Chapter 3: Heterogeneous Clusters

Since the main methods to multiply matrices have already been described, it is now possible to pay all the attention to the installed clusters. In this sense, and as highlighted in the previous chapter, it is necessary to know with a relative high degree of detail the underlying computing architecture in order to analyze the proposed methods.

In the first place, the main characteristics of clusters installed to be used as parallel machines shall be described. Since local area networks have not been initially designed for parallel computing (at least in the contexts of scientific-numerical applications), the capabilities in terms of computation, processors interconnection, synchronization, and scalability must be identified as clearly as possible.

Being the local area networks characteristics properly defined, it is possible to analyze the parallel methods of matrix multiplication and, according to this analysis, decide whether its implementation in clusters could attain an acceptable performance. This analysis will show that the parallel computing methods of the proposed matrix multiplication are not suitable for heterogeneous clusters.

From the previous analysis, a new matrix multiplication algorithm is proposed in parallel, showing its main characteristics in relation to the computing load balances that each local area network has to carry out, the interconnection networks load balance and the memory requirements imposed by the algorithm. The different ways of applying the same matrix multiplication parallelization concepts in heterogeneous clusters to other problems will also be analyzed - though it is evident that this is really difficult to attain in general.

3.1 Clusters Characteristics

As highlighted in the first chapter, the local area networks make up the most advantageous parallel computing platform in terms of the relation cost/performance. And this relation cost/performance is even more useful in the case of employing the already installed networks of workstations. In this sense, and for the sake of the following description/discussion, all the local networks are considered as installed NOW, independently of the fact that the interconnected computers might be PCs, workstations or computers with symmetric processing (SMP: Symmetric MultiProcessing) [71]. Throughout this chapter, the terms and concepts

- local networks,
- computer networks,
- network of workstations, and
- PCs networks

are actually used as synonyms.

It is important to identify the characteristics of the computer local networks used for the parallel computation, since the algorithms proposed for solving most (if not all) the applications in parallel in general -including the matrix multiplication operation- must be assessed in this context. Even though it is true that most (if not all) the proposed parallel algorithms can be implemented on computer local networks with a greater or smaller degree of difficulty, it is also true that the performance can be really different. Since that, in general, the proposed operations parallelization is oriented to a type of architecture, the parallel algorithms implementation has to be analyzed considering the characteristics of the computers local networks used as parallel computing platform.

In the following subsections, the characteristics are identified depending on whether they are considered as belonging to homogeneous networks of workstations or to heterogeneous network of workstations. Initially, the same computers interconnection network - which, in fact, is used for message passing between the different processors -is described as a characteristic of the computers local network used for parallel computation.

All the local networks characterization has as reference the traditional parallel computers - more specifically, multicomputers. This baseline is important from two points of view:

- All the knowledge and experience acquired in terms of interconnection and processing hardware design. In this sense, there is a solid point of reference as from which it is easier to describe the characteristics and it is possible to compare and identify the similarities and differences.
- Applications parallelization and performance obtained. As highlighted before, the algorithms with reasonable performance in a parallel machine are normally oriented to the architecture of the same parallel machine. Thus, when identifying similarities and differences in terms of the traditional parallel machines, there is a tendency to simplify the parallel algorithms analysis proposed in general and, in particular, the ways of processing in parallel the necessary work for the matrix multiplication.

3.1.1 Characteristics of the Processors Interconnection Network

Most of the traditional parallel machines effort has been dedicated to processor interconnection networks. The processor interconnection network, and in particular its performance, is actually considered as fundamental in parallel machines [69] [71] [87].

For parallel computers with (or based on) distributed physical memory, the interconnection network can be clearly identified as the one that provides communication among processors. In other words, if the processors interconnection network is removed from a parallel computer with distributed memory, it is no longer a parallel machine and becomes a set of separate computers or CPU-memory modules unable to cooperate for the solution of the problem. In the case of parallel computers with physically shared memory, removing the processors interconnection network with the (only) memory thoroughly eliminates the possibilities of executing applications over the remaining hardware. Since the local area networks used for parallel computating are clearly distributed memory computers, the interconnection network will still be considered for the data delivery among processors (or, roughly speaking, computers).

In relation to the processors interconnection network flexibility, the aim is not only to find a way of transferring data between the two processors but also to have the maximum of communications at the same time. The typical examples in this sense are focused on the capability or incapability of all the possible processors pairs to be communicated at the same time, or the possibility of transferring information from one processor to the rest of them in a single step (or in a quantity of steps independent of the number of interconnected processors).

The flexibility of an interconnection network will define the simplicity (or difficulty) of the user's applications to solve the communication between its processes. The underlying idea is that each processor will always be in charge of the execution of one or more processes to be communicated to other processes assigned to other processor/s.

In terms of costs, there exists an invariant relation among the different interconnection networks possibilities: the higher the interconnection network flexibility and/or performance, the higher the cost. The growth in the cost varies according to the interconnection network that is used but, in several cases, increasing the parallel computer processors quantity implies a more than linear growth in the processors network cost. In the particular case of the installed local area networks, the cost is zero (in general, worthless) since they are already interconnected.

As previously mentioned, the processors interconnection network of a parallel computer built upon a local network is the network itself. In the particular context of installed workstation networks, local networks or LAN (Local Area Networks), the most used interconnection network is the defined in the standard protocol IEEE 802.3 [73] [109] [108]. This standard is initially known as the Ethernet network of 10 Mb/s due to its delivery capabilities of 10^6 bits per second. The characteristics of this interconnection network are very well defined and known in terms of hardware, and of flexibility and throughput.

In addition, most of the 10 Mb/s Ethernet network characteristics are similar to the 100 Mb/s Ethernet network, also called Fast Ethernet, in which only the parameters/indexes related to the communications performance are changed. This similarity is exemplified in - and also used up by - many of the communication hardware companies dedicated to the design and building of communication interface boards (NIC: Network Interface Card) with both data delivery capabilities, called 10/100 Mb/s net board. Additionally, and always within the norm 802.3, the Gigabit Ethernet has also been defined with a delivery capability of 10^9 bits per second [105] [76], and the definition of the standard for 10^{10} bits per second or 10-Gbps [110] is also being considered.

Figure 3.1 schematically shows the logical basic way in which the computers are connected in a local network using Ethernet. Notice that it is of bus type, where the main characteristics of each data transference are:

- there are no priorities and the access time to the media is unpredictable,
- there is only one sender,
- it occupies the only communication channel,
- it can have multiple receivers,
- the access to the media is CSMA/CD (Carrier Sense, Multiple Access / Collision Detect).

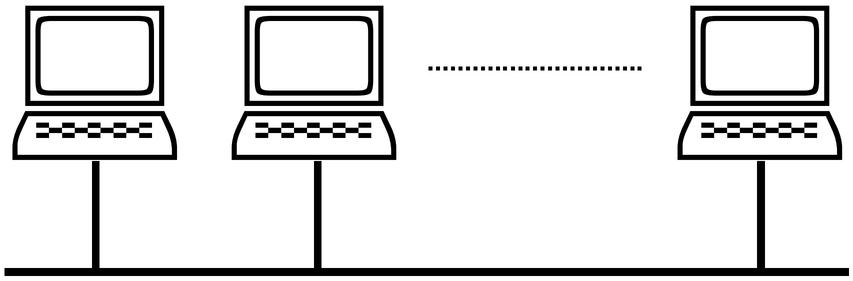


Figure 3.1: Ethernet Network.

The firs two characteristics clearly imply that there must not be more than one simultaneous data transference because there is actually only one communication channel shared by all the computers. The last characteristic turns the broadcast and/or multicast implementation very natural, where a computer sends a message that is received by the remaining ones or by a subset of the network remaining ones, respectively. The communication hardware initially adopted, in most of the installations, was based in coaxial cables, all of which allowed the equality between the physical topology and the logical one of Figure 3.1.

In most of the installations, the wiring rules used have been gradually changed towards the use of twisted pairs with hubs that are basically communications hubs and repeaters. Figure 3.2 shows that, from the wiring point of view, the network has a star wiring (topology), but since the hubs distribute the same signal in all the wires, the logical interconnection is still that of a bus.

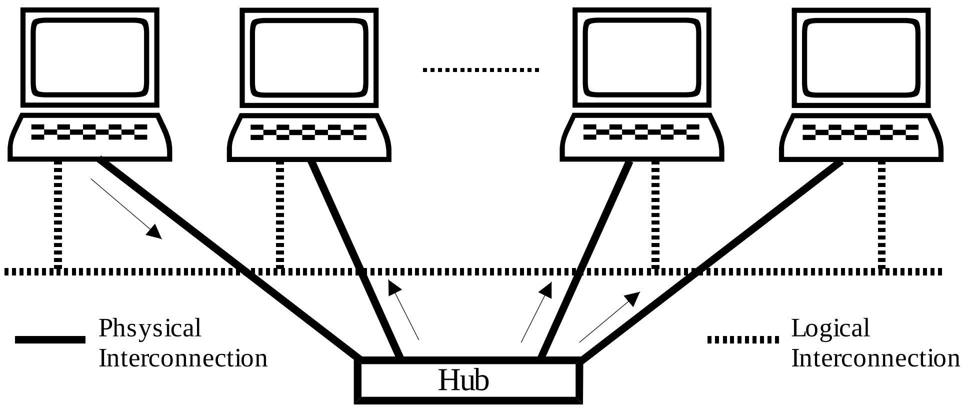


Figure 3.2: Ethernet Network with Hub.

Figure 3.2 also shows that the computer located to the left sends a datum, and the rest of the computers are capable of receiving it simultaneously - only if there is no collision. In this sense, each hub is not only a hub signal coming from each wire but also a broadcast repeater, i.e. that the signal coming from a wire is repeated towards all the others.

The use of switches instead of hubs in Ethernet networks is considered as a great improvement since that, apart from having all the characteristics of the hubs, the switches are capable of isolating point-to-point communications [109] [108] [118]. This isolation in the switches occurs when the hardware detects a point-to-point data transmission between two interconnected workstations and, thus, several point-to-point data transmissions can be carried out simultaneously. All the same, since it is defined in the Ethernet standard, all the collective operations - such as the broadcast at physical level - are kept and, in this case, the switch behaves as a hub.

Figure 3.3 shows an Ethernet network where two point-to-point communications can be carried out simultaneously thanks to the switch activity. The advantages provided by the

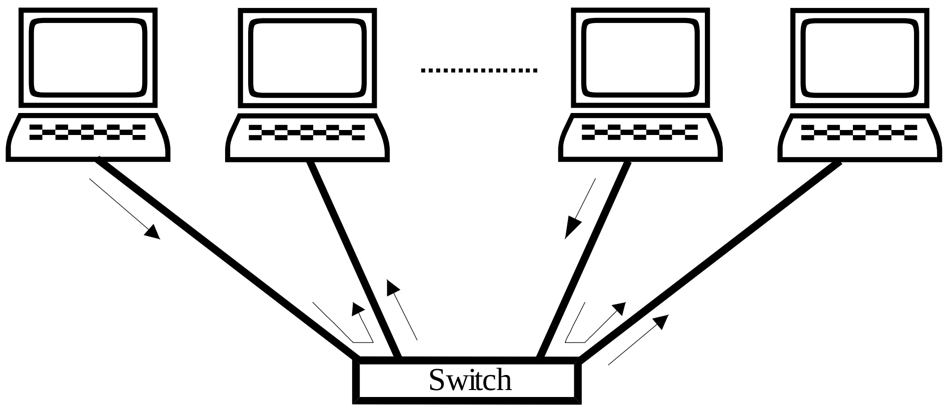


Figure 3.3: Ethernet Network with Switch.

switches over the hubs are evident, but it should also bear in mind the fact that the switches cost is significantly higher than the hubs', though they are rapidly extending in new local networks installations. On the other hand, the switches use is essential in computer network installations meant for parallel computation - of Beowulf type [19] [103] [111].

Hub-oriented wiring Ethernet networks are important not only because of the installations quantity actually functioning but also because they are clearly less costly than the other alternatives in the market. They are less costly in terms of the necessary hardware (boards, connectors and wiring) and with respect to the installation: from workforce (technicians) to recognition and starting-up on the part of the operating system. All of this necessarily reduces Ethernet network installation and maintenance costs.

The cost reduction represented by the hub-based networks with respect to the other workstations interconnection alternatives implies a great inertia in the installed Ethernet networks maintenance as well as in the installation of new networks with this hardware.

From the applications point of view - i.e. the parallel application processes that are being executed and need the interconnection network-, most (if not all) of them make use of Internet "super standards", namely: IP (Internet Protocol), TCP (Transmission Control Protocol), and UDP (User Datagram Protocol) [89] [97] [112] [32].

Thus, both at a physical level and in the interconnection network utilization protocols, most of the local networks are homogeneous and have common and well-known characteristics.

3.1.2 Homogeneous Cluster as a Parallel Machine

Actually, it is difficult to find a local network with equally interconnected computers, unless the local network has been recently installed. In most of the cases, "recently" implies no more than a few months, though it depends on the company in which is being used and the function of the network itself.

Still, it is worth considering the homogeneous case in order to identify/describe the characteristics of NOW from the point of view of the parallel computation since:

- It allows a clear-cut division of the characteristics, thus improving its analysis.
- All the characteristics identified for the homogeneous case will be shared by the heterogeneous NOW. In other words, workstation networks have characteristics that must be considered at the time of the applications parallelization, independently of the fact that the computers are homogeneous or not. From this point of view, heterogeneous computer networks add up other characteristics that should also be considered for the applications parallelization.
- The systems - such as Beowulf, or simply the local networks installed for the parallel computation- are usually homogeneous. In this sense, the important characteristics in an homogeneous environment should be taken into account not only in the already installed homogeneous networks but also in all these systems of parallel computation based on computers local networks.

Summarizing, the principal technical characteristics of homogeneous workstations networks that must be considered for parallel processing are:

- loose coupling,
- processors interconnection network low performance.

Coupling. Computers local networks are built from interconnected complete computers in order to share some type of resource. The classical resources that have been shared are storage space, storage data and printer/s. Thus, the hardware of the different computers is not generally related. In fact, the only condition for a computer to be connected to a local network is the installation of a communications interface card (NIC) and nothing more than the software routines dedicated to its handling. Thus, from the point of view of each computer computing architecture, it implies adding another I/O data device. Now, coming back to the consideration of the local network as a parallel machine, this implies that:

- The computers physical memory in the local network is completely distributed and there is no hardware means that facilitates its sharing.
- The processing in the local network is completely asynchronous and there is no hardware means that facilitates its sharing.

The only way of making a processor read or write a memory position in other machine is to use the communication network appropriately. Similarly, the only way of synchronizing the processors of each local network computer is to use the communication network appropriately. It is clear that both things are possible, but it is also evident that the local networks have not been designed with any of these purposes and, thus, performance penalties in order to attain the synchronization among processors and/or shared memory can be substantial in terms of performance.

Discarding the use of shared memory due to the performance penalty necessarily implies discarding the algorithms designed for the multiprocessors, which are parallel computers with shared memory. In the case of numerical algorithms, in general, and matrix multiplication algorithms, in particular, it often implies discarding the algorithms that have proved to be very effective in terms of successful performance in multiprocessors.

The processors synchronization problem does not seem to be as difficult as the distributed memory problem. In principle, the need of synchronization is not that strong in most of the algorithms - at least, there are no algorithms proposed in function of the whether machines with synchronized processors are used or not). On the other hand, the fact that the processors are not synchronized by hardware does not imply that it is impossible to synchronize them and, thus, if the synchronization is not so frequent, the impact on the performance will not be relevant. Normally, the synchronization frequency among processors not implying a performance loss is given directly by the interconnection network throughput indexes, more specifically by the communications startup.

Processors interconnection network performance. The performance of an interconnection network is directly related to the data transference time among processors of a parallel computer. This view of performance is not necessarily disjunctive from that of flexibility. In fact, the larger the simultaneous data transference quantity between pairs of processors, the greater the capacity or the number of data transferable by an interconnection network per time unit.

Although it is important the idea of bandwidth (transference rate) or asymptotic bandwidth given by the transferable data quantity per time unit, another important throughput index is the minimum communication time between two processors. This minimum time is basically known as communication initialization time (startup) or, according to other authors [124] [25], communication latency between processors. The classical form of communication time computation of an interconnection network bears in mind both the latency and the asintotic bandwidth according to the following equation [69] [68]

$$t(n) = \alpha + \beta n \tag{3.1}$$

where

- n is the information unit transferred (bit, byte, a floating point number representation with simple precision, etc);
- α is the communication latency time (startup);
- β is the communication network asintotic bandwidth inverse value, i.e. that $1/\beta$ is the asymptotic bandwidth.

The main drawback of workstation interconnection networks in terms of performance is that they were not designed for parallel computation. In this sense, several orders of magnitude are placed under the rest of the interconnection networks of traditional parallel computers. For this reason, it is really important to assess its performance from the point of view of the user's processes that make up a parallel application.

It is very difficult to properly quantify the relation of the local networks with the traditional parallel computers interconnection networks in terms of the mentioned throughput indexes. However, it is true that the worst relation is given in relation to the initialization time of the messages to be communicated between two processors. Moreover, in the context of the heterogeneous local networks, the communications initialization time usually depends on the computers used since the times of the calls to the operating system and their consequent overload in terms of the used protocols (or protocols stack) maintenance and management are involved. In a level closer to hardware, the times of the

- access to memory;
- initialization-utilization of DMA (Direct Memory Access) channels, if used; and
- related interruptions management and/or interface with net board of each computer to be communicated

are also involved.

On the other hand, the interconnection network performance is directly related to the performance and the granularity of the parallel applications executable on the computer. Every communication time tends to degrade the total parallel application execution time, unless the capability of overlapping the computation and the communication in time is available and used to the utmost. This capability is another characteristic used up in the traditional parallel machines interconnection networks.

From the performance point of view, if the communication time to obtain a result in the processor P_1 is equal or greater than the computing time necessary for its computation, then the most reasonable operation to carry out is the local processing (in P_1), saving time and/or complexity of the application.

Summarizing, the local network performance is lower than that of the traditional parallel machine processors interconnection networks from several points of view:
- Latency and bandwidth.
- Overlapping capability.
- Latency heterogeneity depending of the machine heterogeneity.

3.1.3 Heterogeneous Cluster as a Parallel Machine

The differences in computers relative computing speed in heterogeneous clusters are the most important and recent issues in terms of what has been identified as interconnection networks and local networks with homogeneous computers. On the other hand, the likelihood of a heterogeneous interconnection network is really low in the installed local networks and, thus, is discarded.

In general, the installed local area networks can be highly heterogeneous in terms of processing hardware or of the computers interconnected in the local network. In the local networks, with a minimum existence (and evolution) time, multiple workstations models, some parallel computers and/or symmetric multiprocessing computers (SMP), and multiple PCs models can be found. In the case of PCs, it is also possible that machines with the same processor and operating with the same clock frequency might have different processing capability depending on the differences of, for instance, speed access to memory, external cache memory capability and the system bus operation frequency. The differences between connected computers in a local network are normally related to:
- Local network existence time, with its consequent impact on computers replacement and/or update. In the case of replacement, there is a tendency to keep a minimum of computers available for each user and, as the time passes by, the hardware is no longer manufactured and is replaced by another one (generally, more rapid). In the case of update, there is a tendency to acquire the best processing hardware for the same cost, as it is the case of basic components such as processor, principal memory and disks.
- Evolution in terms of requirements. The reasons and conditions for which the installation of a local network has taken place are not necessarily kept invariant. The change of tasks or the addition of new tasks carried out in a local network generally imply the addition of new computers, which may be specific for the tasks to be developed in the environment where the local network is installed.

In order to obtain the maximum possible throughput in a heterogeneous network, a proper processing load balance has to be necessarily carried out. If it is assumed that all the computers have to carry out the same quantity of processing, the use of a larger number of computers will imply a worse performance. In fact, the complete parallel machine works in terms of the processing time of the slowest computer used because it will be the last to complete the assigned computations and, thus, the complete processing will not be finished until the worst computer (in terms of processing performance) has stopped.

The basic idea for balancing the computers processing load in a local network is simple as regards its definition, though it is not simple for its implementation in general: if a computer, ws_1, is *n* times more speedy for carrying out the same processing than another one, ws_2, then ws_1 must have a task *n* times more complex in terms of computation than ws_2. Thus, this basic idea simply means that, at the time of assigning computing task/s, the relative speeds among computers have to be considered.

Another characteristic to be mentioned in relation to heterogeneous networks is the difference in the storage capability of the main memory. However, in most of the installed local networks, this difference is neither really big nor proportional to the difference between relative speeds. Making reference to the previous example, it is really difficult to find that, if a computer ws_1 is *n* times more speedy that another one, ws_2, ws_1 memory size is *n* times bigger than ws_2. Actually, it is rather frequent to find memory sizes quite similar in computers with very different relative speeds. In the extreme case of finding out that the difference between memory sizes is really big, there exists the possibility of recurring to the computation of the relative speeds among computers in function of the swap memory utilization so that the computers with the lowest quantity of available main memory carry out the computations making use of the swap memory; this will be thus directly reflected in its computing speed.

In a more general context of heterogeneous processing, the variations are also higher. Reports such as [22] identify, on the one hand, the problems caused - at a numerical stability or algorithm convergence level- by the great variation in terms of numerical data representation (basically, of floating point). Others, such as [41], due to the fact that they are more concerned with theoretical contributions, reach higher complexity levels even in terms of efficiency estimation. From the point of view of computers local networks, these drawbacks might be considered not very likely. In the specific case of data representation, most of the local network interconnected computers use standard processors that, in turn, explicitly stick to the representation standards defined by IEEE, such as ANSI/IEEE 754-1984 [72] for floating point numbers.

3.2 Parallel Computing in Heterogeneous Clusters

The stated characteristics of local networks must be necessarily taken into account for the computing tasks parallelization solved over this processing hardware platform. Summarizing the previous section, the characteristics of a local network considered as a parallel machine are:

- Distributed memory computer, loosely coupled multicomputers.
- Ethernet processors interconnection network.
- Interconnection network low performance.
- Processing heterogeneity, translated into different relative speeds.

The following subsections present an explanation of the impact of these characteristics on tasks parallelization to be solved in installed computers local networks. It is advisable to remember that the parallelization (even though it is carried out over local networks or, what is the same, the parallel algorithms executable on the local networks) has direct impact on the achieved performance and, thus, the parallel algorithms that take into account the underlying computing architecture will be favored (such as it has been traditionally done within high performance computing environments). The numerical algorithms and those belonging to linear algebra do not escape from this principle in general, and in particular, they do not escape from matrix multiplication.

3.2.1 Loosely Coupled Multicomputer

The programming model normally imposed on loosely coupled multicomputers and, even on multicomputers in general, is that of message passing [3] [52] that, in turn, is derived from CSP (Communicating Sequential Processes) [66]. This message-passing model implies that the parallel program will be a set of sequential processes communicated and/or synchronized - where the synchronization in this context normally implies some direct or indirect way of communication among processes.

The previously mentioned hardware characteristic -with respect to a physically distributed memory- can be considered as having a direct relation to the message-passing model. Since the memory is physically distributed, it is very difficult to share memory among processes assigned to (being executed on) various processors without any relatively high penalty in terms of performance.

It has been traditionally assumed (in many cases, experimented) that, on the one hand, it is true that, as a last resort, any algorithm can be implemented on a loosely coupled distributed memory parallel machine with enough effort (normally, at a software intermediate layer level). But, on the other hand, it is also true that the likelihood of obtaining an acceptable or optimal performance in terms of performance is higher when the message-passing programming model is adopted in this hardware environment. Actually, the specific case of the large quantity of the proposed algorithms coexisting for the matrix multiplication does nothing but confirm this fact. In other words, given a shared-memory-based parallel algorithm as example, it is likely to design and implement an algorithm (that solves the same task) based on the message passing that achieves the same or better performance. A priori, it is even more likely that a message-passing-based algorithm will not be outperformed in performance by a shared-memory-based algorithm solving the same task.

The specific area of matrix operations in general, those arising from linear algebra in particular, and more specifically matrix multiplication, do not escape from the previously mentioned rule. Thus, in the specific parallel processing environment provided by the computers local networks, the algorithms to be considered (as best) will be those that could be expressed in terms of message passing without any type of intermediate adaptation or translation involving a processing overload on the same operations basic to be solved.

3.2.2 Ethernet Network for Processors Interconnection

As explained above, Ethernet networks have a great variation in terms of performance (10 or 100 Mb/s, for instance) and wiring, or what it can be considered as hardware topology (hubs and/or switches use, for instance). However, due to the same definition of standard for machines interconnection [73], in all the cases the message physical broadcast capability is kept. That is, independently of the performance and the wiring of Ethernet local networks, the same message can be sent from a computer and received from more than one computer of the same network.

At the level of protocols (or stack or series of protocols [112] [32]) used over Ethernet networks, the broadcast-type communications capability is kept at least in the protocols closer to the hardware such as IP, IGMP and UDP [112] [34] [96] [95]. The capability of carrying out physical broadcast is really important in Ethernet networks, at least for two reasons:

- From the scalability point of view: in relation to the use of the communication physical means (wiring), the physical broadcast is independent of the quantity of receivers. That is, the communications time tends to be the same independently of the quantity of computers involved in a data transference (they basically receive data). It cannot be assured that the total broadcast time is exactly the same for any quantity of receivers since the total time depends on other factors, such as:
    - Receivers synchronization method, so that the same datum is simultaneously received by everyone.
    - Data or method/s acknowledge recognition method in order to assure that the sent data have been received in all of the computers.
    - Loss index in terms of each computers data reception that depends, in turn, on the quality of each computers net board used, and the wiring.
    - Data reception and delivery buffers availability.

  However, one of these factors are considered to be relatively less important than the data transmission in itself (they imply the sum of the times one or more lower orders of magnitude in relation to the transferred data time).
- From the performance point of view: as usually accepted, the use of broadcast in parallel programs is really extended [124]. If broadcast messages among processes are not implemented by using Ethernet networks physical broadcast, they become multiple point-to-point transferences (communicating the same data). It is clear that every data communication among computers implies the use of wiring. If in the wiring there are no switches, each communication implies the use of all the communication environment (Figure 3.1 and Figure 3.2) and, thus, this communication environment has to be multiplexed between the different data transferences. In other words, in all the local networks where there is no use of communication switches, all the data transferences are sequentialized and, in fact, each point-to-point communication among processes engages all the communication environment. In this way, broadcast message communication times are multiplied by the receivers' quantity. On the other hand, even in the case in which switches are used in the local networks wiring, the possibility of assigning more than one process to a computer ends with the capability of carrying out simultaneous point-to-point communications. From another point of view, since Ethernet by definition has broadcast independently of the wiring, provided that the broadcast message implementation uses it, there will be no penalty in terms of the data transference time among computers.

On the other hand, and also from the performance point of view, the use of point-to-point messages between processes leads to an overall higher communication. The communication time between processes proportionally increases when: 1) there are no switches in the wiring, and 2) there are switches, but there is more than one process assigned to each computer and, thus, messages are multiplexed (are transmitted sequentially, one after the other) in the communications channel.

Thus, in the specific environment of parallel processing provided by the computer local networks, the algorithms to be considered (as best) will be those that can be expressed in terms of broadcast-type messages. These messages can be implemented directly into Ethernet networks as follows:

- The broadcast algorithms costs are avoided by using point-to-point messages.
- There is better scalability, at least in terms of communications among processes.
- Penalties are avoided by the sequential use of the communications environment in local networks without wiring with switches.

Taking as reference low cost parallel machine manufacture from interconnected computers with Ethernet, the cost (in the extreme case) could be reduced even more since the switches currently used (and that are considered as essential) could be replaced by hubs without penalties in terms of performance. However, this extreme case implies that all the programs to be executed are expressed in terms of broadcast messages.

3.2.3 Interconnection Network Low Performance

Standard computer local networks performance, and more specifically that of the installed Ethernet networks, is often considered as unacceptable for carrying out parallel computing [12]. In fact, there exist several proposals of low cost standard computer interconnection (PCs or less costly workstations) that:

- are interconnected with more costly network interfaces of higher performance [12] [91].
- are interconnected with more than one network interface of low cost [65] [48] [78].
- are interconnected with network interfaces specifically designed for parallel computation (TTL_/PAPERS: Purdue's Adapter for Parallel Execution and Rapid Synchronization [39] [40] [67] [38] [PAPERS]).

All these possibilities and their possible combinations can be found in [37], in the context of parallel processing using PCs with Linux operating system; but, in most of the cases, they can be applied to computers networks in general.

However, in all of these cases, the substantial cost increase is considered as a counterpart since:

- More hardware or more costly interconnection hardware is used (more network interfaces). The interconnection hardware with higher performance will always be more costly and, in many cases, this cost is multiplied when the hardware is not of massive use such as Ethernet networks. In the case of using more net boards, it is clear that the interconnection hardware cost is multiplied by the quantity of net boards added in each computer.
- Both the base software and the utilization tools become more complex. In many cases, the same operating system kernel must be modified and, in most of the cases, software intermediate layers must be added so that the applications are not affected or hardly

affected in terms of the communication capabilities utilization interface. All of these aggregates involve design and implementation cost (which is usually reduced by the use of free-use libraries) as well as maintenance cost, since any change in the network hardware, or in the operating system kernel, or in the libraries could affect directly the parallel applications using it.

- All the installed local networks are discarded automatically, all of them being in general of really low cost and relative performance: Ethernet of 10 Mb/s.

Taking into account these references, it is very likely that the decision of using the installed computers networks involves a review of the patterns and frequency of the parallel program communications to be executed. Otherwise, parallel programs are implicitly considered "good" enough so that the reduction of several orders of magnitude in terms of communications performance does not imply any minor change or changes in the overall performance.

The installed local networks communications low performance impact on the overall performance of the parallel programs to be executed always depends on the communications pattern and, more specifically, on the communications frequency among parallel program processes. At this point, the applications granularity and the parallel program to be executed become really important. Taking into account the data communication time model of any interconnection network -Equation (3.1)-, two aspects are be considered: latency and asymptotic bandwidth.

The communications local networks latency might be the aspect that should receive most of the attention from the performance point of view because of two reasons:

- As previously stated, this time is several orders of magnitude higher than in traditional parallel computers interconnection networks. Thus, the transferences data quantity between the processes of a parallel program should be several orders of magnitude higher. This evidently imposes a really strong restriction (as strong as higher orders of magnitude) on the parallelization carried out or the parallel programs executed over the workstations networks. This impact is or should be directly visible in the granularity, since that, assuming that the transferred data quantity is the same, increasing the messages size tends to produce less quantity of messages with less frequency and this, in turn, implies programs with higher granularity. In other words, the programs minimum granularity is restricted since the latency is very high and, thus, reducing the granularity implies a significant reduction in the performance.
- Also, as stated before, the communications latency tends to depend or can depend on the computers involved in the data transference. This means that it is not always true that the latency time for transferring data form a computer ws*i* to another ws*j* is equal to the latency time of the data transference from ws*i* to another ws*k*, $\forall$ *i*, *j*, *k*. In other words, it is quite likely that the latency time to be taken into account in the local area networks will be related to the higher latency time between all the interconnected computers pairs. This restriction is not independent of the previous one but complementary because the latency to be taken into account now is the highest of all the network that, in turn, will be equal to or higher than the defined by the Ethernet standard.

As regards the network asymptotic bandwidth low performance, it also affects significantly parallel programs performance. Once again, the strongest impact in terms of performance will fall into the parallel programs with finer granularity.

Assuming, for instance, that an Ethernet network of 10 Mb/s is used and that:

- the latency time is zero,
- data are transferred at the network maximum capability without any packing, protocols, software layers, etc. overhead,

data can be transferred at a 1.25 MB/s rate. To continue with the example, and considering the same context of the matrix multiplication operation, 1,990,000 floating-point operations are necessary (using the numer of operations given in the third equation of the previous chapter) in order to solve a matrix multiplication of 100×100 elements. Since it is rather common to find computers in a local network with a processing power of 300 Mflop/s or more, the approximate time for solving the matrix multiplication of 100×100 elements is less than 0.0067 seconds (6.7 ms). And the time for transferring a matrix of 100×100 elements represented in simple precision floating point over the network is of 0.032 or 32 ms! Thus, locally computing the result is $32/6.7 \cong 4.7$ times better than receiving the response from another computer connected by an Ethernet network of 10 Mb/s.

The previous example has several simplifications and several solution alternatives (such as using a network of 100 Mb/s), but it also gives an idea of the orders of magnitude of the times and risks of assuming that the parallel programs designed for parallel machines will not be penalized in terms of performance. Many times, this kind of analysis has lead to assume (prematurely) that it is not useful to solve in parallel the numerical problems in computers local networks. Once again, in order to avoid the communications network bandwidth impact on the performance, there is a tendency to increase granularity. But many times it is not possible (or it is not enough) to decrease the frequency and increase the messages size such as in the mentioned case about the solution of the message latency time problem. Actually, in the stated example, the latency time is considered to be zero.

The granularity increase related to the improvement of the parallel applications performance makes use of one of the typical characteristics of numerical problems. Normally, the numerical problems are solved with one or more operations on data structured in arrays (in general, multidimensional). It is also common to find that the processing involved is one or more orders of magnitude higher than the data quantity to be processed. In the case of matrix multiplication, it has already been explained that the data quantity over which the operation is carried out is $O(n^2)$, and the necessary number of operations is $O(n^3)$. Thus, as the amount of data is increased, it is relatively simple to increase the granularity, since a higher quantity of operations has to be carried out. This is the same as to consider in the previous example matrices of 100×100 elements with all the rest invariant:

- The computing time is now of approximately 6.7 s.
- The communications time for receiving the result matrix is of 3.32 s.

AAnyway, it should be bear in mind that increasing the messages size necessarily implies processing higher data quantity or increasing the minimum size of the problems or operations to be solved (what would be the same).

Summarizing, parallel programs with the best performance in local area networks will be those of higher granularity. In this sense, workstation networks do not differ from most (if not all) the other parallel computing platforms. But Ethernet interconnection networks imply an effort much more greater when increasing parallel programs minimum granularity, and this effort is proportional to the latency and bandwidth difference of Ethernet networks with traditional parallel computer processors interconnection networks.

3.2.4 Processing Heterogeneity

As previously explained, heterogeneity in terms of processing capability in a local area network is directly translated into the various relative speeds of the computers interconnected in local networks. Going a step further in the load balance problem- or load balance proportional to the relative speeds problem-, two aspects must be solved:

- Identification (as precise as possible) of computers relative speeds to be used.
- Processing workload distribution.

Relative Speed. Computers relative speed exact identification is, in general, not a simple task to carry out. Actually, it is one of the bigger problems that benchmark programs attempt to solve, and there is still much discussion about it. Luckily, as the specification of the numerical problems to be solved advances, the situation tends to be steadier or more predictable as regards the relative computing speed. In fact, in the numerical applications, and those of linear algebra in particular, the processing is highly regular and, thus, the relative speed computation among different computers is simpler as well. Within this scope, the steps to take are:

- Using a reduced size of the problem to be solved in order to execute it in all the computers and compute the differences in speed on the basis of each computer processing time. Assuming that, for instance, a FFT (Fast Fourier Transform) of 10^7, 10^8 or more data has to be computed, the computation of FFT with 10^5 or 10^6 data can be carried out in all the computers, and the relative differences directly proportional to the computing time of this "reduced" FFT can be assessed. The problem size has to be considered as cautiously as possible since erroneous results may arise [11] if the "reduced" problem data can be stored completely in cache memory in some computers and in others not. On the other hand, if large sets of data are taken, the execution time of this "mini-benchmark" will be really high.
- Using each computer speed for the matrix multiplication - as a reference in terms of computing speed. This would be the previous case for matrix multiplication. Even though - as stated in the previous chapter - matrix multiplication is particularly suitable for taking advantage of all the code optimizations, it is foreseeable that the differences in the processing type imposed by each operation or numerical problem will have similar performance consequences in the used computers processors. In the case of the previous problem, it would imply taking as reference (in terms of processing speed), for instance, a 1000×1000-element matrix multiplication execution time in order to make the necessary processing distribution to compute the FFFT of 10^7, 10^8 or more data. In this way, it is assumed that, if a computer wsi is two times faster than another one, ws_j, for carrying out a matrix multiplication, this difference of processing capability will be also kept for carrying out the necessary computation for a FFT. In this sense, the cost of executing a "reduced" problem for each application is avoided, though some

imprecision is added derived from the processing difference that may arise between a matrix multiplication and any other numerical problem solved in a computer network.

In both cases, it is clear that there exists an associated cost with respect to the relative speed computation itself that is not present in the homogeneous parallel machines in terms of the processing elements computing capability. For the specific case of matrix multiplication, the two previous steps are equal. Extending the scope of applications to L3 BLAS, the second option is highly suitable since the basic operation of all the defined ones in L3 BLAS is actually the matrix multiplication in itself. In the case of extending even more the scope of the operations arising from linear algebra (to, for instance, the operations defined in LAPACK), the matrix multiplication computing type is still very similar and, in fact, the operations that are relevant from the performance point of view (and the necessary computing time) are still those of L3 BLAS.

In the scope of computers with heterogeneous processing elements, the computing capability of each computer or processor tends to be used for computing the parallel machine heterogeneity [125]. In fact, there is a general tendency to resort to some way of "relative computing power of each computer wsi" or $pw(ws_i)$. Within this context, it is generally useful to resort to the computing power of wsi expressed in millions of floating point operations per second or Mflop/s(ws_i) and use it for the computation of $pw(ws_i)$.

$$pw(ws_i) = \frac{Mflop/s(ws_i)}{\underset{j=0..P-1}{máx}\ (Mflop/s(ws_j))} \tag{3.2}$$

where P computers are considered identified as ws_0, ..., ws_{P-1} and $pw(ws_i)$ is computed in function of the fastest one.

Load Distribution. Once the method to compute relative speeds and the specific relative speeds between local network computers to be used is defined, each computer processing load must be distributed.

Based on the idea of the relative speed computation of each computer given in Equation (3.3), a step further is taken to obtain the "normalized relative computing power of ws_i" or directly pw_i,

$$pw_i = \frac{Mflop/s(ws_i)}{\sum_{j=0}^{P-1} (Mflop/s(ws_j))} \tag{3.3}$$

so that

$$\sum_{j=0}^{P-1} (pw_j) = 1 \tag{3.4}$$

And with this metrics or index, each computer ws_i must carry out pw_i (or $pw_i \times 100$ %, in percentage), out of the total work to be performed.

In the specific case of matrix multiplication, this means that the workstation ws_i carries out all the necessary computations for $pw_i \times n^2$ of the result matrix C elements, which is the same as to establishing that ws_i executes $pw_i \times (2n^3 - n^2)$ operations. In the more general case of linear algebra applications and, even more general, of the numerical problems, the identification of the task to be solved in each computer wsi might not be so clearly definable. However, as a minimum of all the problems for which parallel methods have already been designed and implemented, some kind of division of the total work has been evidently done (although these parts have normally been *equal*).

3.3 Parallelization Guidelines for Clusters

Having already described clusters in general, and heterogeneous clusters in particular from the point of view of parallel processing, the parallelization guidelines that should be followed in order to obtain optimized parallel performance are:

- Load balance given by the data distribution, which, in turn, is carried out according to the relative computing capacity of each computer.
- Only broadcast type communications, so that the Ethernet networks' facility is used to the maximum.
- Unidimensional data distribution, following almost uniquely the very hardware of the physical interconnection defined by the Ethernet standard. In addition, this data distribution eases the use of broadcast messages because, for instance, there are no defined rows and columns of processors, like in the bidimensional static processors interconnection networks.

These guidelines could be said to be specifically related to heterogeneous clusters. In the specific case of matrix multiplication, none of the algorithms described in the previous chapter follows or fulfils these parallelization guidelines. In fact, Fox's algorithm has been specifically modified so that broadcast messages can be performed in an optimized way in processors interconnection bidimensional networks.

However, other parallelization guidelines can be applied; those derived from the area of linear algebra applications and numerical processing in general, such as

- The adoption of the SPMD processing model (Single Program - Multiple Data), which significantly simplifies programming, debugging and optimizing applications of parallel computing.
- The advantage of using the facility (if there is any) of overlapping communications with local computing on each computer. Even though in the specific cases of local networks this facility is not assured, it is possible to at least attempt reducing the latency time by carrying out communications in background with respect to the local computing in each machine.

3.4 Parallel Matrix Multiplication

Although some ideas about matrix multiplication on heterogeneous workstations networks have already been presented, it is necessary to move a step further in the level of detail and precision of the method by means of which $C = A \times B$ is computed in parallel. This method has already been explained briefly in [117].

In a similar way to the traditional presentation of numerical algorithms oriented to distributed memory parallel computers, both the data distribution and the sequence of steps to be executed by each computer (processor) are explained so as to obtain the expected result. As in all the algorithms oriented to distributed memory parallel computers, the likelihood of data replication is discarded in all the processors (in this case, local network computers) almost from the beginning, since this imposes a really high memory requirement in each of them.

3.4.1 Data Distribution

Assuming that each of the P computers using ws_i ($0 \leq i \leq P$-1) has its "normalized relative computing power", pw_i, computed and fulfils Equation (3.3), one of the ways to balance the processing load is to make each workstation ws_i compute pw_i out of the total data of matrix C. In the context of computing parallelization over matrices, establishing that certain part of a matrix X, or a submatrix $X^{(i)}$, is computed in a computer wsi is the same as to assign $X^{(i)}$ to ws_i. That is, the submatrix data computed in a computer reside locally in that computer.

Figure 3.4 schematically shows the simplest ways to determine the matrix C data to be computed in a computer,

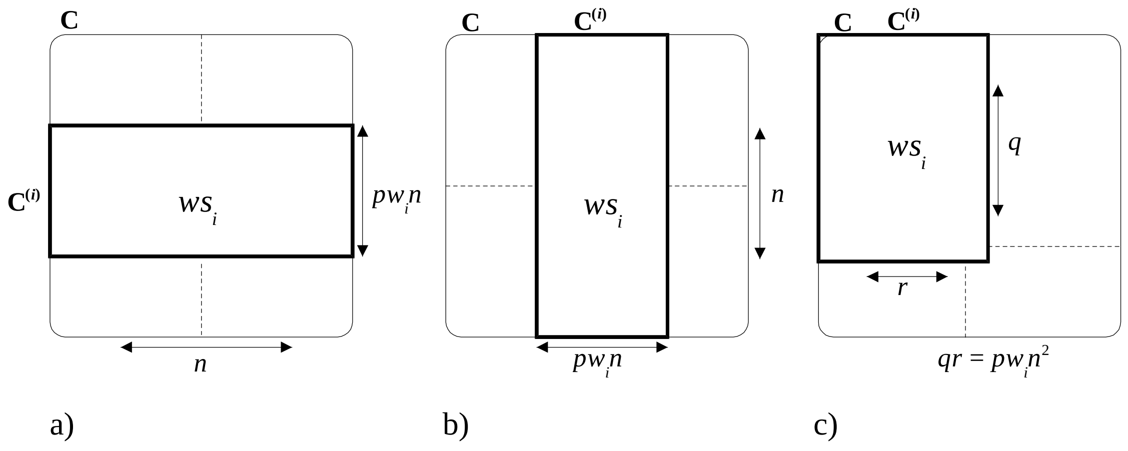


Figure 3.4: Simple Ways of Data Assignments.

where:

- $C^{(i)}$ is C's submatrix assigned to computer ws_i or, equally, the part of C that resides and is computed in ws_i.

- Figure 3.4-a) shows the assignment by rows ($pw_i \times n$ rows), i.e. that ws_i computes a part of C, $C^{(i)}$, of $fC_i = pw_i \times n$ rows by n columns. This way of assigning matrix data is called row block striped partitioning in [79].
- Figure 3.4-b) shows the assignment by columns ($pw_i \times n$ columns), i.e. that ws_i computes a part of C, $C^{(i)}$, of n rows by $cC_i = pw_i \times n$ columns. This way of assigning matrix data is called column block striped partitioning in [79].
- Figure 3.4-c) shows the assignation by "bocks in general," i.e. that ws_i computes a part of C, $C^{(i)}$, of $fC_i = q$ rows by $cC_i = r$ columns so that $qr = pw_i n^2$. This way of assigning matrix data can be directly related to the so-called block-checkerboard partitioning in [79].

From the ways of data assignation shown in Figure 3.4, notice that:

- Row assignment is completely equivalent to column assignment.
- Assignment by "blocks in general" is rather more complex than the previous ones and it does not seem to contribute with any benefit and it can thus be discarded.

Although there exist more complex and more appropriate ways of distributing data for parallel computing in the matrix multiplication parallel computing algorithm, result matrix C is distributed by rows. The relation and the implications of carrying out this partitioning, comparing it with the more complex ones, will be explained after the algorithm is thoroughly presented. In short, the computer ws_i "is in charge of" (has the data of and computes) submatrix $C^{(i)}$ that is of $fC_i = pw_i \times n$ rows and n columns, i.e. $C^{(i)}_{fC_i \times n}$.

Once result matrix C distribution is defined in the network computers, the way of distributing data of matrices A and B is defined in function of the local computations to be carried out in each computer. Since it has been established that ws_i is to compute $C^{(i)}_{fC_i \times n}$, it must carry out the multiplication

$$C^{(i)}_{fC_i \times n} = A^{(i)}_{fA_i \times n} \times B \quad (3.5)$$

Figure 3.5 shows in each matrix (shadowed) the data involved in the computation of $C^{(i)}$ according to Equation (3.5). Thus, it is quite clear that computer ws_i should locally have the data of submatrix A, $A^{(i)}$, of Figure 3.5, and this means that matrix A's distribution among P computers ws_0, ..., ws_{P-1} is equal to that of matrix C.

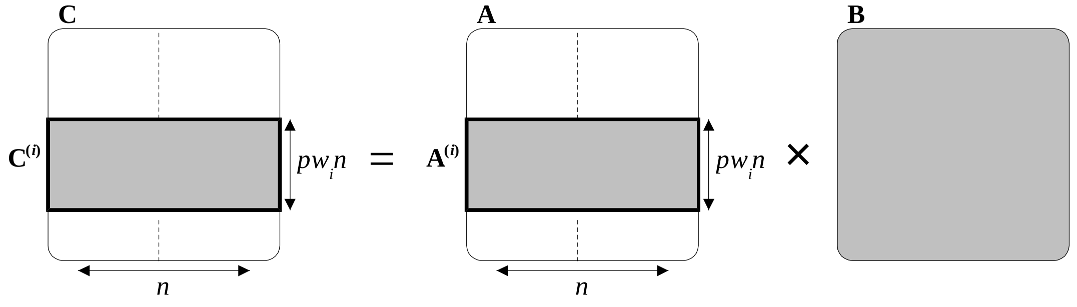


Figure 3.5: A submatrix Computation.

In short, computer ws_i "is in charge of" (has the data of) submatrix $A^{(i)}$ that is of $fA_i = fC_i = pw_i \times n$ rows and n columns, i.e. $A^{(i)}_{fA_i \times n}$.

According to Equation (3.5) and Figure 3.5, the optimum in terms of data availability in ws_i for the computation of $C^{(i)}$ would be to have all matrix B available (in local memory). With this, the simultaneous computation of all $C^{(i)}$ could be carried out, because all the computers would have all the necessary for carrying it out. As previously explained, all matrices data should be distributed among different computers so as not to establish too big memory requirements and, thus, matrix B should also be distributed among computers. According to Equation (3.5) and Figure 3.5, all the computers require the complete matrix B, thus this matrix is distributed in equal parts among all the computers ws_0, ..., ws_{P-1}. During the parallel program execution, the computers should communicate with each other in order to exchange-receive the parts of matrix B that they need for the result matrix portion computation. Once more, this distribution can be carried out similarly by rows or by columns. Taking into account that all the computers will compute $C^{(i)}$ in function of this distribution, the possibility of finding a computer with any advantage over another should be analyzed (at least, approximately).

B's Distribution by rows. If matrix B is distributed in equal parts by rows among computers, each submatrix $B^{(i)}$ would be of n/P rows by n columns, and all the matrices data would be distributed such as Figure 3.6 shows for computer ws_i.

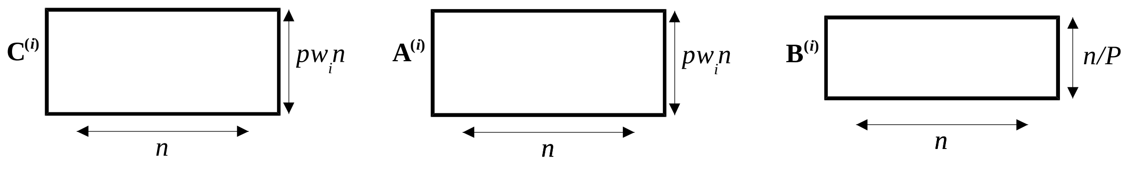


Figure 3.6: Submatrices in ws_i with B Distributed by Rows.

In this way, the partial computation of submatrix $C^{(i)}$ (denoted as $C^{(i,j)}$) - that each computer can carry out with the data locally available- is given by the matrix multiplication such as Figure 3.7 shows.

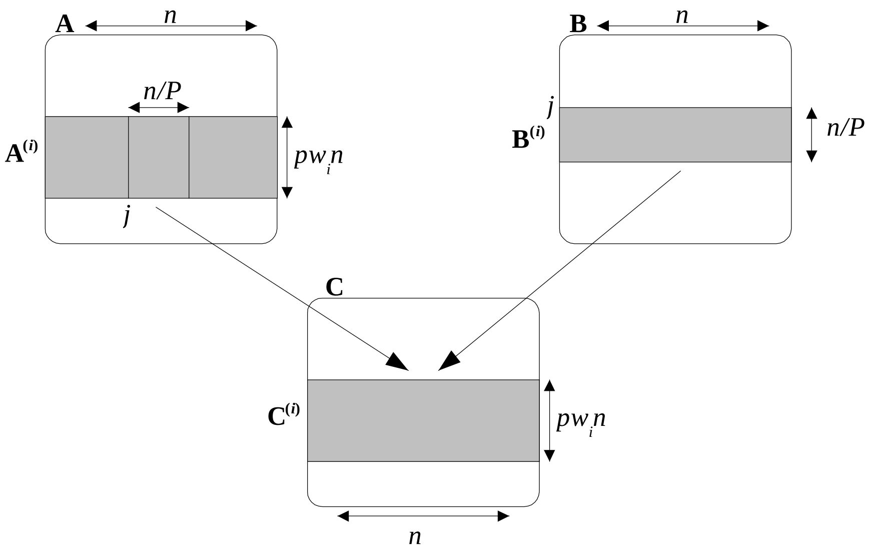


Figure 3.7: Partial Computation of $C^{(i)}$ with B Distributed by Rows.

Similarly, with $B^{(k)}$ data in each computer ws_k, the partial calculation with a $A^{(i)}$ part (of pw_in rows and n/P columns) -such as Figure 3.7 shows -will be added with $B^{(k)}$ (of n/P rows and n columns) to $C^{(i_i)}$. In this way, with successive partial computations of $C^{(i)}$, the final result is achieved. This means that each partial computation of $C^{(i)}$ involves a part of $A^{(i)}$, a complete block of $B^{(k)}$, and all $C^{(i)}$, i.e., the multiplication of two matrices: one of $pw_in \times n/P$ and another of $n/P \times n$.

B's Distribution by columns. If matrix B is distributed in equal parts by columns between the computers, each submatrix $B^{(i)}$ would be of n rows per n/P columns, and all the matrices data would be distributed as Figure 3.8 shows.

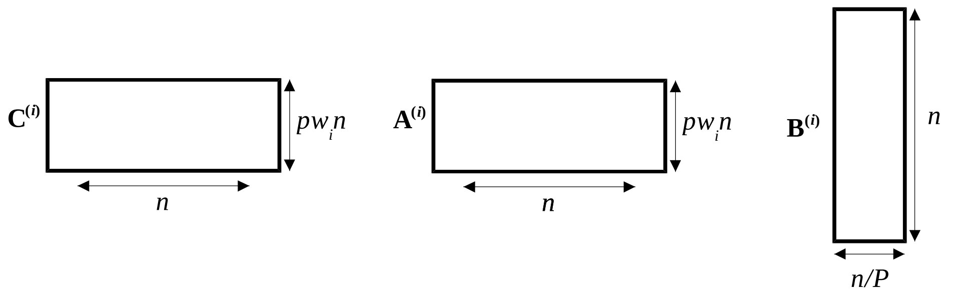


Figure 3.8: Submatrices in ws_i with B Distributed by Columns.

In this way, submatrix $C^{(i)}$ partial computation (denoted as $C^{(i_i)}$), that each computer can carry out with the data locally available, is given by the matrix multiplication as Figure 3.9 shows. In that case, the submatrix $C^{(i)}$ partial computation that can be carried out with $B^{(k)}$ data of each computer ws_k is that of a $C^{(i)}$ column block, denoted as $C^{(i_k)}$. In this case, the multiplication to be carried out is that of the whole matrix A, that is of $pw_in\times$n with a complete block of B, that is of $n\times n/P$, and a part of $C^{(i)}$ of $pw_in\times n/P$ is computed.

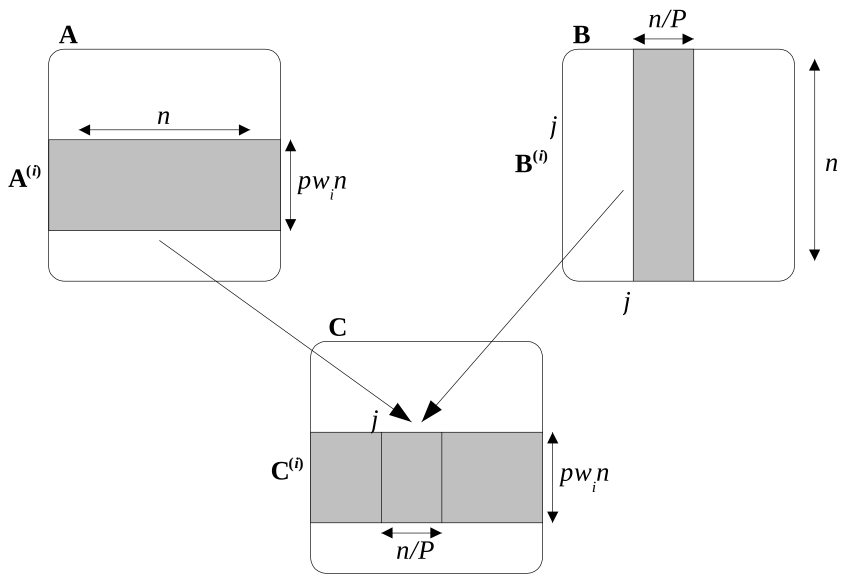


Figure 3.9: $C^{(i)}$ Partial Computation with B Distributed by Columns.

It is clear that both distribution alternatives of B data do not alter the number of operations to be carried out, even though the distribution by columns seems to be simpler in principle (or rather more intuitive with respect to the matrix multiplication definition itself); thus, this distribution will be chosen for the algorithm.

Summary and Comments on Data Distribution. The main characteristics of data distribution among computers are:

- Matrix A and matrix C are distributed by rows, with the number of rows given by its relative computing power. That is, computer ws_i with pw_i normalized relative computing power will have a submatrix of A, $A^{(i)}$, and a submatrix C, $C^{(i)}$, of $fA_i = pw_i n$ rows per n columns.
- Matrix B is distributed uniformly by columns, i.e. that all the computers will have the same number of columns (and, thus, of data) of B. That is, computer ws_i will have a submatrix of B, $B^{(i)}$, of n rows per $cB_i = n/P$ columns, where P is the total number of computers.

Even though

$$\sum_{j=0}^{P-1} (pw_j) = 1 \quad \text{and} \quad \sum_{j=0}^{P-1} (n/P) = n$$

is fulfilled, $fA_i = pw_i n$ will not necessarily be a integer, since $0 < pw_i < 1$, and thus, operating with integers, it is likely that:

$$df = \lfloor fA_1 \rfloor + ... + \lfloor fA_P \rfloor < n$$

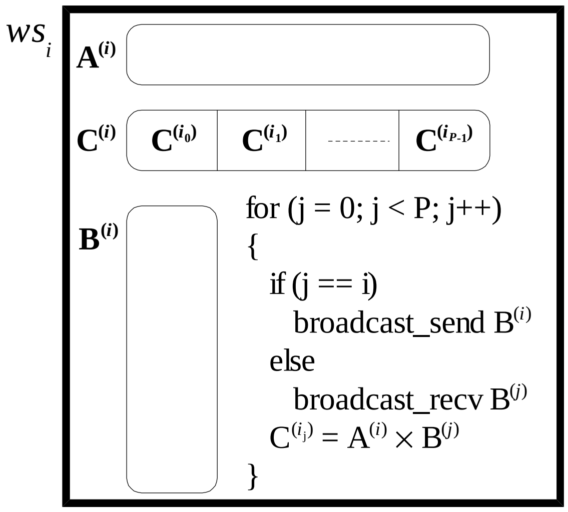
where $\lfloor fA_i \rfloor$ is the highest integer so that $\lfloor fA_i \rfloor < fA_i$.

The rows "left" undistributed, n-df, are uniformly distributed between the computers ws_0, ..., ws_{n-df-1}. Since that normally $P << n$, this "added" row can be considered irrelevant with respect to the likelihood of generating a computing load unbalance. The principle to follow in relation to the number of columns of B (n/P is not necessarily an integer) is similar.

3.4.2 Computation

As expected for most (or every) numerical algorithms solved in parallel, the computation is designed under the SPMD model (Single Program - Multiple Data) and is directly given in pseudocode in Figure 3.10 for computer ws_i, where

- P is the total number of computers.
- Matrices A and C are distributed by rows, and $A^{(i)}$ and $C^{(i)}$ are the submatrices that ws_i has locally.
- Matrix B is distributed by columns, and $B^{(i)}$ is the submatrix that ws_i has locally.



```
for (j = 0; j < P; j++)
{
  if (j == i)
    broadcast_send $B^{(i)}$
  else
    broadcast_recv $B^{(j)}$
  $C^{(i_j)} = A^{(i)} \times B^{(j)}$
}
```

Figure 3.10: Multiplication Pseudocode in ws_i.

Figure 3.11 schematically shows the sequence of the algorithm running steps for four computers.

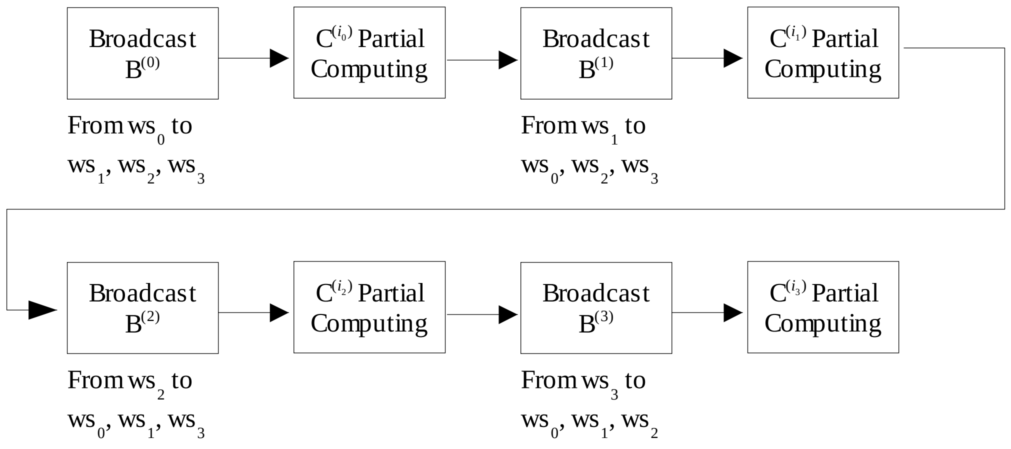

Broadcast $B^{(0)}$
From ws_0 to ws_1, ws_2, ws_3
$C^{(i_0)}$ Partial Computing
Broadcast $B^{(1)}$
From ws_1 to ws_0, ws_2, ws_3
$C^{(i_1)}$ Partial Computing
Broadcast $B^{(2)}$
From ws_2 to ws_0, ws_1, ws_3
$C^{(i_2)}$ Partial Computing
Broadcast $B^{(3)}$
From ws_3 to ws_0, ws_1, ws_2
$C^{(i_3)}$ Partial Computing

Figure 3.11: Running Steps Sequence for Four Computers.

Both from Figure 3.10 pseudocode and the sequence of steps shown in Figure 3.11, it can be noticed that the most important characteristics of the algorithm are:

- It has P broadcast communication steps (broadcast messages), since each computer sends a broadcast message and receives P-1 broadcast messages.
- It has P local computing steps, since all the computers carry out P partial computations (k = 0, 1, ..., P-1) from the part of matrix C locally stored.
- The total number of running sequential steps is $2P$, and the total computing time is roughly computed as Equation (3.6) shows

$$t_{par} = P\,(t_{bcast} + t_{comp}) \tag{3.6}$$

where

- t_{bcast} is the necessary time to carry out a broadcast operation of a submatrix $B^{(i)}$, bearing in mind that $|B^{(i)}| \cong |B^{(j)}|$ $0 \leq i, j \leq P$-1, where |X| denotes the number of elements of matrix X.
- t_{comp} is the necessary time to execute the local computation in all the computers, which is the same, since the load is balanced in function of A and C data stored in each computer.

In the context of local networks interconnected by Ethernet networks, the communication time to be used in Eq. (3.6), t_{bcast}, can be computed by directly using Eq. (3.4), for which it is necessary to know the communications network latency and asymptotic bandwidth times. However, this is not possible for any parallel computer and, in many cases (even in Ethernet interconnected local networks) broadcast time can depend on the way the communications libraries used for message passing have been implemented. In fact, it is rather common to find that the time of a broadcast with *k* receptors is *k* times longer than that of a point-to-point communication, since broadcast messages are implemented as multiple (*k*) point-to-point messages. In any case, notice that both latency and asymptotic bandwidth times of the communications network should be computed from/in the user's processes making up a parallel program [115] [117].

Thus, considering that
- Broadcast messages are implemented by taking advantage the Ethernet networks broadcast capability.
- The message latency time is known among the processes of a parallel program and is denoted as α.
- The message asymptotic bandwidth is known between the processes of a parallel program and is denoted as $1/\beta$ (expressed in terms of the processed elements type, such as double precision floating point numbers: 8 bytes).
- The amount of B data that each computer has, $|B^{(i)}|$, is given by *n* rows by $cB_i = n/P$ columns, i.e. n^2/P elements.

$$t_{bcast} = \alpha + \beta n^2/P \tag{3.7}$$

In order to compute the local processing time to be used in Eq. (3.6), t_{comp}, the computing power value of each computer used can be reused in order to compute the normalized relative speeds. According to Eq. (3.3), for the computation of pw_i, the computing power of ws_i should already be in Mflop/s, i.e. Mflop/s(ws_i). These Mflop/s values can be used in two ways for the local time computation:

1. Taking into account the computing capability of a single computer: Mflop/s(ws_i).
2. Taking into account the computing capability of the complete parallel machine (with the sum of all the computers): $\Sigma_{i=0}^{P-1}(Mflop/s(ws_i))$.

The first case, taking into account the computing capability of a single computer, uses up the fact that all the computers have to execute the same number of floating point operations, and thus the running time is the same in all of them. The second case, taking into account the computing capability of the complete parallel machine, uses up the facts that:
- the complete parallel machine computing power is known,

- the total number of operations to be carried out ($2n^3$-n^2) is known,
- the total number of operations is divided in *P* steps, thus 1/*P* out of the total is carried out in each step.

Thus, the running time t_{comp} of each processing step can be estimated independently of the computers heterogeneity with

$$t_{comp} = \frac{2n^3 - n^2}{P\ pw} \tag{3.8}$$

where *pw* is the parallel machine computing power

$$pw = \sum_{i=0}^{P-1} Mflop/s(ws_i) \tag{3.9}$$

In this way, using Eq. (3.8) and Eq. (3.9) in Eq. (3.6), the total computing time of the algorithm is given by:

$$t_{par} = P\alpha + \beta n^2 + \frac{2n^3 - n^2}{pw} \tag{3.10}$$

That is, in the total time, *P* latency times corresponding to *P* broadcast messages are accounted for, plus all matrix B data communications time at the maximum transference speed, plus the matrix multiplication computing time considering the parallel machine computing power (which is the sum of the used machines computing powers).

3.4.3 Overlapping Computation and Communication

The above presented algorithm does not take into account the likelihood of overlapping computation with communication since this is not considered in the specification itself (pseudocode in Figure 3.10). As previously stated, the likelihood of overlapping computation with communication in traditional parallel computers has been a constant and was associated many times with the interconnection networks design in itself. In the case of the standard computers found in the local networks, this capability cannot be necessarily assured a priori. However, adapting the previous algorithm in order to be able to overlap communications with the local computation in each computers has many advantages:

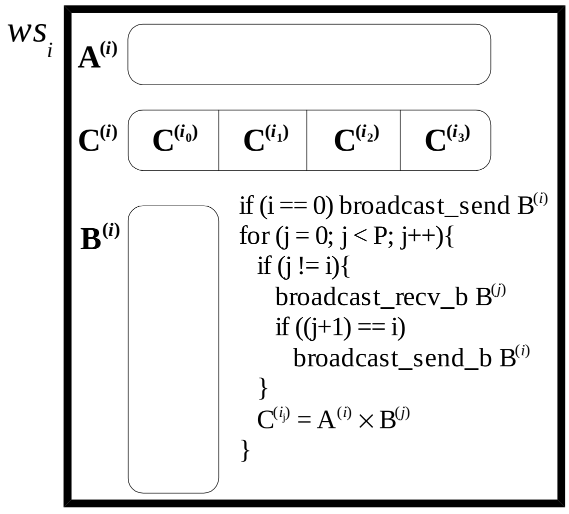
- Although it depends on the adaptation to be made (added overload), the performance is not lost in general in relation to the presented algorithm, since the worst that could happen is that the communications may become sequential with respect to the computation,
- There is a tendency to use up the likelihood of making computation overlapped with communications - in the cases in which this is possible in one or more computers. In general, the performance can be upgraded even though some computers may be only able to receive or send data in an overlapped fashion with the local computation.
- In the best of the cases, if the application granularity is big enough, and if all the

computers are capable of overlapping communications with local computation, a near-to-optimum performance value can be achieved.

- The computing algorithm overlapped with communications can identify each machine capability of overlapping (or lack of it) and, in this way, assess each machine separately and the complete machine (as a class of benchmark for the assessment of overlapping computing capability with communications).

The adaptation of the algorithm presented in Figure 3.10 -in order to take advantage of the computation overlapped with communications- is given in Figure 3.12, where

- P is the total quantity of computers.
- The matrix data distribution remains invariant with respect to the previous algorithm.
- The functions "recv broadcast_s" and "send broadcast_s" receive and send broadcast data in an overlapped fashion whenever possible (or in background from the operating system point of view), while partial results of $C^{(i_j)}$ are computed.
- It can be noticed that the first broadcast message is not carried out in an overlapped fashion since, initially, only ws_0 has the data of $B^{(0)}$, and for the first computing step, all the computers need $B^{(0)}$.
- The pseudocode on each computer turns a little bit more complex, but not too much in relation to that of Figure 3.10.



```
if (i == 0) broadcast_send B^(i)
for (j = 0; j < P; j++){
  if (j != i){
    broadcast_recv_b B^(j)
    if ((j+1) == i)
      broadcast_send_b B^(i)
  }
  $C^{(i_j)} = A^{(i)} \times B^{(j)}$
}
```

Figure 3.12: Pseudocode of Overlapped Computation with Multiplication in ws_i.

Figure 3.13 schematically shows the running steps sequence of the algorithm for four computers. Both from the pseudocode of Figure 3.12 and from the steps sequence shown in Figure 3.13, it can be noticed that the most important characteristics of the algorithm are:

- It has *P* broadcast communication steps (broadcast messages), since that each computer sends a broadcast message and receives *P*-1 broadcast messages. Except for the first broadcast, the rest of *P*-1 can be carried out (depending on the computers capabilities) in an overlapped fashion with the local computation.
- It has *P* local computation steps, since all the computers carry out *P* partial computations ($C^{(i_k)}$, $\forall\ k = 0, 1, ..., P-1$) of the part of matrix C locally stored.
- The total number of sequential steps is *P*+1.

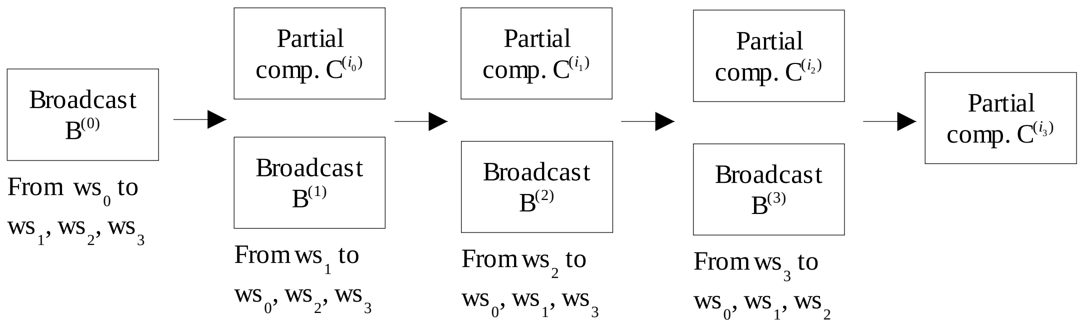

Broadcast $B^{(0)}$
From ws_0 to ws_1, ws_2, ws_3
Partial comp. $C^{(i_0)}$
Broadcast $B^{(1)}$
From ws_1 to ws_0, ws_2, ws_3
Partial comp. $C^{(i_1)}$
Broadcast $B^{(2)}$
From ws_2 to ws_0, ws_1, ws_3
Partial comp. $C^{(i_2)}$
Broadcast $B^{(3)}$
From ws_3 to ws_0, ws_1, ws_2
Partial comp. $C^{(i_3)}$

Figure 3.13: Overlapped Computation with Communication in Four Computers.

Now, the total computing time depends on several factors, namely:

- Computers capability of overlapping computation with communications. This will determine whether the communications are carried out "simultaneously" with the computation (as Figure 3.13 shows schematically) or not. If a computer is not capable of overlapping computation with communications, the computing and communications tasks will have to be sequentialized.
- The impact of the "background" communication task on the computation. Due to the fact that data are being transferred, there will be one or more processes controlling this transference and, thus, using CPU and resources such as the memory hierarchy. Consequently, the computing process/es will now compete for these resources.
- The problem granularity will determine the percentage of the communications to be overlapped with the computation. In other words, even though a computer is capable of overlapping computation with communications, if the time of communicating $B^{(i)}$ is greater than that of the local computation, the communications time will necessarily determine the total running time.
- Assuming the best of the cases, i.e. that:
    - o all the computers are capable of overlapping computation with communications;
    - o and there are no penalties in the computing capabilities due to carrying out overlapped data transferences,

  the total running time of the algorithm of Figure 3.12, can be estimated with Eq. (3.11),

$$t_{par} = t_{bcast} + (P\text{-}1)\ max(t_{bcast}, t_{cómp}) + t_{cómp} \qquad (3.11)$$

where t_{bcast} y t_{comp} are computed according to Eq. (3.7) and with Eq. (3.8), respectively.

3.4.4 Reduction of Memory Requirement for Messages

Both the initially presented algorithm in Figure 3.10, with the computing and communication periods sequentially run, and that of Figure 3.12, organized to take advantage of the possibility of overlapping local computing with communications, have a common characteristic in terms of memory for messages: both communicate the whole submatrix of B. This means that buffers or memory are eventually necessary in each computer so that ws_i receives, at least, the data of matrix B, $B^{(j)}$, of the computer ws_j

without destroying or overwriting local data.

In the specific case of the algorithm with overlapped computing and communications, this not only generates higher memory requirements, but also makes the initial time of sending the first data to all computers even higher (Figure 3.12, first communications step: delivery of $B^{(0)}$), as matrix B becomes greater and, thus, the submatrix involved in communications becomes greater as well.

The reduction of memory requirements for messages is relatively simple and is based on the idea of processing by blocks: instead of sending (and receiving) the entire local submatrix B of each computer, it is sent (received) in parts. If, for instance, each submatrix of B, $B^{(i)}$, is sent in ten parts, each of these parts will be of $|B^{(i)}|/10$ elements, and thus memory requirements for communications are immediately reduced to 1/10. In fact, the basic "communications block" defined for the presented algorithms is exactly $n/10$, where n is the order of square matrices to be multiplied.

It should be remembered that matrix $B^{(n\times n)}$ is distributed by column blocks among computers, i.e. having P computers: each of them will have a submatrix of B with n rows and n/P columns. Submatrices are sent-received in "blocks" of $n/10$ rows (in order to reduce to 1/10 the memory requirements for communications) and n/P columns (all of the columns).

In the specific case of the algorithm with overlapped computing and communications, this also implies reducing the initial time of sending the first data to all the computers (Figure 3.12, first communication step: delivery of $B^{(0)}$ since now we have to communicate only the 1/10 of the whole data of submatrix in order to start performing the partial computations. More specifically, the very delivery of $B^{(0)}$ is overlapped with the partial computations in which submatrix $B^{(0)}$ is involved.

3.4.5 General Characteristics

Beyond the differences in terms of the likelihood of overlapping computation with communications, the most important characteristics of both ways of computing a matrix multiplication $C = A\times B$ in P computers in parallel are that:

- They follow the SPMD model (Single Program - Multiple Data).
- There is no data replication since all the data are distributed and a single datum remains locally in a single computer;
- Communications among computers are only of broadcast type, and are carried out so that they do not generate interferences in the communications network independently of the wiring used (hubs, switches, etc.).
- There is a tendency to the maximum granularity, all the data to be communicated from one computer to another are transferred only once. This tends to reduce the penalty given by the messages latency time (startup).
- The load balance in terms of computation is given by the quantity of result matrix data to be computed by each computer that, in turn, is defined proportionally to its relative computing capability with respect to the other computers computing in parallel. Thus, with the distribution of the result matrix C data, all the computers carry out the

processing in nearly the same time.

- The communications balance is given by the assignation of matrix B data, which are the only transferred between computers. Since every computer counts with approximately the same data quantity of matrix B, and all the computers send a broadcast message and receive *P*-1 broadcast messages, all the computers send and receive approximately the same amount of data.

3.4.6 Other Ways of Distributing Data

Such as previously stated in the data distribution explanation for the algorithm, there exist more complex ways of distributing matrix data among multiple processors of a parallel machine than the one presented and used in the present chapter. In fact, there are some discussions with this respect, such as the technical reports [47] [106]. One of the most appropriate ways is the so-called block-cyclic partitioning in [79], even though it is more commonly known as two-dimensional block cyclic decomposition, such as mentioned in [Cho] [21] [45] and within the context of the ScaLAPACK library (Scalable LAPACK) [27] [28] [ScaLAPACK]. Figure 3.14 shows this way of distributing data of a matrix A of 7×8 elements, considering 1×1 elements blocks, in a 3×2 processors mesh.

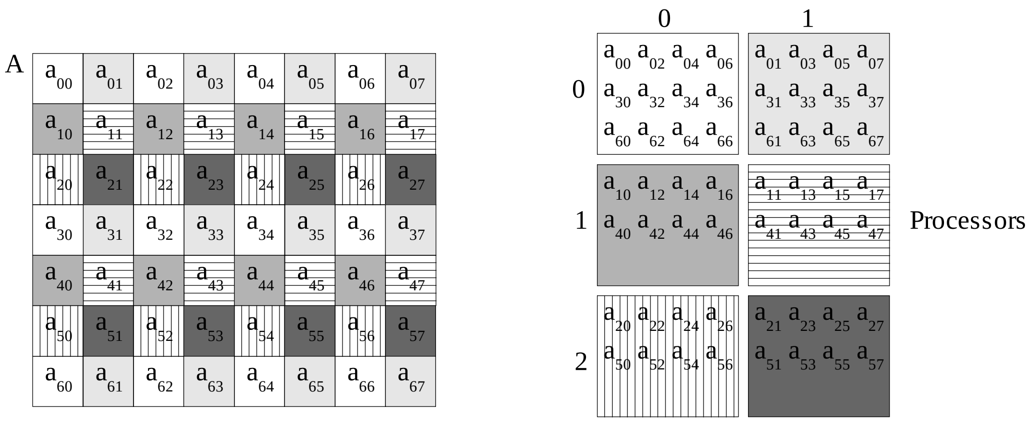

A
a_{00} a_{01} a_{02} a_{03} a_{04} a_{05} a_{06} a_{07}
a_{10} a_{11} a_{12} a_{13} a_{14} a_{15} a_{16} a_{17}
a_{20} a_{21} a_{22} a_{23} a_{24} a_{25} a_{26} a_{27}
a_{30} a_{31} a_{32} a_{33} a_{34} a_{35} a_{36} a_{37}
a_{40} a_{41} a_{42} a_{43} a_{44} a_{45} a_{46} a_{47}
a_{50} a_{51} a_{52} a_{53} a_{54} a_{55} a_{56} a_{57}
a_{60} a_{61} a_{62} a_{63} a_{64} a_{65} a_{66} a_{67}
0
1
0
a_{00} a_{02} a_{04} a_{06} a_{30} a_{32} a_{34} a_{36} a_{60} a_{62} a_{64} a_{66}
a_{01} a_{03} a_{05} a_{07} a_{31} a_{33} a_{35} a_{37} a_{61} a_{63} a_{65} a_{67}
1
a_{10} a_{12} a_{14} a_{16} a_{40} a_{42} a_{44} a_{46}
a_{11} a_{13} a_{15} a_{17} a_{41} a_{43} a_{45} a_{47}
Processors
2
a_{20} a_{22} a_{24} a_{26} a_{50} a_{52} a_{54} a_{56}
a_{21} a_{23} a_{25} a_{27} a_{51} a_{53} a_{55} a_{57}

Figure 3.14: Two-dimensional Block Cyclic Decomposition.

As Figure 3.14 shows, the two-dimensional block cyclic decomposition is clearly oriented to processors interconnection networks with grid or two-dimensional torus topology. It is also applicable for any size of matrix blocks, any size of matrices, and any processor mesh or two-dimensional torus.

All the ScaLAPACK library assumes that the data (matrices and vectors) are distributed according to this decomposition. Actually, since there exist several alternatives for this distribution, it is represented with a distribution array descriptor, in which the following is identified:

- Number of matrix rows.
- Number of matrix columns.
- Number of block rows.

- Number of block columns.

This array descriptor is used (as a parameter) to solve computing basic routines such as matrix multiplications. The main advantage explicitly identified with this distribution is that of load balance.

In the context of parallel computation in workstations networks, three considerations must be taken in relation to the comparison of the Two-dimensional Block Cyclic Decomposition with the presented in this chapter:

- Size of the blocks to be distributed.
- Data distribution and their relation to the processors interconnection.
- The algorithm load balance.

Size of the Blocks to be Distributed. This is an important drawback for libraries like ScaLAPACK and also for PLAPACK [4] [119] [PLAPACK], which carry out the parallel computation tasks with the same block size (given in ScaLAPACK by the data distribution array descriptor). The blocks size directly determines two aspects in the parallel processing:

- Local computation in each processor.
- Communication among processors.

As regards local computation, the block size is defined in order to obtain the optimal performance of the involved processors. This is relatively simple in the case of parallel machines with homogeneous processors, but it represents an unavoidable problem for the computers with heterogeneous networks. Within this context, it is even possible that two processors may have the same processing capability (relative speed or Mflop/s), but with different block values, since this highly depends on the memory hierarchies (cache memory levels and sizes) which can be very different.

Thus, having a same block size is another problem to be solved, since the best value should be determined, all of which is not generally a simple task given the large variety of computers in the local networks. Being the chosen block of any size, some computers will always be used up to the maximum in terms of performance, while a fraction of the maximum possible performance will be processed in other computers.

On the other hand, in the chosen distribution, there is no predefined block size, and this implies at least two advantageous consequences:

- A parameter is deleted in the parallel processing routines. Data distribution is invariant, there is no more than one possibility for distributing data distribution data.
- The block size optimizing each processor performance can be locally chosen and has no collateral effects; it does not affects the performance or the processing in the remaining computers.

From the ScaLAPACK point of view, this can be considered as an optimization definable given the heterogeneity of the installed local networks.

Data Distribution and Processors Interconnection. In both algorithms oriented to parallel computers of distributed memory presented in the previous chapter and the one presented in this chapter, notice that the data distribution is closely related to the processing algorithm and to the processors interconnection itself. In fact, this is a distinctive characteristic of most (or all) of the numerical algorithms oriented to distributed memory parallel computers. Thus, a two-dimensional data distribution is directly related to an underlying parallel computation architecture that counts with at least two characteristics:

- Distributed memory.
- Interconnected processors with mesh topology or two-dimensional torus.

As already explained, the local area networks interconnection based on Ethernet (most of them) can vary mainly depending on the wiring, and more specifically on the use of hubs and/or switches. In the networks in which switches are used, the physical interconnection can be considered as a mesh or two-dimensional torus (and, in fact, as in other several ways, given the flexibility of networks with switches), but it is clear that this is not valid for all the networks, in which hubs can be found. In general, what is actually valid is immediately considering the fact that in all the local networks based on Ethernet ("by hardware", given the definition of Ethernet), computers interconnection is given logically by a communications bus, such as the one of Figure 3.1 or that of Figure 3.2.

Thus:

- A local network computers interconnection cannot always be obtained immediately as a mesh or two-dimensional torus. This implies that all the two-dimensional data distributions with their computing algorithm oriented to two-dimensional interconnection networks will tend to be penalized in terms of messages serialization over the communications bus defined by Ethernet, over which collisions will arise in data transferences.
- When considering a data distribution as the presented in this chapter, the processors interconnection in "one-dimensional topologies" is also under consideration, such as that of the bus defined by Ethernet or the rings with point-to-point connections. More specifically, the data distribution presented in this chapter is oriented to the direct utilization of the local networks, which are mostly based on Ethernet, and, more independently, of the potential variety in the wiring of the same.

Load Balance of the Algorithm. As asserted in general, the load balance is trivial in the case of the Two-dimensional Block Cyclic Decomposition. This characteristic is kept also in the context of installed local networks heterogeneous processors.

But from the point of view of the distribution presented in this chapter, neither the load balance nor the utilization generality in different circumstances are lost, since:

- The processing load balance is kept in the case of parallel computers with homogeneous processors,
- The processing load balance is kept in the case of parallel computers with heterogeneous processors,
- The load balance is trivial, even though - in the case of heterogeneous processors- the relative processors speed must be known beforehand.

3.5 Chapter Summary

All throughout the chapter, the main concepts presented are:

- The characteristics of the clusters interconnected by Ethernet networks, both homogeneous and heterogeneous, when they are used as parallel computing platforms.
- The principles of applications parallelization to be solved specifically over clusters in order to obtain optimized performance. These principles are oriented to the optimized use of all the characteristics of local computing and processing in clusters.
- Two parallel algorithms for matrix multiplication specifically oriented to obtaining optimized performance in clusters.
- Commentaries necessary to the characterization of each algorithm and, also, the comparisons of these algorithms with those used in ScaLAPACK, for instance, basically regarding data distribution.

This thesis makes clear the differences not only in terms of the proposed algorithms data distribution, but also as regards the use (and, in a way, dependency) of broadcast messages as the sole manner of communicating data among processes of a parallel application.