Proceedings of the *International Conference on Theory and Application of Mathematics and Informatics ICTAMI 2005* - Alba Iulia, Romania

MEASURING THE PERFORMANCE FOR PARALLEL MATRIX MULTIPLICATION ALGORITHM

Costel Aldea

Abstract. After configuring and activating a parallel or distributed computing medium it is important to know and to have the possibility to decide if the algorithm solved by this mechanism solves the problems in the desired time with respect with the initial parameters of the system. For this experiment it is configured a massage passing interface medium with many computers working in parallel and on 2 computers of this medium there are installed two high precision clocks GPS clocks, that use the Network Time Protocol for the synchronization with the satellites. Further a C program that does the matrix multiplication in parallel is implemented using the MPI libraries. For this algorithms, using this medium and the mentioned tools some performance monitoring measures are made.

2000 *Mathematics Subject Classification*: 68W10, 65Y20.

1. Introduction

Lately a good performance for the computers has been reached due to the software and hardware components. However there are enough problems that cant be solved on the existing hardware in the right time. World wide many teams of researchers are doing experimental studies for constructing the future virtual organization and computers using the distributed resources, such kind of projects are grids, Internet computing, scalable computing, global computing. When multiple users are using others users platforms then the performance can be influenced in various and unexpected ways. For the efficient use of the worldwide distributed platforms one use different adaptation techniques for the distributed resource management and scheduling. Always o good question in this direction will be if the distributed application has or not the right performance and consequently the measurement of the performance parameters for a such application.

In a distributed system the components work together being interconnected and the actions in the most of cases coordinated using the message passing. Observing this definition it is obvious that a distributed system has to deal with: the concurrent running of the components, missing of a global clock, the possibility for the component to fail separately. There are two very big examples of distributed systems: the Internet and an Intranet, which is a part of the Internet but managed by an administrator

The common use of resource it is a sufficient motivation for the construction of the distributed systems. The resources can be managed by a server and accessed from clients or can be encapsulated in objects such that all the other objects can access and use them.

The common problems of the distributed system are heterogeneity of the components, inserting and replacing the components, security, scalability, error handling, concurrent execution and transparence.

In the following work is made the empirical approach to a distributed system for measuring the execution time. The described experiment is constructed.

2.Install and configure NTP and PPS on a Linux system

The following installation and configuration steps must to be made on a Linux system for activating the Network Time Protocol (NTP) together with PPSkit (Pulse Per Second). The presented steps were tested on a Debian GNU/Linux 3.0r1. Theoretically the steps are the same on all Linux distributions. As a prerequisites one must to be mentioned that on each used machine, Debian GNU/Linux 3.0r1, it work the standard version of the kernel, 2.4.18-bf2.4. For installation of the PPSkit the kernel must be patched, recompiled and installed and then configured the protocol NTP. The GPS clocks with Meinberg GPS167TGP are connected directly to the normal computers (x86) using serial ports.

Adding PPS to the kernel

Preparation

First of all one must be assure that has the needed software. The linux kernel website gives for downloading http://www.kernel.org/. When the right version is taken into consideration it must be compatible with PPSkit. The best method to reach that is to consult in parallel the PPSkit webpage where one can see the compatible versions. If we dont have any graphical program

(easiest way) for visualizing the archive, like ark, then the archive must be downloaded and consulted using tar utility:

tar tvjf PPSkit-2.1.1.tar.bz2

In the next one search the line containing 'patch-x.x.xx'

-rw-r–r– ezbs/users 301872 2003-04-28 22:12:01 PPSkit-2.1.1/patch-2.4.20

The patch doesnt work correctly if the kernel version isnt identical with this (in this case). The best way in obtaining the two compatible versions is to download first the patch and then with respect to the number that is inside the archive on the website of the kernel to download the corresponding kernel (http://www.kernel.org/). After download we copy and untar the archive in /usr/src:

cp ./linux-2.4.20.tar.bz2 /usr/src/

cd /usr/src/

tar xvjf /usr/src/linux-2.4.20.tar.bz2

If we have more than one source for the kernel than is good to build a symbolic link to the current version using:

ln -s /usr/src/linux-2.4.20/ linux

Then we untar the corresponding PPSkit:

cp ./PPSkit-2.1.1.tar.gz /usr/src/

cd /usr/src

tar xvzf PPSkit-2.1.1.tar.gz

cd /usr/src/linux

patch -p1 ¡../PPSkit-2.1.1/patch-2.4.20

"make config" and compiling the kernel

The new kernel must be configured and compiled:

cd /usr/src/linux/

make config

If there is installed the package ncurses (in Debian using apt-get install libncurses5-dev), the 'make config' command can be replaced with 'make menuconfig'. (The another alternative is 'make xconfig'). Regardless of which compilation method is used, the important thing is that the next properties are included and compiled within the kernel. The sign [*] signify that the option is activated (answer Y when make config is running).

In the section Code maturity level options

[*] Prompt for development and/or incomplete code/drivers

In the section: Processor type and features

[*] NTP kernel support[*] NTP PPS support

In the section Character devices
[*] NTP PPS support on serial port[*] Support for console on serial port
After follows the compilation:
make dep
make bzImage

This phase can last long time and doesnt need assistance, so that with respect to the computer power this can be left alone to work for a short or long time until the compilation is ready. Then follows the modules installation (only if it is the case):

su
make modules
make modules_install

Kernel instalation

The compressed image of the kernel reside in /usr/src/linux/arch/i386/boot/ and has the name bzImage. The installation is made as follows:

cd /usr/src/linux/
su
cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.20
cp System.map /boot/System.map-2.4.20
cp .config /boot/config-2.4.20

After this steps it is required also to modify the operating system boot loader by adding one new entry with the new kernel (in the file vi /etc/lilo.conf): image=/boot/vmlinux-2.4.20label=2.4.20read-only

After rebooting the system and running the command uname r we should see:

uname -r2.4.20-NANO

Such that the PPS kernel is ready and follows the NTP instalation.

Install and configure NTP

In the next we download the last NTP version from the web: http://www.ntp.org/. (direct link for download: http://www.eecis.udel.edu/ ntp/ntp_spool/ntp4/ntp-4.1.2.tar.gz)

Add the needed header filed:
mv /usr/include/sys/timex.h /usr/include/sys/timex.h.old
ln -s /usr/src/linux/include/linux/timex.h /usr/include/sys/timex.h
ln -s /usr/src/linux/include/linux/timepps.h /usr/include/sys/timepps.h
bf Untar and installation
tar xvzf ntp-4.1.2.tar.gz

cd ntp-4.1.2/
./configure –enable-MEINBERG
make
su
make install

Configure

For the synchronization of the NTP demon with the GPS Meinberg clock and the PPS should be created symbolic links between the serial port 2 (where the GPS clock is connected) and the serial port 1 (where the pulse is read):

cd /dev/
ln -s ttyS1 refclock-0
ln -s ttyS0 pps0

Then the NTP configuration file (/etc/ntp.conf) must be modified:

```
### Configuration section ###
server 127.127.8.0 mode 135 prefer  # Meinberg GPS167 with PPS
fudge 127.127.8.0 time1 0.0042      # relative to PPS for my hardware

server 127.127.22.0                 # ATOM(PPS)
fudge 127.127.22.0 flag3 1 flag2 1  # enable PPS API

enable pps

enable stats
driftfile /etc/ntp.drift    # remember the drift of the local clock

### Statistics section ###
statistics loopstats clockstats
statsdir /var/log/ntplog
filegen peerstats file peers type day link enable
filegen loopstats file loops type day link enable
filegen clockstats file clocks type day link enable
logconfig =all
logfile /var/log/ntp.log
```

In the file /etc/ntp.conf must exist at lest one reference time source described using the keyword server. Here we have the following reference time sources

- Meinberg GPS 167, with the IP address 127.127.8.0

- The Pulse-per-second transmitter (again the Meinberg clock), with the IP address 127.127.22.0

At the IP address association one have to watch that the lest two numbers (127.127.y.x) are the same with the descriptor refclock for the devices in the folder /dev. For example for the Meinberg clock is created the link /dev/refclock-0, it results that the address in the configuration file must be ended in 0 (server 127.127.8.0). Similar is for the PPS impulses at the serial port 1, the link /dev/pps-0, resulting server 127.127.22.0 The mode in the line server 127.127.8.0 mode 135 prefer is specified from the clock producer and is different for each clock. One can see another details in the NTP archive.

Starting the NTP demon:

su

ntpd

Using the command ntp -q it could be verified if the ntpd is started and works.

3. MPI - Message Passing Interface

MPI (Message Passing Interface) is a specification for a standard library for message passing that was defined by MPI forum - a group consisting in parallel computer distributors, library programmers and application specialists. Were developed more implementations of this specification. In the next one use $MPICH$, which is a freeware distribution and that has as a main goal combination of performance with portability.

The project that offers a portable implementation for MPI it began in the same time with the process of it's definition. The idea was to offer an answer to the decisions taken by the forum and to offer an implementation who permits definition experiments for the users even if these aren't entirely developed. The implementation scope was the inclusion of all the systems capable to use the message passing. $MPICH$ is a free and complete implementation of the MPI specifications projected to be portable and also efficient. "CH" from $MPICH$ comes from chameleon, adaptability to medium symbol and like that symbol for the portability. Chameleons are quickly and so from the beginning a secondary scope was a bigger portability against efficiency.

So that $MPICH$ is a research project and also a software implementation project. As a research project is scope is to search methods for the elements

describing that a parallel computer programmer must use to reach the performance of the physical components that are at his disposal. For $MPICH$, was adapted the programming constraint for use of MPI, the constraints related to the architecture of the used machine were rejected and retained only as a scope the high performance (bandwidth and latency for the message passing operations). As a software project it's scope was to promote the standards by offering to the users a freeware and high performance implementation on platform diversity, on the other hand the firms were helped to offer their own implementation.

4. Matrix multiplication

Matrix multiplication is often used in numerical algorithms, graph theory, digital image processing and signal processing. The matrix multiplication is a time consuming operation ; in medium a complexity $O(n^3)$, where n is the dimension of the matrix. While all the actual application need a lot of computations the researchers have tried to overcome this problem. Also the advantages of the Strassens algorithm have a limited performance so that the parallel matrix multiplication it always is a big concern.

The majority of the parallel matrix multiplication algorithms are based on the number of the processors. Between those algorithms one can enumerate: systolic algorithm, Cannons algorithm, Fox and Ottos algorithm, PUMMA [1], SUMMA [2], DIMMA [3]. Each of those are using the decomposition of matrix into blocks. At the runtime a processors evaluates the partial results where it has access. It does the successive computation with the new blocks adding the current result to the previous result. When all the multiplications are complete the root processor sum all the partial result and give the result.

Matrix decomposition

For the implementation of the parallel matrix multiplication the matrix A and B are divided into blocks. There exist many methods for a good matrix decomposition:

- one dimensional decomposition the matrix is divided horizontal. The processor i retains the blocks i, A_{block} and B_{block} and send them to the processors $(i-1)$ and $(i+1)$. The processors 0 and $(n-1)$ communicate to each other like into a ring topology.

[1]Parallel Universal Matrix Multiplication
[2]Scalable Universal Matrix Multiplication
[3]Distributed Independent Matrix Multiplication

- two dimensional decomposition in this case the matrix is divided into quadrates. Each processors communicate with all its neighbors (north , south, west, east).

- general two dimensional composition the matrix is divided into templates of processors with two dimension. Each processor communicate with its neighbors like in the case of the simple two dimensional decomposition. IN plus in this case are possible not only quadrate templates (for example 2×3 ąnd 3×4 with 6 ąnd respectively 12 processors).

- scattered two dimensional decomposition the matrix is divided in some sets of blocks. Each block contain a number of elements equal with the number of processors and each element from a set of blocks is divided with respect to the two dimensional templates of processors. The two dimensional templates of processors are containing 2×2, 2×3, 3×3, 3×4 ąnd 4×4 structures for $4, 6, 9, 12$ ąnd respectively 16 processors.

Fox

The Fox algorithm divides the matrix into data blocks. If the matrix has $dimMatrix$ elements and the program is executed on $nrProcs$ processors then the number of blocks is equal with the number of processors and the size of a block is: $block\_size = dimMatrix/nrProcs\_x$, where $nrProcs\_x$ is the number of horizontal processes. For simplify, in our case are used only quadrate matrix and then $nrProcs\_x = |\sqrt{nrProcs}|$.

At the beginning the root process reads and then sends the second matrix in block forms to the other processes so that each process has the corresponding block from the second matrix.

In parallel each of the process receive the matrix block.

For each process (root and the other) the result block is initialized with null.

For each line from the first matrix are send to each process the corresponding diagonal elements from the first matrix. It makes the product between those elements and the block from the previous step. The diagonal is shifted to the left and the block from the second matrix is shifted to the bottom using the message passing with the neighbor processes.

The root process computes also the final result after receiving the partial results.

The MPI matrix multiplication program is passing three types of messages:

- send one block from the second matrix to each process (from type DATA_B)
- send the block diagonals from the first matrix (from type DATA_A)
- send at the end the partial result to the root process (from type RESULT).

Example:
Matrix multiplication on 4 processors:

$$\begin{pmatrix} \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix} & \begin{pmatrix} 3 & 4 \\ 3 & 4 \end{pmatrix} \\ \begin{pmatrix} 1 & 3 \\ 1 & 3 \end{pmatrix} & \begin{pmatrix} 2 & 4 \\ 2 & 4 \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 4 & 3 \\ 1 & 3 \end{pmatrix} & \begin{pmatrix} 2 & 1 \\ 2 & 4 \end{pmatrix} \\ \begin{pmatrix} 1 & 2 \\ 3 & 2 \end{pmatrix} & \begin{pmatrix} 3 & 4 \\ 4 & 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 21 & 23 \\ 21 & 23 \end{pmatrix} & \begin{pmatrix} 31 & 25 \\ 31 & 25 \end{pmatrix} \\ \begin{pmatrix} 21 & 24 \\ 21 & 24 \end{pmatrix} & \begin{pmatrix} 30 & 25 \\ 30 & 25 \end{pmatrix} \end{pmatrix}$$

Where, usually, the value 31 from the line one of the result matrix is obtained like that $31 = 1 \times 2 + 2 \times 2 + 3 \times 3 + 4 \times 4$. Using the Fox algorithm this value is obtained in the process 1 (describing with 0,1,.. the processes) like that: product between the lines of the block and the diagonal of the corresponding block from the first matrix $(1, 2)$ $\begin{pmatrix} 2 & 1 \\ 2 & 4 \end{pmatrix}$ and it obtain: $\begin{pmatrix} 2 & 1 \\ 4 & 8 \end{pmatrix}$
Next the diagonal is shifted o the left and the block from the second matrix is shifted to the bottom and successively are made the following products:
$\begin{pmatrix} 2 & 4 \\ 3 & 4 \end{pmatrix}$ with the elements of diagonal $(2, 3)$
$\begin{pmatrix} 3 & 4 \\ 4 & 1 \end{pmatrix}$ with the elements of diagonal $(3, 4)$
$\begin{pmatrix} 4 & 1 \\ 2 & 1 \end{pmatrix}$ with the elements of diagonal $(4, 1)$

Sum of this results:

$$\begin{pmatrix} 2 & 1 \\ 4 & 8 \end{pmatrix} + \begin{pmatrix} 4 & 8 \\ 9 & 12 \end{pmatrix} + \begin{pmatrix} 9 & 12 \\ 16 & 4 \end{pmatrix} + \begin{pmatrix} 16 & 4 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 31 & 25 \\ 31 & 25 \end{pmatrix}$$

give a block of the final result.

Some result using a MPI implementation of the Fox algorithm

For the Fox algorithm the following data were collected running the program 100 times:

- Total running time with matrix dimension 1600x1600, 16 processes on 16 coputers, one processor each:

Measured Int.	Int. start	Int. stop
63.956421852	17:06:16 0.886212826	17:07:20 0.842634678
63.547276974	17:07:20 0.845926762	17:08:24 0.393203735
63.516219616	17:08:24 0.396577597	17:09:27 0.912797213
63.519518852	17:09:27 0.915988207	17:10:31 0.435507059
63.520165443	17:10:31 0.438711405	17:11:34 0.958876848
63.511228085	17:11:34 0.962056398	17:12:38 0.473284483
63.518872261	17:12:38 0.476442337	17:13:41 0.995314598
63.507167101	17:13:41 0.998455524	17:14:45 0.505622625
63.522676706	17:14:45 0.508868694	17:15:49 0.031545401
63.504557371	17:15:49 0.034837484	17:16:52 0.539394855
63.520976305	17:16:52 0.542572260	17:17:56 0.063548565
63.532964230	17:17:56 0.066716194	17:18:59 0.599680424

Table 1: Total running time - matrix product 1600x1600/16 processes/16 computers

- barrier time, from the arriving of the root process to the barrier until the barrier is passed.

0.000726938	19:00:11 0.771681309	19:00:11 0.772408247
0.000687838	19:01:15 0.320037603	19:01:15 0.320725441
0.000654459	19:02:18 0.862794161	19:02:18 0.863448620
0.000651360	19:03:22 0.410071373	19:03:22 0.410722733
0.000653982	19:04:25 0.965284824	19:04:25 0.965938807
0.000690937	19:05:29 0.517423630	19:05:29 0.518114567
0.000693560	19:06:33 0.065347672	19:06:33 0.066041231
0.000698090	19:07:36 0.627914667	19:07:36 0.628612757
0.000684977	19:08:40 0.199651957	19:08:40 0.200336933
0.000654221	19:09:43 0.754739046	19:09:43 0.755393267
0.000693560	19:10:47 0.306187391	19:10:47 0.306880951
0.000651360	19:11:50 0.843770266	19:11:50 0.844421625

Table 2: Barrier time - matrix product 1600x1600/16 processes/16 computers

- time from the blocking *send* start until its end.

0.051883936	21:50:03 0.699914455	21:50:03 0.751798391

```
| 0.051164389 | 21:50:03 0.769125462 |  21:50:03 0.820289850 |
| 0.051176548 | 21:50:03 0.837716818 |  21:50:03 0.888893366 |
| 0.051173449 | 21:50:03 0.906309843 |  21:50:03 0.957483292 |
| 0.051983595 | 21:50:03 0.975007772 |  21:50:04 0.026991367 |
| 0.051350117 | 21:50:04 0.045039177 |  21:50:04 0.096389294 |
| 0.051228285 | 21:50:04 0.113791943 |  21:50:04 0.165020227 |
| 0.051254272 | 21:50:04 0.182529688 |  21:50:04 0.233783960 |
| 0.051250458 | 21:50:04 0.251266003 |  21:50:04 0.302516460 |
| 0.051243305 | 21:50:04 0.320041656 |  21:50:04 0.371284962 |
| 0.051254749 | 21:50:04 0.388795614 |  21:50:04 0.440050364 |
| 0.051229477 | 21:50:04 0.457580328 |  21:50:04 0.508809805 |
-------------------------------------------------------------
```

Table 3: Send time - matrix product 1600x1600/16 processes/16 computers

- Total running time with matrix dimension 1600x1600, 64 processes on 16 computers, one processor each:

```
-------------------------------------------------------------
| 66.639107704 | 19:57:23 0.129729986 |  19:58:29 0.768837690 |
| 65.565699816 | 19:58:29 0.775774956 |  19:59:35 0.341474771 |
| 65.337359190 | 19:59:35 0.348362684 |  20:00:40 0.685721874 |
| 65.387441397 | 20:00:40 0.692636728 |  20:01:46 0.080078125 |
| 66.049248219 | 20:01:46 0.087269306 |  20:02:52 0.136517525 |
| 65.443219423 | 20:02:52 0.143181562 |  20:03:57 0.586400986 |
| 65.751941919 | 20:03:57 0.593228102 |  20:05:03 0.345170021 |
| 65.998049498 | 20:05:03 0.352014780 |  20:06:09 0.350064278 |
| 65.667123795 | 20:06:09 0.356660128 |  20:07:15 0.023783922 |
| 65.155098915 | 20:07:15 0.031222105 |  20:08:20 0.186321020 |
| 65.658370495 | 20:08:20 0.193147659 |  20:09:25 0.851518154 |
| 65.561710596 | 20:09:25 0.858139992 |  20:10:31 0.419850588 |
-------------------------------------------------------------
```

Table 4: Total running time - matrix product 1600x1600/64 processes/16 computers

- barrier time, from the arriving of the root process to the barrier until the barrier is passed.

```
-------------------------------------------------------------
| 0.002666950 | 13:53:45 0.603855610 |  13:53:45 0.606522560 |
| 0.002766371 | 13:54:50 0.567428112 |  13:54:50 0.570194483 |
```

```
| 0.002991438 | 13:55:55 0.517137766 |  13:55:55 0.520129204 |
| 0.002712965 | 13:57:01 0.281398535 |  13:57:01 0.284111500 |
| 0.003050327 | 13:58:06 0.224049568 |  13:58:06 0.227099895 |
| 0.003218651 | 13:59:11 0.019721031 |  13:59:11 0.022939682 |
| 0.002737999 | 14:00:16 0.219644070 |  14:00:16 0.222382069 |
| 0.003038645 | 14:01:21 0.306068897 |  14:01:21 0.309107542 |
| 0.002910852 | 14:02:27 0.145992279 |  14:02:27 0.148903131 |
| 0.003094196 | 14:03:32 0.560567856 |  14:03:32 0.563662052 |
| 0.002947807 | 14:04:37 0.981052160 |  14:04:37 0.983999968 |
| 0.002984762 | 14:05:42 0.719491959 |  14:05:42 0.722476721 |
-------------------------------------------------------------
```

Table 5: Barrier time - matrix product 1600x1600/64 processes/16 computers

- Total running time with matrix dimension 160x160, 16 processes on 16 computers, one processor each:

```
-------------------------------------------------------------
| 0.456625462 | 16:25:40 0.862607002 |  16:25:41 0.319232464 |
| 0.382536650 | 16:25:41 0.320107222 |  16:25:41 0.702643871 |
| 0.368335485 | 16:25:41 0.703450918 |  16:25:42 0.071786404 |
| 0.367173672 | 16:25:42 0.072438478 |  16:25:42 0.439612150 |
| 0.370925665 | 16:25:42 0.440252066 |  16:25:42 0.811177731 |
| 0.379559994 | 16:25:42 0.811825514 |  16:25:43 0.191385508 |
| 0.373143196 | 16:25:43 0.192024231 |  16:25:43 0.565167427 |
| 0.383926392 | 16:25:43 0.565838337 |  16:25:43 0.949764729 |
| 0.381397247 | 16:25:43 0.950471163 |  16:25:44 0.331868410 |
| 0.372228861 | 16:25:44 0.332519531 |  16:25:44 0.704748392 |
| 0.404785872 | 16:25:44 0.705416918 |  16:25:45 0.110202789 |
| 0.378345966 | 16:25:45 0.110870361 |  16:25:45 0.489216328 |
-------------------------------------------------------------
```

Table 6: Total running time - matrix product 160x160/16 processes/16 computers

- barrier time, from the arriving of the root process to the barrier until the barrier is passed.

```
-------------------------------------------------------------
| 0.000337839 | 16:22:59 0.389591694 |  16:22:59 0.389929533 |
| 0.000429630 | 16:22:59 0.773173809 |  16:22:59 0.773603439 |
| 0.000441074 | 16:23:00 0.143934965 |  16:23:00 0.144376040 |
```

0.000424862	16:23:00 0.523376226	16:23:00 0.523801088
0.000426769	16:23:00 0.893172026	16:23:00 0.893598795
0.000437498	16:23:01 0.265645027	16:23:01 0.266082525
0.000442743	16:23:01 0.656387806	16:23:01 0.656830549
0.000436783	16:23:02 0.055243254	16:23:02 0.055680037
0.000392914	16:23:02 0.432395697	16:23:02 0.432788610
0.000287294	16:23:02 0.801849365	16:23:02 0.802136660
0.000430822	16:23:03 0.205770254	16:23:03 0.206201077
0.000428915	16:23:03 0.582705498	16:23:03 0.583134413

Table 7: Barrier time - matrix product 160x160/16 processes/16 computers

5. Conclusions

There is a large spectrum of parameters that can be observed with respect to the performance of the parallel multiplication algorithms in particular and of the parallel algorithms in general.

Like in theory it is obvious that more resources bring more performance in our case. The cost is the need of a good understanding of parallelism and of the methods to distribute the computations.

It will be always difficult to evaluate the performance. The amount of data is very big. There are observed in the tables only number of processes, matrix dimension, barrier time and the total running time, but are still a lot of parameters that can be observed and introduced into an optimization if needed.

A multi criteria optimization of the distributed computing methods and algorithms is imposed and that implies also a lot of computations.

Also the hardware influences are notable and must be taken into consideration when using a parallel or distributed medium.

References

[1] Boian F.M., Programare distribuită în Internet. Metode şi aplicaţii, Editura Albastră, Cluj, 1998

[2] Boian F.M. şi alţii, Programare concurentă pe platforme Unix, Windows, Java, Editura Albastră, Cluj, 2002

[3] Cory Quammen, Introduction to Programming Shared-Memory and Distributed-Memory Parallel Computers, ACM Crossroads Student Magazine, 2002

[4] Lazăr I., Frenţiu M., Niculescu V., Programare orientată obiect în Java, Editura Universităţii "Petru Maior", Târgu Mureş, 1999

[5] Mellor-Crummey, J.M. şi Scott, M., Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors, ACM Transactions on Computer Systems, Feb. 1992.

[6] Thomas E. Anderson, The performance of spin lock alternatives for shared-money multiprocessors, IEEE Transaction on Parallel and Distributed Systems, 1990.

[7] W. Gropp, E. Lusk, N. Doss şi A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard", Parallel Computing, vol. 22, nr. 6, pg. 789–828, sep. 1996

[8] W. Gropp, Ewing Lusk, PVM and MPI are completely diffrent, Science Division, Argonne National Laboratory, sep. 1998

[9] William D. Gropp şi Ewing Lusk, User's Guide for `mpich`, a Portable Implementation of MPI, ANL-96/6, Mathematics and Computer Science Division, Argonne National Laboratory, 1996

[10] http://www.mcs.anl.gov/mpi/mpich2 - Implementarea MPICH2 pentru MPI

Aldea Constantin Lucian
Department of Computer Science
Faculty of Mathematics and Computer Science
University Transilvania Braşov
Address Iuliu Maniu 50, Braşov, 500091, Braşov, România
email:*costel.aldea@gmail.com*