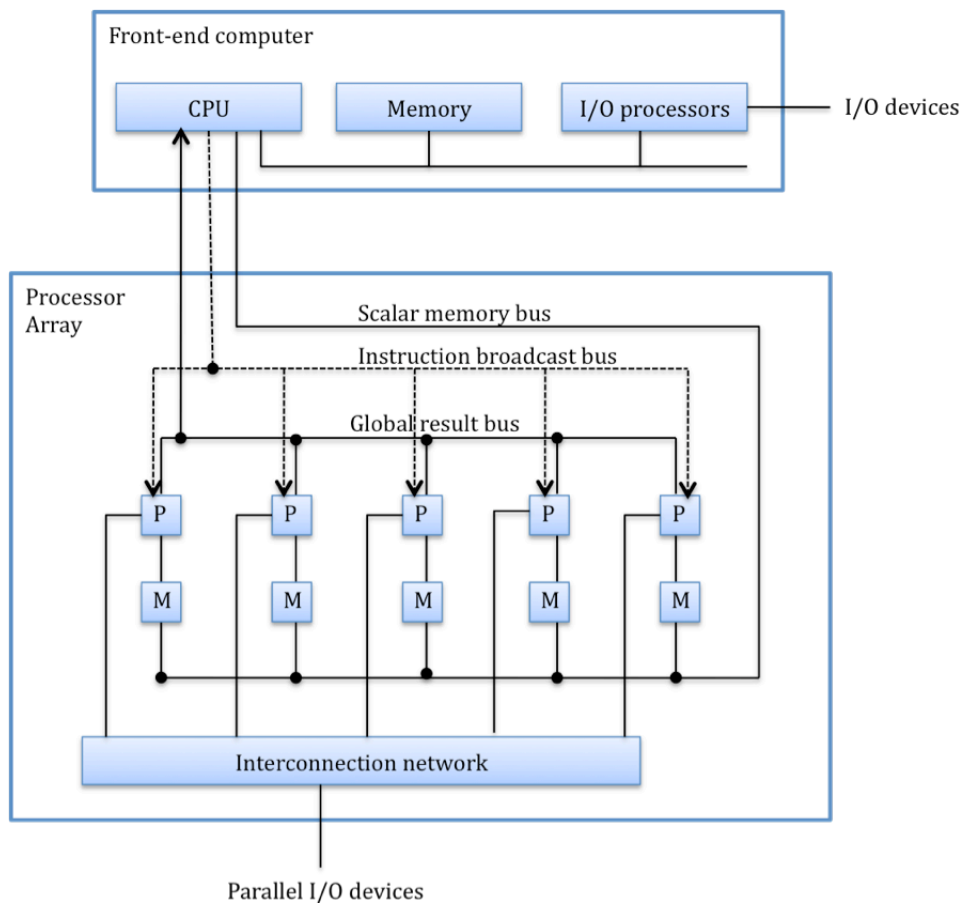


Figure 4.1: Architecture of a generic processor array

As you can see in figure 4.1 this architecture consists of many processor-memory pairs, which are instructed by one front-end computer. The memory of this front-end computer contains the commands that are sent to the processors in the array. All the receiving processors get the same instructions simultaneously. That means they all

perform the same operation on the individual data that is saved in their memory. The result is stored in the memory of the front-end computer again.

The main advantage of this approach is that all of these manipulations of data in the different processors are stand-alone operations but are executed at the same time. Using this functionality allows for not only performing thousands of parallel data manipulations (depending on the amount of processors used in the array), but also prevents conflicts in the data stream so that bypasses are not needed. This saves a lot of work and costs when building this system. [Me05]

The most famous computer using processor arrays is the “Earth Simulator”. “It is able to run holistic simulations of global climate in both the atmosphere and the oceans down to a resolution of 10 km.” This supercomputer was built in 2002 and is composed of 5120 processors and ten terabytes of memory. It was rated the fastest computer in the world for two years. [To09]

Today there is only one computer left in the TOP500, which is based on processor arrays. [To09] The decreasing consideration of this architecture is due to the many disadvantages that come with this approach.

The most obvious drawback of using processor arrays is the outcome of strict data-parallel solutions. Many problems cannot be solved by this method. Others might be solved eventually but very ineffectively, e.g. if the problem includes conditionally executed parallel code. The restriction of executing only one instruction at a time would require if-then-else statements, which will protract the whole process. The fact that processor arrays are naturally single-user systems and that they don't scale down very well also have a share in the growing unpopularity of this architecture. [Qu04] In the end, one of the biggest reasons for this development is losing one of the main benefits. Control units used to be very expensive which lead to a clear cost advantage of parallel architectures based on processor arrays. Through the development of CPUs becoming cheaper this price difference became insignificant.

In order to classify this architecture in one of the four categories from Flynn's taxonomy, we have to count the instruction streams and data streams. As said before, one command will be executed in many processor-memory pairs, so we have a single instruction stream but multiple data streams (SIMD).

## 4.2 Multiprocessors

Unlike the processor arrays, multiprocessors can be built out of commodity CPUs. These multiple CPUs share the memory, so “the same address on two different CPUs refers to the same memory location.” [Qu04]

There are two types of multiprocessors. In the first type, the complete primary memory is in one place. The idea for the second one is to distribute the primary memory among all processors in the system. According to this the two multiple-CPU computers are called „Centralized Multiprocessors“ and „Distributed Multiprocessors“. [Qu04]

### 4.2.1 UMA Multiprocessors

A centralized multiprocessor is a multiprocessing architecture in which multiple CPUs are connected to a single shared primary memory. This type of architecture is also called “uniform memory access” (UMA) multiprocessor or “symmetric multiprocessor” (SMP). [Qu04]

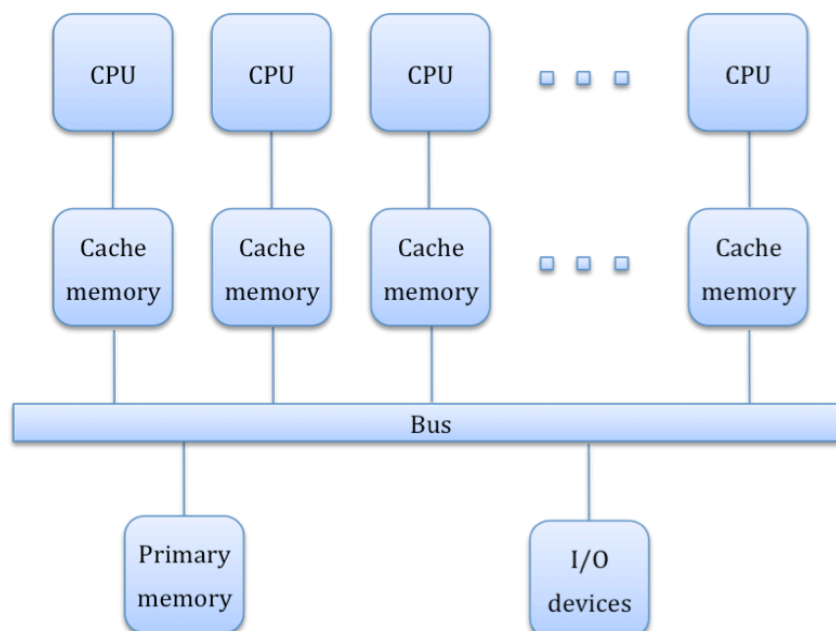


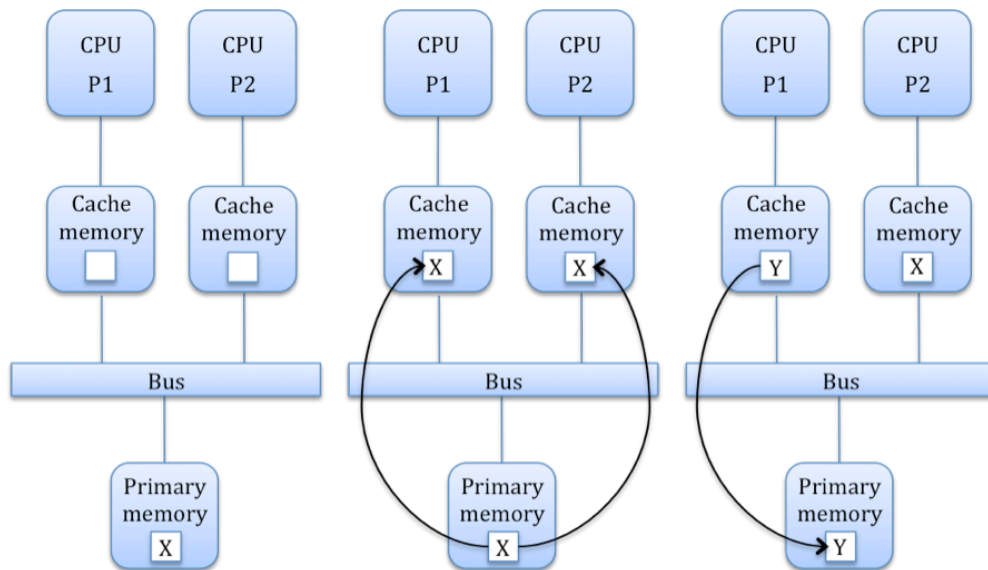
Figure 4.2: Architecture of a generic centralized multiprocessor

All of these names derive from the simple composition of this approach. As shown in figure 4.2, it is based on a normal uniprocessor with a CPU and the appendent cache,

connected by a bus to the I/O devices and the primary memory. This standard design is symmetrically expanded with multiple pairs of CPUs and caches.

Since every processor has its own cache, the memory access time is reduced. Because only one bus is used for the communication between the processors, no network connection is needed, which would slow down the process. [BBKS09] This is a useful effect to enhance the system performance, but at the same time causes a problem with this procedure. All of the processors share the same bus to communicate with the primary memory, so the bandwidth of this bus is a usual bottleneck of the system performance. The amount of processors used in such an approach is very limited because of this. [YZ95]

Working with multiple processors, the data used can be split into two categories. If it is used only by one processor it is private data. Data used by more than one processor is called shared data. In the last case you can imagine the different processors being linked to a shared pointer. This pointer refers to list of tasks that have to be executed. The processors successively read the address of the list item that is pointed to before they advance the pointer one step. [Qu04] These actions create a well known problem, known as the cache coherence problem. A simple example (figure 4.3) helps to understand the issue. Two processors P1 and P2 read the same block B from the shared primary memory. Both find the value X, which they save in their own cache. Now processor P1 modifies the value in its cache to Y and writes it back into block B on the primary memory. The cache of processor P2, which refers to block B wasn't affected by this change and still shows the old Value X. [YZ95]



Figur 4.3: Cache coherence problem (example)

There are several mechanisms to solve this problem. The one that is used most of the time when working with centralized multiprocessor is called “Snooping”. The idea is that every cache “snoops” the bus for any command coming from other processors that affects it’s currently saved data. With this information four different conditions (MESI) can be defined for the copies inside the cache: [BBKS08]

<b>Modified</b>	The value has been modified inside the cache, so it’s no longer the same as in the primary memory. The new data needs to be copied into the primary memory before other processors are allowed to access it again.
<b>Exclusive (unmodified)</b>	The value is not modified. It’s still the same as in the primary memory and the only copy of this data.
<b>Shared (unmodified)</b>	There is more than one copy of the value. They are all consistent to the copy in the primary memory.
<b>Invalid</b>	The copy is invalid.

Table 4.1: MESI protocol

With this extra information rules can be implemented that can provide a conflict free cooperation between the processors. (e.g. Only one processor gets the right to modify the data item. At the same time it is blocked for all others. After the block is removed, the other processors have to start the read command again to get the new value.)

This is the most common architecture when more than one CPU is used in the same system. The popular multi-core processors that you can find them in almost every commodity computer today are based on the same idea. Here the approach is realized with two or more cores inside one processor.

While it is very useful in everyday life, today's supercomputers are not built this way anymore. The restricted amount of processor because of the limited bandwidth let this architecture become out-of-favor for the developers of high-end supercomputers. [To09]

### 4.2.2 NUMA Multiprocessors

The second type of multiprocessors avoids the strong limitation of CPUs through the bandwidth by distributing the primary memory among the processors. This leads to the fact that every processor can use its local memory or the ones that are connected to the other processors (non-local). [So97] The different physical distances between processor and memory entail variable access times. This difference is known as the NUMA factor. [To09] According to the UMA (Uniform Memory Access) this approach is called NUMA (Non-Uniform Memory Access).

Building a system this way allows for reducing the memory access time as well as enhancing the aggregate memory bandwidth. In addition to this, the scalability of I/O devices is also improved. [Qu04] (cf. figure 4.4).

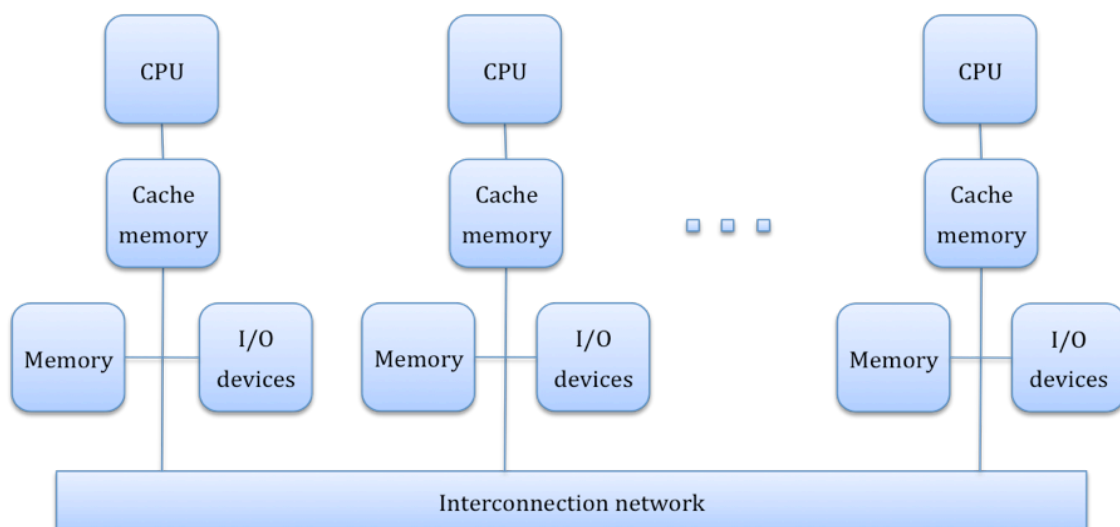


Figure 4.4: Architecture of a generic distributed-memory

While this approach allows us to use more processors at the same time than a UMA does, each still processor brings its own cache memory with it. The cache coherence problem is not only there again, but in a higher dimension, because there can be so many different caches. Snooping (c.f. chapter 4.2.1) only works efficiently for a small number of processor connected trough a bus. When using an interconnection network only the processor should be involved, which are interested in the cached block. For this reason there is another approach to keep the cache coherency of a NUMA. It's called dictionary-based cache coherence protocol. [PDX07] In this dictionary the states of all lines, which are caches at the moment, are kept in a shared list. If a processor addresses a block, the list can tell if other processors cache it already or not. If this list just saves this basic fact (cached or not) it's called a coarse list. A precise list also saves the information in which processors are involved. [SL05]

Bull NovaScale 5000 series, HP Superdome and SGI HLRB-II-Altix 4700 are some examples for commercially produced NUMA machines. [AS07]

Coming back to Flynn's taxonomy, we notice that multiprocessors have a multiple instruction stream as well as a multiple data stream no matter if the memory is shared or centralized. Therefore this architecture belongs to the MIMD category.

### 4.3 Multicomputers

At first glance a multicomputer looks like a NUMA multiprocessor, and it is very similar. But while in a NUMA system all processor share one global address space, a multicomputer has disjoint local address spaces. That means that every processor can only access it's own local memory. To get information from non-local memory the processors have to communicate via messages.

Since the same addresses of different processors don't point to the same spot on the memory anymore, the coherence problem we noticed with multiprocessors doesn't exist here. [Qu04]

Again we can distinguish between two different types. A multicomputer can be established in an asymmetrical or a symmetrical way.

### 4.3.1 Asymmetrical Multicomputers

As you can see in figure 4.5, in an asymmetrical system an interconnection network connects multiple computers. The user can access these computers only by entering the system through a front-end computer. In this case only the front-end computer needs the full functionality of a multiprogrammed operating system. The back-end computers on the other hand only get executing tasks. For these computers a simple operating system is enough, as they normally don't even have monitors or any I/O devices. [Qu04]

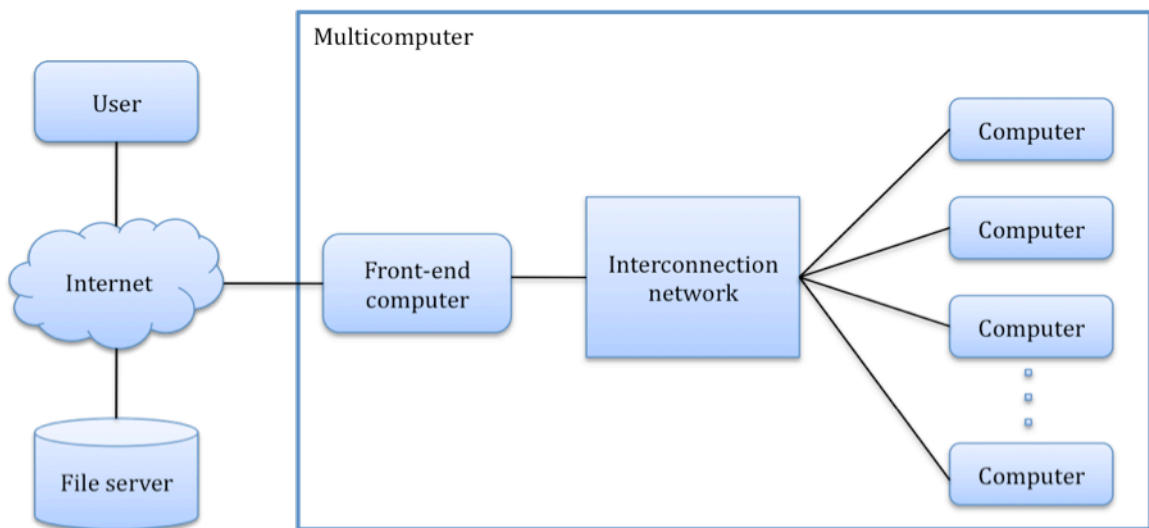


Figure 4.5: Architecture of an asymmetrical multicomputer

This is a currently a very popular architecture. In the Top 500 list from November 2009, 81 of the ranked computers are built as an asymmetrical computer. The highest ranked one is the Blue Gene/P Solution on place four. This system includes 294192 cores and was developed in Forschungszentrum Jülich in Germany. [To09]

Again we have a system that allows us to send a multiple instruction stream to modify a multiple data stream. Therefore, according to Flynn, an asymmetrical multicomputer belongs to the class of MIMD.

### 4.3.2 Symmetrical Multicomputers

A symmetrical architecture looks a little different. Here the computers are not divided into front-end or back-end. All of them have the same functionality so the user can log into the system through any of the connected computers.

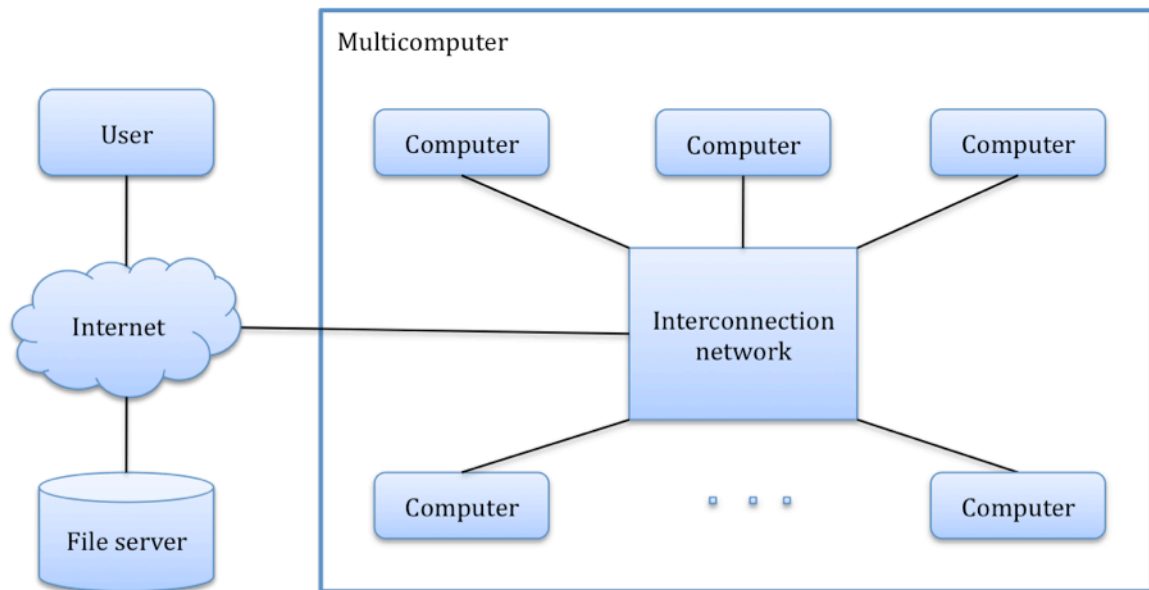


Figure 4.6: Architecture of a symmetrical multicomputer

This is the most widely used architecture for building supercomputers. Today 417 systems are listed in the Top 500. That is more than 80%. The number one ranked computer in the world is called Cray XT5-HE Opteron Six Core 2.6 GHz. It works with 224162 cores and has a performance of 1.75petaflops/s. [To09]

Symmetrical multicomputers fit in the same category as asymmetrical ones. (MIMD) They work with multiple instruction streams and multiple data streams as well.

### 4.3.3 Asymmetrical vs. Symmetrical Multicomputers

When comparing both approaches you can find advantages and disadvantages for each side. The following table is a summary of the main points to consider: [Qu04]

Asymmetrical Multicomputers	Symmetrical Multicomputers
The back-end computers can be primitive. (cheap)	All the computers have the same functionality. (more expensive)
If the front-end computers don't work the whole system is down.	If one of the computers is down, the performance of the system will shrink, but will still be running. The user can log into another computer.
The back-end computer can't communicate directly with the user. The front-end	Since every computer has the same functionality, they all can communicate to

computer always has to moderate. This can be time consuming and costly e.g. for debugging.	the user directly.
The front-end computer limits the system performance, even if the back-end ones could handle more instructions.	The full performance of every computer can be used.
Easy to build and to maintain.	<p>It is a lot more complex to maintain this system.</p> <ul style="list-style-type: none"> <li>- Create the illusion of a single computer</li> <li>- Balance the workload between the computers</li> <li>- Achieve high performance while the computers compete about cache space, memory bandwidth etc.</li> </ul>

Table 4.2: Asymmetrical vs. Symmetrical Multicomputers

#### 4.4 Flynn's Taxonomy: Allocation of the Systems

Now that we have learned more about the different parallel computer architectures we can now fill them into table of Flynn's taxonomy. (see figure 4.7)

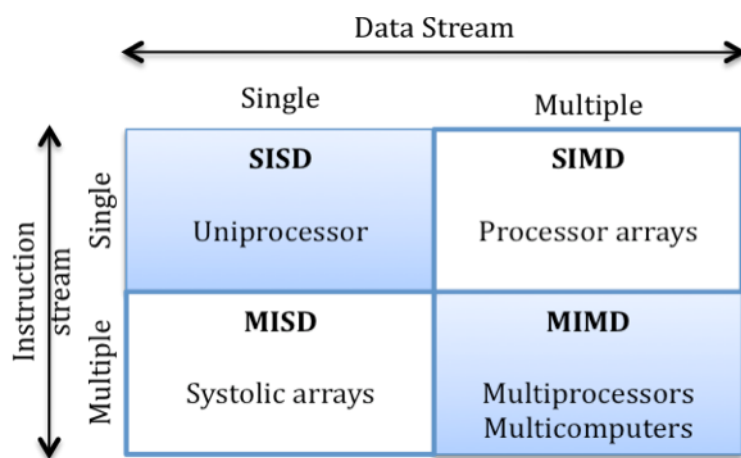


Figure 4.7: Flynn's Taxonomy of computer architectures

#### **4.4.1 SISD**

The SISD class is best represented by the famous von Neumann architecture. Pipelined processors are also included. Here several instructions are sent simultaneously, but only one is implemented per machine clock cycle. [Ko81]

Since there is no parallel computing possible in this category, the architecture is inconsiderable when developing supercomputers. The average PC is still based on this approach even though more and more use multi core processors that work with UMA technology. [To09]

#### **4.4.2 SIMD**

We already saw the processor array as an example of a system in which one instruction is executed on multiple data.

This architecture had its high point in the 80s when the Cray computers controlled the market for supercomputers. With the development of cheap commodity processors this approach became decreasingly popular. Today there is only one computer left in the Top500, which is still based on a processor array. [To09]

Even though the processor array approach is not relevant for the development of supercomputers anymore, the SIMD class still offers an important technology. Especially for calculations concerning multimedia files like pictures or videos, it occurs often that the same operation has to be executed on different data streams. [Fr09]

#### **4.4.3 MISD**

In a MISD computer multiple processors give multiple instructions to the same, shared data. This sounds more like a theoretical idea than a practical architecture. [CGWR93]

There are systems which are assigned to this class, e.g. systolic arrays. [Qu04] But in general this approach is very rare and not relevant for the goal of this paper. Because of this, we won't go into any further detail.

#### **4.4.4 MIMD**

Most parallel computers fall into this category. To avoid losing the overview it's wise to know the subcategories of MIMD computers. UMA and NUMA processors as well as symmetrical and asymmetrical multicomputers belong to this approach.

It can be seen as the most powerful of these four classes, because different instructions can modify different data simultaneously.

MIMD is by far the most used architecture for supercomputers.

## 5 Shared-Memory and Distributed-Memory Programming

We already learned that multiprocessors can work with shared or distributed memory among the processors. If the whole system is based on one single, shared address space the processors can communicate by using variables on this address space. On the one hand that makes the referencing of data very easy, it is like in programs for single processors. On the other Hand integrity of the shared data ads a whole new complexity.

In architectures in which every CPU has his own address space, the processors have to communicate between each other by using a network. This approach brings a new problem as well. Programs for this structure need a solution to distribute the commands to multiple processors with distinct memory spaces. And after that, all the results coming back have to be merged into one outcome. [Qu02]

### 5.1 OpenMP

“OpenMP is an application programming interface (API) for parallel programming on multiprocessors.” [Qu04] It can be used for C, C++ and FORTRAN, which belong to the most used languages for parallel programs.

The API consists of a set of compiler directives and a library of support functions. With the compiler directives OpenMP defines specific rules for the parallelization. The directives also communicate the regions of code that should be parallelized to the compiler. To make the work for the programmer even easier, OpenMP contains precompiler tools, which automatically convert serial programs into parallel programs.

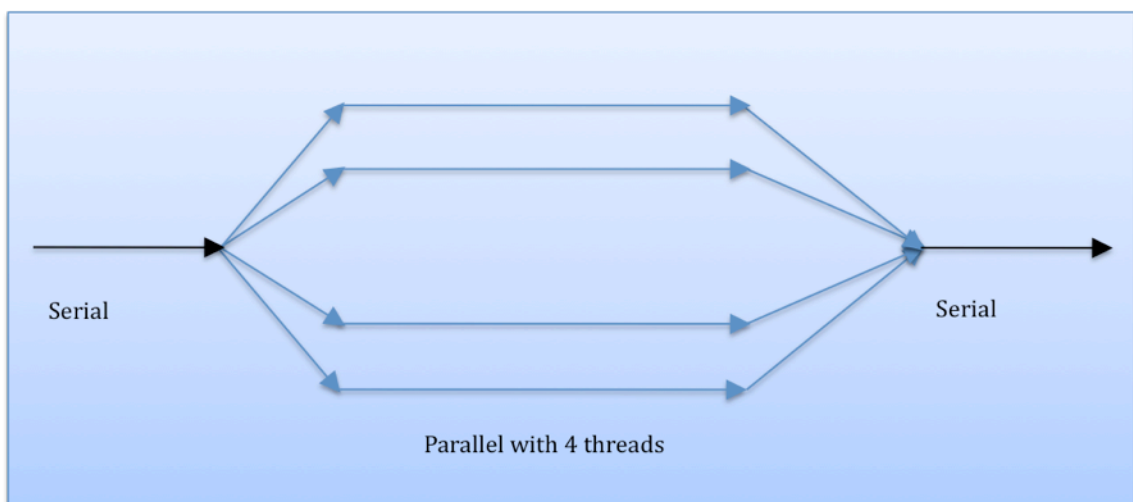


Figure 5.1: Program flow in an OpenMP execution model

OpenMP is based on the idea of multithreading. “A thread is an active execution sequence of instructions within a process.” [Qu02] According to the fork-joint principle, a master thread runs serially as long as it doesn’t hit a directive. When it reaches parallel code, it will fork off threads (see figure 5.1) so that multiple instructions can be executed at the same time. At the end of the parallel code, these threads rejoin in the master thread again.

## 5.2 Message-Passing Interface (MPI)

“The MPI (Message Passing interface) standard is the most popular message-passing specification supporting parallel programming. Virtually every commercial parallel computer supports MPI, and free libraries meeting the MPI standard are available for “homemade” commodity clusters.” [Qu04]

Like OpenMP MPI is an API that supports C, C++ and FORTRAN, but the course of action with this interface is very different. Because the processors don’t have the same address space in a distributed-memory system, they can’t use shared variables, but have to send data packets amongst each other.

A data packet contains the following information: “the sending process, the receiving process, the starting address in memory of the data to be sent, the number of data items being sent, a message identifier, and the group of processes that can receive the message.” [Qu02]

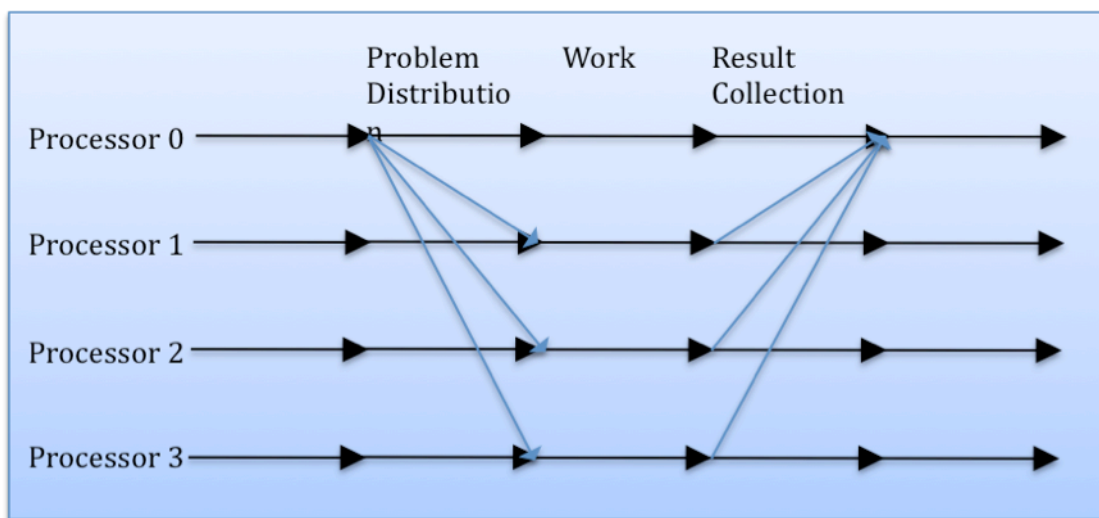


Figure 5.2: Program flow in an MPI execution model

In the very simple example in figure 5.2 you can see the general idea behind this approach. A master process sends tasks out to the other processors. They execute the instructions and send the results back to the master processor. This combines the data packets to one solution.

Comparing figures 5.1 and 5.2 you can immediately see a big difference in the design. While with OpenMP the non-master processors run only after threads got forked off, in the MPI approach all processors are active the whole time. [Qu02]

# 6 Outlook

We have gotten to know the three most common parallel computer architectures: Processor Arrays, Multiprocessors and Multicomputers. We've learned that we have to distinguish between different approaches inside these methods and by which supercomputers they are established. Considering Flynn's taxonomy there are four different categories for computer systems; SISD, SIMD, MISD and MIMD. The vast majority of parallel processing systems belong to the MIMD class. These are the most powerful computers and that's why even in the future most newly developed systems will follow the same MIMD idea. The SIMD architecture Processor Arrays have become decreasingly important over the last decade, but since this SIMD technique is extremely useful for multimedia calculations, the development in this field will continue as well.

The famous Moore's law says simplified: "Computer chips get twice as fast every year or two". Even though the development has slowed down a little bit, he was fairly right. [Br06] Considering this we can assume that the development of supercomputers will continue for years to come. To create faster and more powerful systems, the advancement of parallel processing will play an important role. Regardless of how fast a single processor can be in the future, working with multiple processors will always be faster.

The average PC user can also be excited about coming developments. Even if he never comes in contact with a supercomputer, the new techniques will continue to become cheaper and more affordable until commodity computers will contain it as well.

---

## Bibliography

- [AS07] Aad J. van der Steen: *Overview of recent supercomputers*, NCF Utrecht University, 2007.
- [BADS08] Marian Bubak, Geert Dick van Albada, Jack Dongarra, Peter M.A. Sloot: *Computational Science – ICCS 2008*, Springer, 2008.
- [BBKS08] Günther Bengel, Christian Baun, Marcel Kunze, Karl-Uwe Stucky: *Masterkurs Parallele und Verteilte Systeme*, Vieweg + Teubner, 2008.
- [Br06] David C. Brock: *Understanding Moore’s law: Four Decades of Innovation*, Chemical heritage Foundation, 2006.
- [CGWR93] Andrew McCulloch, Julius Guccione, Lewis Waldmann, Jack Rogers: *Large-Scale Finite Element Analysis of the Beating Heart*, High Performance Computing in Biomedical Research, p.44, CRC Press, 1993.
- [DYN03] José Duato, Sudhakar Yalanchili, Lionel Ni: *Interconnection Networks: An Engineering Approach*, Elsevier Science, 2003.
- [Fr09] Nicolas Fritz: *SIMD Code Generation in Data-Parallel Programming*, epubli, 2009.
- [Hu08] Yongge Huáng: *Supercomputing research advances*, Nova Science Publisher, 2008.
- [Ko81] Peter M. Kogge: *The Architecture of Pipelined Computers*, Taylor & Francis, 1981.
- [Me05] Dr.-Ing. Matthias Menge: *Moderne Prozessorarchitekturen, Prinzipien und ihre Realisierung*, Springer, 2005.
- [PDX07] Guoteng Pan, Qiang Dou, Lunguo Xie: *Two-Level Directory Organization Solution for CC-NUMA Systems*, Algorithms and Architectures for parallel processing, p142 f., Springer, 2007.
- [Qu02] Cory Quammen: *Introduction to Programming Shared-Memory and Distributed-Memory Parallel Computers*, ACM Crossroads, 2002.
- [Qu04] Michael J. Quinn: *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill, 2004.

- 
- [Ro90] Yves Robert: *The impact of vector and parallel architectures on the Gaussian elimination algorithm*, Algorithms and architectures for advanced scientific computing, p.20f, Manchester University Press, 1990.
- [SK09] Steven T. Karris: *Networks: Design and Management*, 2nd.ed., Orcahrd Publications, 2009.
- [SL05] John Paul Shen, Mikko H. Lipasti: *Modern Processor Design: Fundamentals of Superscalar Processors*, McGraw-Hill, 2005.
- [So97] Marc Songoni: *NUMA steps into the spotlight*, Network World No. 35, p.20, IDG Network World Inc, 1997.
- [To09] TOP500.Org, < <http://www.top500.org> >, November 2009.
- [YZ95] Marshall C. Yovits, Marvin V. Zelkowitz: *Advances in Computer Volume 40*, p.128f, Academic Press, 1995.