The performance of the MATLAB Parallel Computing Toolbox in specific problems

Dimitris N. Varsamis*, Christos Talagkozis*, Paris A. Mastorocostas*, Evangelos Outsios* and Nicholas P. Karampetakis†
*Department of Informatics Engineering
Technological Educational Institute of Central Macedonia, Serres 62124, Greece
Email: dvarsam@teiser.gr, talagozis@hotmail.com, mast@teiser.gr, outsios@teiser.gr
†Department of Mathematics
Aristotle University of Thessaloniki, Thessaloniki 54124, Greece
Email: karampet@math.auth.gr

***Abstract*—In this work, we present the performance of three different parallel computing approaches of the MATLAB Parallel Computing Toolbox. In particular, we use the command "parfor", the command "spmd" and the technique "scheduler". The comparison of the three approaches in terms of computations and memory are presented. The three approaches are applied to two specific problems: a) searching of a value into a matrix and b) prime factorization. The first problem is bounded by MATLAB for the size of matrix, namely, has memory problems, and the second problem is bounded by MATLAB for numerical precision and time complexity. Finally, the executions of the corresponding parallel algorithms in a multi-worker lab are presented.**

***Index Terms*—MATLAB, Parallel Computing Toolbox, Searching, Prime factorization.**

I. Introduction

THE MATLAB Parallel Computing Toolbox (PCT) supports the two known parallel computer memory architectures such as the shared memory architectures and the distributed memory architectures [1], [2], [3], [4]. The shared memory architecture is implemented with the option "local" in a local computer with many cores [5]. Presently, the most common personal computers have CPU with two or four cores. The distributed memory architecture is implemented with the option "jobmanager" in a network of computers.

Additionally, the Matlab PCT supports the following parallel programming models: Data parallel, Distributed memory (message passing), Single Program Multiple Data (SPMD) and Multiple Program Multiple Data (MPMD). The command **parfor** implements the data parallel programming, the function **spmd** implements the SPMD and the technique **scheduler** implements both the distributed memory programming and the MPMD.

For the measurement of the performance of the Matlab PCT we select two specific problems: a) searching of a value into a matrix and b) prime factorization. The problem of the searching of a value into a matrix has limitation to the size of the matrix while the problem of the prime factorization has limitation to numerical precision and high computational cost.

This work was supported by the Research Committee of the Serres Institute of Education and Technology

A. The searching of a value into a matrix

The aim of this problem is to find a value into a large or a very large matrix. The questions for this problem are the following

- what is the maximum size of the matrix?
- how we can reduce the execution time?

The maximum size of a matrix in Matlab depends on the memory of the machine, the operating system and the release of Matlab. For example, in a PC with 4GB RAM, Windows XP(32 bit) and Matlab 7.3 the maximum memory which is used for an array is approximately 707 MB or 7.409×10^8 bytes (the function **memory** returns the maximum size of an array in Matlab), that means a matrix with

$$\frac{7.409 \times 10^8 \text{ bytes}}{8 \text{ bytes (double)}} = 92612500 \text{ cells}$$

Thus, we have a limit size for the simple use of Matlab. Generally, if we consider M the maximum size of an array then the array has

$$\text{Total Cells} = \frac{\text{Maximum size}}{\text{bytes for double}} = \frac{M}{8}$$

Consequently, we use the PCT for the confronting of the limitations for the size of matrix and for the reduction of the execution time.

The serial approach in MATLAB of this problem is the following function:

```
function pos = search_for(x,key)
n=length(x);
pos=[];
for i=1:n
    if x(i)==key
        pos=[pos i];
    end
end
```

where `x` is an array (dataset), `key` is the searching value and `pos` is an array with the positions of the key value. In Table I the results of the performance tests for this function are presented. The size of the array x is bounded, thus we cannot use an array with size greater than of 2^{27} cells.

TABLE I
THE PERFORMANCE OF THE SEARCHING PROBLEM IN SERIAL EXECUTION

Length of array x	Execution time	Comments
2^{15}	0.000333	
2^{20}	0.010854	
2^{25}	0.347254	
2^{27}	1.386923	Time problems
2^{28}	–	Memory problems

TABLE II
THE PERFORMANCE OF THE PRIME FACTORIZATION PROBLEM IN SERIAL EXECUTION

Order of number n	Execution time	Comments
2^{40}	0.069156	
2^{50}	2.189364	Time problems
2^{52}	6.152502	Time problems
2^{53}	–	Numerical precision problems
2^{54}	$\geq$ day	Use of Symbolic Toolbox

B. *Prime factorization*

The aim of this problem is to find the factorization of a very large integer number to two prime numbers. The prime factorization is the main idea of the RSA cryptosystem. The prime test is included in this problem, namely, an integer is or not prime number. The questions for this problem are the following

- what is the maximum number of digits of a number with double precision?
- how we can reduce the execution time?

The maximum number of decimal digits of a number with double precision is 16 because in double precision we have 8 bytes per number or $8 \times 8 = 64$ bits. That means a number with 53 binary digits or equivalently, a number with 16 decimal digits. This limitation can be confronted by using the Matlab Symbolic Toolbox. The main disadvantage of the symbolic toolbox is the high computational cost for the execution of simple arithmetic operations. Consequently, we use the PCT for the confronting of the limitations for the numerical precision and for the reduction of the execution time. The serial approach in MATLAB of this problem is the following function:

```
function [a,b] = prime_factorization(n)
k=double(floor(sqrt(sym(n))));
a=0;
b=0;
for m = 2:k
    if (mod(n,m) == 0)
        a=m;
        b=n/m;
        return
    end
end
```

where `n` is the number that is factorized and `a`, `b` are the factors, namely, $n = a \cdot b$. In Table II the results of the performance tests for this function are presented.

These specific problems are applied in three approaches of the Matlab PCT to reach conclusions and to find the advantages and disadvantages of each approach. In particular, in Section 2 we present the three parallel computing approaches which are the command **parfor** the command **spmd** and the technique **scheduler**. In Section 3 the performance of the parallel approaches are presented.

II. THE THREE PARALLEL COMPUTING APPROACHES

In this section, we present the command **parfor** which follows to Data parallel programming model. Additionally, we present the command **spmd** which follows to Simple Program Multiple Data programming model. Finally, we present the technique **scheduler** which follows to Distributed memory and Multiple Program Multiple Data programming model.

A. *The command parfor*

The command **parfor** (parallel loop) is MATLAB's simplest parallel programming command. **parfor** replaces the **for** command in cases where a loop can be parallelized. The loop iterations are divided up among different workers, executed in an unknown order, and the results gathered back to the main copy of MATLAB.

In the first problem we can replace the command **for** by the command **parfor** as see in the below function

```
function pos = search_parfor(x,key)
n=length(x);
pos=[];
parfor i=1:n
    if x(i)==key
        pos=[pos i];
    end
end
```

In the second problem we can not replace the command **for** by the command **parfor** because in body of loop the statement **return** exists which interrupts the execution of function when the condition of the statement if is true.

B. *The command spmd*

The **spmd** (single program multiple data) command is like working with a very simplified version of Message Passing Interface. There is one client process, supervising labs who cooperate on a single program. Each lab has an identifier, knows how many labs there are total, and can determine its behavior based on that identifier.

In the first problem we can partition the data according to the number of available labs and we can send these separately in each lab. The commands which are used are the following:

```
pos=0;
spmd(labs)
    for i=1:labs
        if(labindex==i)
           pos=search_for_file(key,n);
        end
    end
end
```

The function `search_for_file` is the same with the function `search_for` with the addition of a command which it reads the data from a file.

Similarly, in the second problem we can partition the number of iterations according to the number of available labs and we can execute these separately in each lab. The commands which are used are the following:

```
k=double(floor(sqrt(sym(n))));
step=ceil(k/labs);
spmd(labs)
   for i=1:labs
     if(labindex==i)
         [a,b]=prime_factorization_spmd(n,...
                        1+step*(i-1),step*i);
     end
   end
end
```

The function `prime_factorization_spmd` is the following

```
function [a,b] = prime_factorization_spmd(n,x,y)
a=0;
b=0;
m=x;
while m<=y
    if (mod(n,m)==0 && m~=1)
        a=m;
        b=n/m;
        return
    end
    m=m+1;
end
```

In above function the arguments x and y are the lower and the upper index of iterations.

C. The technique scheduler

The technique **scheduler** with a Job Manager manage a big job which is divided into vary independent tasks. For simplicity, assume each task will be carried out by the same MATLAB function. Each task runs on a single processor (although there's no need to rule out parallelism) and has its own memory. Tasks do not communicate while running; they start with input, they return results upon completion. When all the tasks are completed, it is possible to gather, analyze and plot the combined results.

In the first problem we can apply the technique **scheduler** with the following commands

```
jm = findResource;
pj = createJob(jm);
set(pj,'MinimumNumberOfWorkers',1);
set(pj,'MaximumNumberOfWorkers',64);
set(pj,'FileDependencies',...
         {'dataset25.mat','search_for_file.m'});
for i=1:labs
    obj(i)=createTask(pj,...
          @search_for_file, 1, {key,log2(n)});
end
submit(pj);
waitForState(pj);
out=getAllOutputArguments(pj);
```

In above program we create task for each lab and we send the tasks with the corresponding files in each lab. The sending files are the function `search_for_file` with specific arguments `key`, `log2(n)` and the file `dataset25.mat` with the dataset.

In the second problem we can apply the technique **scheduler** with the following commands

```
k=double(floor(sqrt(sym(n))));
step=ceil(k/labs);
jm = findResource;
pj = createJob(jm);
set(pj,'MinimumNumberOfWorkers',1);
set(pj,'MaximumNumberOfWorkers',64);
set(pj,'FileDependencies',...
        {'prime_factorization_spmd'});
for i=1:labs
    obj(i)=createTask(pj,...
          @prime_factorization_spmd,...
          1, {n,1+step*(i-1),step*i});
end
submit(pj);
waitForState(pj);
out=getAllOutputArguments(pj);
destroy(pj);
```

In above program we create task for each lab and we send the tasks with the corresponding file in each lab. The sending files are the function `prime_factorization_spmd` with specific arguments `n`,`1+step*(i-1)` and `step*i`.

III. The performance of the parallel approaches

The performance tests are implemented in a lab with 24 computers. The specifications of each computer are Intel Core Quad CPU (Q9400) at 2600 GHz with 4 Gb RAM. We run the parallel programs with 1, 2, 4, 8, 16, 32 and 64 cores. The sizes of the dataset of the searching problem are 2^{20} and 2^{25}. The number of digits of the prime factorization problem are order of 2^{40}, 2^{50} and 2^{52}. In Tables III and IV the results of the performance test are presented.

For the searching problem we have the following characteristics:

- we cannot use a matrix with total size (number of cells) greater than $\frac{M}{8}$, therefore we do not gain in memory issues with command **parfor**.
- we can use a matrix with total size (number of cells) greater than $\frac{M}{8}$, therefore we gain in memory issues with the command **spmd** and the technique **scheduler**. For example, if the numbers of labs is 64 and the matrix size in each lab is 2^{26} then we have total size $64 \times 2^{26} = 2^{32}$.
- the execution times (see Table III) show us that we do not gain in execution time in relation to serial execution in all approaches.
- the execution times (see Table III) show us that in matrix with size of 2^{20} cells the command **parfor** has better times instead of **spmd** and **scheduler**.
- the execution times (see Table III) show us that in matrix with size of 2^{25} cells the command **spmd** has better times instead of **parfor** and **scheduler** and while the cores increasing the time is the same.

TABLE III
EXECUTION TIMES FOR THE SEARCHING PROBLEM WITH MATRIX SIZE OF 2^{20} AND 2^{25}

cores	parfor	spmd	scheduler	parfor	spmd	scheduler
1	1.902	0.8367	1.1850	45.13	18.3763	23.4657
2	0.99	0.9235	1.2984	30.61	18.4263	30.3068
4	0.839	1.3980	1.5278	24.32	19.2775	41.0596
8	0.736	1.3143	1.8254	23.08	19.1012	52.6693
16	0.74	1.3900	2.4786	22.91	19.0348	72.3122
32		1.3630	3.6418		19.6694	109.2248
64		1.3431	6.1749		21.9759	187.8599

TABLE IV
EXECUTION TIMES FOR THE PRIME FACTORIZATION PROBLEM WITH NUMBER n ORDER OF 2^{40}, 2^{50} AND 2^{52}

cores	spmd	scheduler	spmd	scheduler	spmd	scheduler
1	0.2510	0.6341	4.0960	5.1087	10.8393	11.2883
2	0.1741	0.5984	2.0935	2.5601	5.6405	5.9604
4	0.1679	0.6360	1.2267	1.6284	3.2516	3.4032
8	0.1687	0.6317	0.6770	1.1801	1.6976	2.1890
16	0.1992	0.6297	0.4079	0.9058	0.9172	1.4083
32	0.2601	0.6659	0.3416	0.7943	0.5780	1.0249
64	0.4150	0.7617	0.4351	0.8650	0.5421	0.9388

For the prime factorization problem we have the following characteristics:

- In number with order of 2^{40} the execution times (see Table IV) are the same for the **spmd** and **scheduler**.
- In number with order of 2^{50} and 2^{52} the execution times (see Table IV) are decreasing while the cores are increasing for the **spmd** and **scheduler**.

The theoretical measurements of parallel computing are a) the speed up of the parallel algorithm which is given by

$$S_p = \frac{T_1}{T_p}$$

where T_1 is the execution time of the serial algorithm and T_p is the execution time of the parallel algorithm with p processors and b) the efficiency of the parallel algorithm which is given by

$$E_p = \frac{S_p}{p}$$

The efficiency of the searching problem are presented in Figures 1 and 2 and the efficiency of the prime factorization problem are presented in Figures 3,4 and 5.

IV. CONCLUSIONS

From the results of the performance tests and the theoretical measurements of parallel processing we conclude in the according observations: a) in problems with large or very large dataset we can use the MATLAB PCT so as to partition the dataset in each lab, which has its own memory. That is, we use the distributed memory parallel model. On the other hand, the PCT cannot reduce the execution times (very poor efficiency in all approaches). b) In problems with high computational cost we can use the PCT to reduce the computational time (good efficiency in very large numbers with small number of cores). In particular, the command **spmd** is the best choice for the searching problem. The command **spmd** is the best choice for the prime factorization problem with the number of cores less equal to 32 (efficiency $\geq 50\%$).

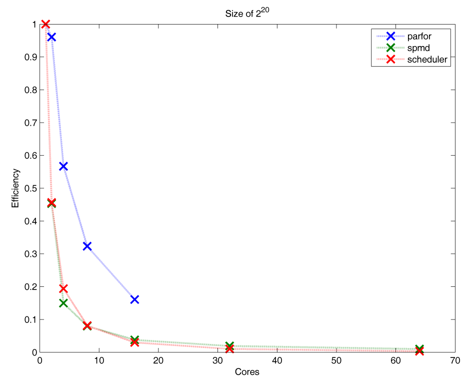

Fig. 1. The efficiency of the PCT in searching problem with matrix size of 2^{20}.

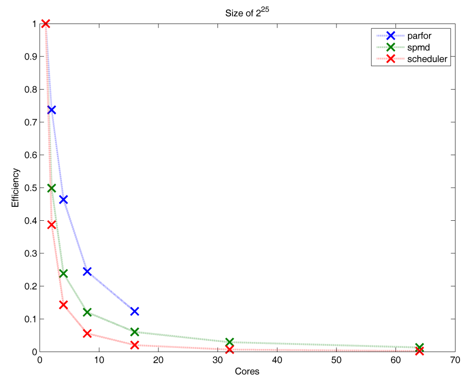

Fig. 2. The efficiency of the PCT in searching problem with matrix size of 2^{25}.

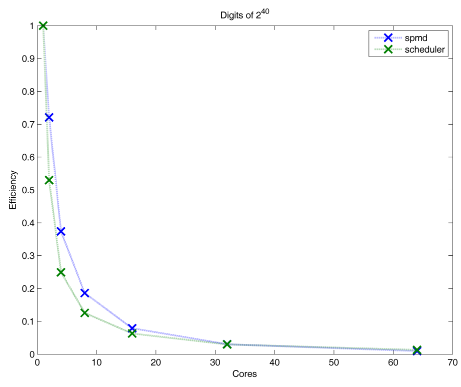

Fig. 3. The efficiency of the PCT in prime factorization problem with number with order of 2^{40}.

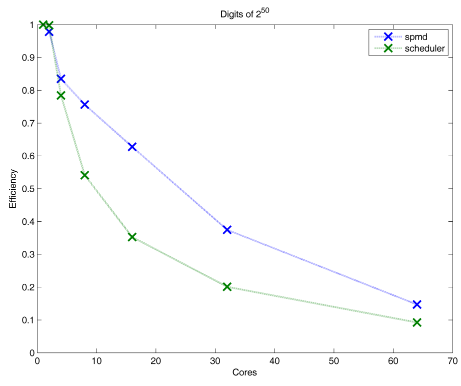

Fig. 4. The efficiency of the PCT in prime factorization problem with number with order of 2^{50}.

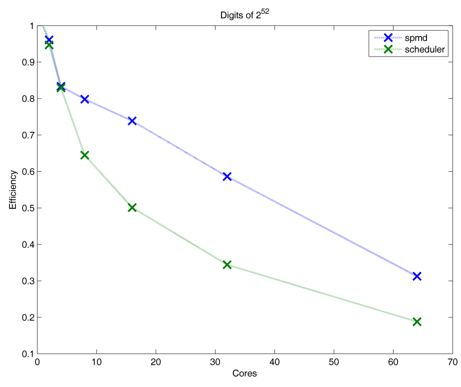

Fig. 5. The efficiency of the PCT in prime factorization problem with number with order of 2^{52}.

Acknowledgment

This work was supported by the Research Committee of the Serres Institute of Education and Technology.

References

[1] J. Kepner, *Parallel MATLAB for Multicore and Multinode Computers*. Philadelphia, USA: SIAM, 2009.
[2] P. Luszczek, "Parallel programming in matlab," *International Journal of High Performance Computing Applications*, vol. 23, no. 3, pp. 277–283, 2009.
[3] C. Moler, "Parallel matlab: Multiple processors and multiple cores," *The MathWorks News & Notes*, 2007.
[4] G. Sharma and J. Martin, "Matlab : A language for parallel computing," *International Journal of Parallel Programming*, vol. 37, pp. 3–36, 2009.
[5] D. N. Varsamis, P. A. Mastorocostas, A. K. Papakonstantinou, and N. P. Karampetakis, "A parallel searching algorithm for the insetting procedure in matlab parallel toolbox," in *Federated Conference on Computer Science and Information Systems (FedCSIS), 2012*. IEEE, 2012, pp. 587–593.
[6] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.
[7] J. W. Demmel, M. T. Heath, and H. A. van der Vorst, "Parallel numerical linear algebra," *Acta Numerica*, vol. 2, pp. 111–197, 1993.
[8] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, 2nd ed. Addison-Wesley, 2003.
[9] C. Lin and L. Snyder, *Principles of Parallel Programming*. Boston, USA: Addison-Wesley, 2008.
[10] C. Moler, *Numerical Computing with MATLAB*, 2nd ed. SIAM, 2008.