Proceedings of the Federated Conference on Computer Science and Information Systems pp. 553–560
ISBN 978-83-60810-51-4

Parallel Communication-Free Algorithm for Triangular Matrix Inversion on Heterogenoues Platform

Ryma Mahfoudhi, Zaher Mahjoub
University of Tunis El Manar, Faculty of Sciences of Tunis, Manar II 2092, Tunis, Tunisia
Email: rimahayet@yahoo.fr, zaher.mahjoub@fst.rnu.tn

Wahid Nasri
Higher School of Sciences and Techniques of Tunis (ESSTT), Montfleury-Tunis 1008, Tunisia
Email: wahid.nasri@ensi.rnu.tn

***Abstract*—We address in this paper the parallelization of a recursive algorithm for triangular matrix inversion (TMI) based on the 'Divide and Conquer' (D&C) paradigm. A series of different versions of an original sequential algorithm are first presented. A theoretical performance study permits to establish an accurate comparison between the designed algorithms. Afterwards, we develop an optimal parallel communication-free algorithm targeting a heterogeneous environment involving processors of different speeds. For this purpose, we use a non equitable and incomplete version of the D&C paradigm consisting in recursively decomposing the original TMI problem in two subproblems of non equal sizes, then decomposing only one subproblem and so on. The theoretical study is validated by a series of experiments achieved on two platforms, namely an 8-core shared memory machine and a distributed memory cluster of 16 nodes. The obtained results permit to illustrate the interest of the contribution.**

***Keywords*—communication free; divide and conquer; heterogeneous platform; parallel algorithm; recursive algorithm; triangular matrix inversion**

I. Introduction

TRIANGULAR matrix inversion (TMI) is a basic kernel in large and intensive scientific applications. TMI is commonly performed when calculating the explicit inverse of a (dense) matrix from its LU factorization. Given its cubic complexity i.e. $O(n^3)$ in terms of the matrix size (n), several works addressed the design of efficient parallel algorithms for solving this problem. Apart the standard TMI algorithm consisting in solving n linear triangular systems of size $n, n–1,...,1$ [1], a recursive algorithm, of same complexity, has been proposed by Heller in 1973 [2][3][4]. It uses the 'Divide and Conquer' (D&C) paradigm and consists in successive decompositions of the original matrix. To our knowledge, few original works have been devoted to the parallelization of this algorithm [5][6][7][8][9]. Our first objective here is the design of a series of sequential algorithms based on Heller's algorithm. Our second and main objective is the design of efficient parallel algorithms based on the former ones. The efficiency we target is in fact two-fold i.e. cost-optimality as well as communication free.

We have proposed in [10] a particular study on the parallelization of a divide and conquer algorithm for p homogeneous processors, ending up to a cost optimal parallel communication free algorithm. In this paper, we are interested in the generalization of this study to an heterogeneous platform.

The remainder of the paper is organized as follows. In section 2, we detail a theoretical study on diverse sequential versions of Heller's algorithm. Section 3 is devoted to the parallelization of the former designed algorithms. Finally, we present in section 4 an experimental study achieved on two target parallel machines i.e. an 8-core shared memory machine and a distributed memory cluster of 16 nodes.

II. Sequential Recursive TMI Algorithms

We first recall that the well known standard algorithm (SA) for inverting, say a lower triangular matrix [1], say A of size n, consists in solving n triangular systems AB(i) = e(i) of size $n–i+1$ ($i=1...n$) where B(i) (resp. e(j)) is the i-th column of the inverse matrix B (resp. identity matrix). The complexity of (SA) is as follows [1][6][7]:

$$C_{SA}=n^3/3+n^2/2+n/6. \quad (0)$$

I. Heller's Recursive Algorithm (HRA)

Using the Divide and Conquer (D&C) paradigm, Heller proposed in 1973 a recursive algorithm [2][3][4] for TMI. The main idea he used consists in decomposing matrix A as well as its inverse B (both of size n) into 3 submatrices of size $n/2$ (see Fig 1, A being assumed lower triangular). The procedure is recursively repeated until reaching submatrices of size 1. This original decomposition will be called henceforth complete equitable decomposition (CED). Here 'complete' means that the binary decomposition is applied to each triangular submatrix and 'equitable' means that, at each decomposition level, the submatrices are of same size.

We hence deduce:

$$B_1= A_1^{-1}, \quad B_3= A_3^{-1}, B_2= – B_3A_2B_1. \quad (2)$$

Therefore, inverting matrix A of size n consists in inverting 2 submatrices of size $n/2$ followed by two matrix products (triangular by dense) of size $n/2$. In [6][7] Nasri proposed a slightly modified version of the above algorithm. Indeed, since we have $B_2= – B_3A_2$ $B_1= – A_3^{-1}A_2A_1^{-1}$, let $Q= A_3^{-1}A_2$. From (2), we deduce:

$$A_3Q=A_2 \; ; \quad B_2A_1= –Q. \quad (3)$$

Hence, instead of two matrix products needed to compute B_2, we have to solve 2 matrix systems of size *n/2* i.e. $A_3Q=A_2$ and $(A_1)^T(B_2)^T=-Q^T$. The interest of such version is that neither B_1 ($=A_1^{-1}$) nor B_3 ($=A_3^{-1}$) are needed to compute B_2. The impact of this property will be seen when parallelizing the algorithm (see section 3). We precise that both versions are of $n^3/3+O(n^2)$ complexity [6][7].

Now, for sake of simplicity, we assume that $n=2^q$ ($q\geq 1$). Let RCA_0.5*_k be the (Recursive Complete) Algorithm designed by recursively applying the complete equitable decomposition (CED) until reaching a threshold size $n/2^k$ ($1\leq k\leq q$), the factor 0.5 meaning here that at each decomposition the matrix size is divided by 2 (i.e. multiplied by 0.5). The complexity of RCA_0.5*_k is as follows [6][7]:

$$C_{RCA\ 0.5*\ k}=n^3/3+n^2/2^{k+1}+n/6. \quad (4)$$

B. Variations of the D&C Paradigm

The standard D&C paradigm [4][11] is based on the CED as previously seen. As a matter of fact, we may derive other versions based on the notions of incomplete and non equitable decompositions as depicted in Fig 2. We precise that a decomposition is called non equitable (NE) when a given problem is decomposed into subproblems of different sizes, whereas 'incomplete 'designates the fact that the decomposition is not applied on all the derived subproblems.

Since the recursive TMI algorithm is binary, we'll associate to the non equitable decomposition (NED) a set of decomposition factors (i.e. ratios between the size of a subproblem and the size of its *father* problem) $\lambda_i \neq ½$, $i=1...k$. Remark that $\lambda_i=½$, corresponds to an equitable decomposition (ED) done at level *i*. The incomplete decomposition is achieved when, at each decomposition level, only one subproblem is decomposed. Furthermore, the decomposition is called static (resp. dynamic) when λ_i, $i=1...k$ is constant (resp. varies) at each decomposition level *i*. We summarize in Table I, the complexities of the standard algorithm SA, the recursive algorithm with complete dynamic non equitable decomposition RCA_$\lambda_{1...k}$_k (denoted RCA_λ*_k when $\lambda_i=\lambda : i=1...k$) and the recursive algorithm with incomplete dynamic non equitable decomposition RIA_$\lambda_{1...k}$_k (denoted RIA_λ*_k when $\lambda_i=\lambda : i=1...k$).

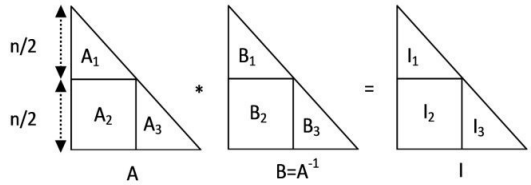

Fig 1. Matrix Decomposition in Heller's algorithm

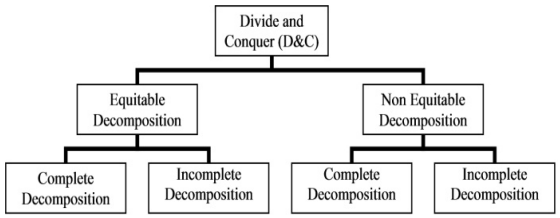

Fig 2. Versions of the D&C paradigm

Remark that RCA (resp. RIA) may be represented by a complete (resp. an incomplete) binary tree as depicted in Fig 3.

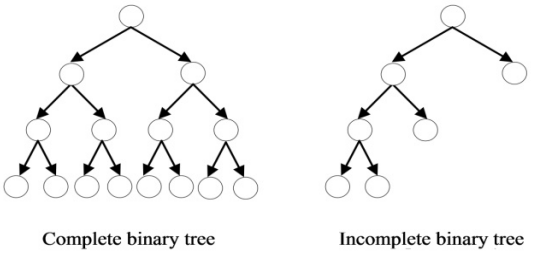

Fig 3. Recursive binary trees for algorithms RCA and RIA

From Table II, we may deduce the following:

- We always have an $n^3/3+O(n^2)$ complexity.
- The recursive versions are (slightly) better than the standard one as far as the $O(n^2)$ term is concerned.
- The complete decomposition version is slightly better that the incomplete one as far as the $O(n^2)$ term is concerned.
- The complexities of the recursive versions decrease when k increases as far as the $O(n^2)$ term is concerned, the best case occurring for $k=q=\log n$ i.e. we obtain $n^3/3+2n/3$ for RCA_0.5*_q and $n^3/3+n^2/6+n/6+1/3$ for RIA_0.5*_q.

By sorting the algorithms in decreasing performances, we get:

- RCA_0.5*_k (Equitable Complete Decomposition)
- RCA_$\lambda_{1...k}$_k (Dynamic Non Equitable Complete Decomposition)
- RIA_0.5*_k (Equitable Incomplete Decomposition)
- RIA_$\lambda_{1...k}$_k (Dynamic Non Equitable Incomplete Decomposition)
- SA

TABLE I. COMPLEXITIES OF THE DIFFERENT ALGORITHM VERSIONS

Algorithm	RCA_$\lambda_{1...k}$_k	RIA_$\lambda_{1...k}$_k
Complexity	$\frac{n^3}{3}+\frac{n^2}{2}\prod_{i=1}^{k}(2\lambda_i^2-2\lambda_i+1)+\frac{n}{6}$	$\frac{n^3}{3}+\frac{n^2}{2}\left(\prod_{i=1}^{k}\lambda_i^2+\sum_{j=1}^{k}\left[(1-\lambda_j)\prod_{i=j+1}^{k}\lambda_i\right]^2\right)+\frac{n}{6}$
Best case (ED: $\lambda_i=1/2$)	$\frac{n^3}{3}+\frac{n^2}{2^{k+1}}+\frac{n}{6}$	$\frac{n^3}{3}+\frac{n^2}{6}\left[1+\frac{1}{2^{2k-1}}\right]+\frac{n}{6}$
k=q	$\frac{n^3}{3}+\frac{2n}{3}$	$\frac{n^3}{3}+\frac{n^2}{6}+\frac{n}{6}+\frac{1}{3}$

We detail further in section 4, an experimental study validating the previous study and permitting to determine for each matrix size the best sequential algorithm.

III. PARALLELIZATION OF THE RECURSIVE TMI ALGORITHM

A. Brief State-of-the Art

The parallelization of the standard algorithm (SA) has been largely studied [12][13]. As to the recursive one, its parallelization interested fewer researchers to our knowledge. We may particularly cite [6][7] where Nasri studied TMI in both homogeneous and heterogeneous environments to design cost optimal parallel algorithms i.e. whose efficiency is asymptotically equal to 1. This performance could be reached thanks to a recursive task segmentation fitted to the number of available processors, who are recursively decomposed then grouped. The aim is to guarantee a perfect load balancing. Experimentations achieved only on two homogeneous multiprocessors show the practical efficiency of the approach.

In [8], the parallelization of the original Heller's algorithm is studied in a shared memory environment for p≤n processors and the designed algorithms are sub-optimal. Recently, Keqin Li [9] presented, in a survey paper, a deep theoretical study and gave precise optimal complexity results as well as a bounding interval for the minimal number of processors required to reach the optimal parallel complexity. However, the practical performances lack since no experimental study is achieved.

To conclude, the main remark one may make here is the fact that even if the designed algorithms known so far are optimal or (sub) cost-optimal, they all require inter-processor communications which may generate important overheads. On top of that, no original work has been devoted to the experimental study on heterogeneous platforms. Therefore, our contribution is to intend to fill this gap.

B. Machine model

We assume that the target multiprocessor system we use consists of a collection of ,p ,heterogeneous processors, denoted P_i ($i = 1,...,p$), each provided with a local memory, a cycle time t_i ($i = 1,...,p$) and connected by an homogeneous interconnection network [6]. Moreover, we assume that the speed of each processor (v_i for P_i) is known and does not vary during the program execution.

C. Parallel Algorithm

As previously seen, the designed algorithm in the (dynamic) incomplete decomposition case denoted RIA_$\lambda_{1...k}$_k, requires much less decompositions steps than its homologous RCA_$\lambda_{1...k}$_k in the complete case. We have shown that RIA has a much better degree of parallelism than RCA [10]. Therefore the study is limited to the incomplete decomposition case.

1)Description of the heterogeneous ARI algorithm for p=2

We start our study by describing the case where *p=2* processors are available. So, we assume that we have at one's disposal two processors P_1 and P_2 with speeds v_1 and v_2 respectively ($v_1 \geq v_2$). We note $v = v_1 + v_2$ and $\rho = v_1/v_2$. Let us first consider the RTMI algorithm with only one level decomposition, namely algorithm RCA_λ_1. We define the 3 following tasks:

T_1: $B_1 = A_1^{-1}$, T_2: $B_3 = A_3^{-1}$, T_3: $\{A_3Q = A_2,\ A_1^T B_2^T = -Q^T\}$

It is easy to notice that the 3 tasks are independent. However, if we consider the original Heller's algorithm consisting in two matrix inversions followed by two matrix products and adopt a similar task decomposition i.e. T_1: $B_1 = A_1^{-1}$, T_2: $B_3 = A_3^{-1}$, T_3: $B_2 = -B_3A_2B_1$, we'll have the following precedence relations: $T_1 \rightarrow T_3$, $T_2 \rightarrow T_3$. i.e. T_1 and T_2 are independent but must be finished before T_3 can begin. Therefore the modified version exhibits a higher parallelism (see Fig 4).

Let T_{23} be the (supertask) consisting in grouping T_2 and T_3 i.e. $T_{23} = T_2 \cup T_3$. Remark that now, any task corresponds either to a matrix inversion or to an inversion followed by

two matrix system resolution. The size of A_1 (denoted n_1) and that of A_2 (denoted n_2) are not necessarily $n/2$ (but in general different) since the two processors have not the same speed. In fact, n_1 and n_2 are chosen in terms of v_1 and v_2 in order to guarantee the optimality of the algorithm (see further). Therefore, the matrix A_3 is rectangular.

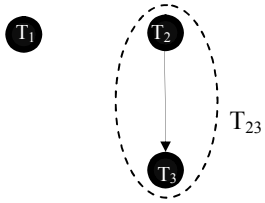

Fig 4. Precedence graph corresponding to the modified version

Here, we can assign T_1 to P_1 and T_{23} to P_2. In order to guarantee a perfect load balancing, we have to have Cost(T_1) = Cost(T_{23}). This leads (after equation solving) to $\lambda = \sqrt[3]{\rho / \rho + 1}$.

It should be noted that the value of the ratio ρ and the size of the matrix A are two factors that must be taken into account to determine whether the use of the slower processor is inefficient and hence decide to exclude it.

The calculation gives:

$$\rho \le \left((n-1)/n\right)^3 / (1 - \left((n-1)/n\right)^3) \Rightarrow \rho \le n/3$$

2) Generalization to p processors

Let us recall that the target multiprocessor machine is composed of p heterogeneous processors, denoted $P_1, P_2, \ldots, P_p$, with speeds $v_1 \ge v_2 \ge \ldots \ge v_p$ respectively. Let $s=\Sigma v_i$ $(i=1..p)$.

We detail now how to construct the optimal algorithm when we dispose of p heterogeneous processors. The principle consists in generalizing the approach developed in section 3 (p=2) by recursively decomposing the p processors into two sub-sets (each one corresponds to a virtual processor) as follows: let $\boldsymbol{L}=\{P_1,\ldots,P_p\}=\boldsymbol{L}_1 \cup \boldsymbol{L}_2$. $\boldsymbol{L}_1$ involves p_1 processors (and corresponds to a virtual processor $\mathcal{P}_1$) and $\mathbf{L}_2$ involves $p-p_1=p_2$ processors (and corresponds to a virtual processor $\mathcal{P}_2$). Thus, we end up in the case already seen of two processors $\mathcal{P}_1$ (whose speed is equal to the sum of the speeds of the processors of $\boldsymbol{L}_1$) and $\mathcal{P}_2$ (whose speed is equal to the sum of speeds of the processors of $\boldsymbol{L}_2$). $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ will then be decomposed each into two sub-sets and so on. We therefore generate a recursive binary decomposition tree.

The problem of the construction of this « optimal » decomposition (in fact partition) is known as the Set Partition Problem « SPP » [14]. It has been proved that the SPP is NP-complete. In addition, this decomposition generates communications (Fig 5). As an illustration, we present below (Fig 7) the decomposition for $p=4$, $V=\{10,7,5,3\}$, $s=\Sigma v_i=18$, $\mathcal{P}_1=\{P_1,P_4\}$, $s_1=13$, $\mathcal{P}_2=\{P_2,P_3\}$, $s_2=12$, $\rho=13/12=1.083$. We first calculate the appropriate λ to ensure the load balancing by solving the following equation: cost (T_1) = ρ (cost (T_{23})). The next step is to share T_1 (resp. T_{23}) between processors $\mathcal{P}_1$ (resp. $\mathcal{P}_2$). This division does not lead to communication between processors P_1 and P_4 (constituting $\mathcal{P}_1$). However, it will lead to communications between processors P_2 and P_3 (constituting $\mathcal{P}_2$). We add that the symmetric allocation i.e. which assigns T_1 (resp. T_{23}) to $\mathcal{P}_2$ (resp. $\mathcal{P}_1$) will generate communication between P_1 and P_4 but not between P_2 and P_3.

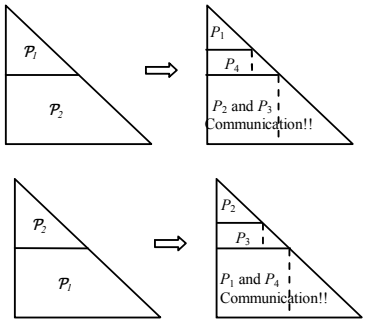

Fig 5. Communications generated by the equitable decomposition

The optimal solution consists in a recursive decomposition of the p processors into two subsets $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ such that $\boldsymbol{L}_1$ contains a single processor, i.e. the faster (P_1) and $\boldsymbol{L}_2$ contains the remaining $p-1$ processors.

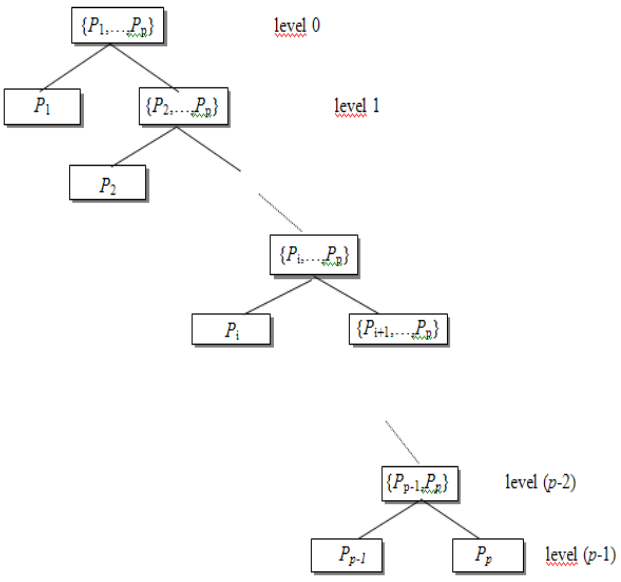

Fig 6. Recursive binary tree corresponding to a non equitable decomposition

Thus, we end up in the case already seen of two processors. After this first step, $\boldsymbol{L}_2$ (of cardinality $p–1$) will in turn be decomposed into two subsets and so on until leading to singleton subsets while $\mathbf{L}_1$ is a singleton for each decomposition (see Fig 6).

To be consistent, we will assume that the speed of $\mathcal{P}_1$, denoted v($\mathcal{P}_1$) is always greater than or equal to the speed of $\mathcal{P}_2$, denoted v($\mathcal{P}_2$). In other words, the virtual processor $\mathcal{P}_1$ which may consist of one or more processors will always be the fastest processor. Determining the values of λ reduces to solving cubic equations. It should be noted that in order to guarantee no communication at each level of decomposition, task T_{23} is always assigned to one processor while task T_1 is assigned to $p–1$ processors at level *1*, to $p–2$ processors at *2*, ..., and to *1* processor at level $p–1$. The tasks that are generated will be denoted $T_1^1 ... T_1^{p-1}$ and $T_{23}^1 ... T_{23}^{p-1}$.

As an illustration, we resume a previous example where p = 4 and V = {10,7,5,3} (see Fig 7).

- Step 1. $\mathcal{P}_2 = P_1$: Speed *10*, $\mathcal{P}_1=\{P_2,P_3,P_4\}$: Speed *15*, $\rho=15/10=1.5$

  T_1^1 (resp. T_{23}^1) is assigned to $\mathcal{P}_1$ (resp. $\mathcal{P}_2$ i.e. P_1) with cost (T_1^1) = ρ cost (T_{23}^1)
- Step 2. $\mathcal{P}_2 = P_2$: Speed *7* et $\mathcal{P}_1=\{P_3,P_4\}$: Speed *8*, $\rho=8/7=1.14$

  T_1^2 (resp. T_{234}^2) is assigned to P_1 (resp. $\mathcal{P}_2$ i.e. P_2) with cost (T_1^2) = ρ cost (T_{234}^2)
- Step 3. $\mathcal{P}_2 = P_4$: Speed *3* et $\mathcal{P}_1=P_3$: Speed *5*, $\rho=5/3=1.67$

  T_1^3 (resp. T_{23}^3) is assigned to. P_4 (resp. P_3) with cost (T_{234}^3) =ρ (cost (T_1^3)

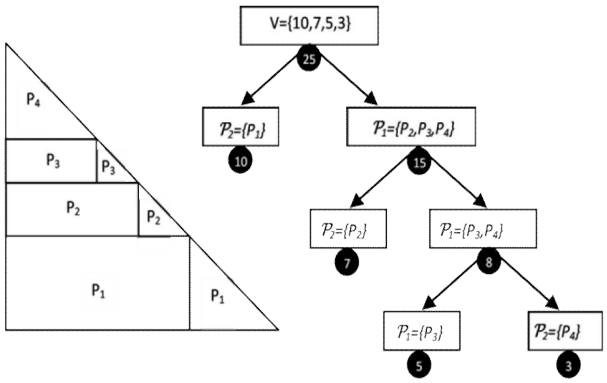

Fig 7. Processor decomposition in the example

In order to validate our theoretical contribution covering both the sequential and the parallel cases, we present in the next section an experimental study involving two parts. The first (resp. second) deals with the sequential (resp. parallel) algorithms.

IV. Experimental Studies

A. Sequential Algorithms

We discuss in this section the variations of the execution time in terms of the matrix size n. For this purpose, n was chosen in the range [64 6000], the input matrices involving real floating point elements randomly generated. The target machine is a 3 GHZ Fujitsu Siemens PC. C language under Linux OS was used. Algorithms SA, RCA_$\lambda_{1...k}$_k and RIA_$\lambda_{1...k}$_k (for different values of k) were implemented.

Remarks.

- As a first remark, we verified that for all the algorithms, the execution time (denoted ext) follows a cubic relation in terms of n (see Fig 8). On the other hand, the decomposition factor λ as well as the recursivity (decomposition) level k have important impacts on the execution time.
- The execution time decreases when k grows until reaching a threshold $k^*(n)$ which varies with n.
- For fixed k, the time ratio i.e. ext obtained for k divided by ext obtained for any k' such that $k<k' \leq k^*(n)$ increases with n. Hence better performances are reached for matrices of large sizes.

In Table II, we give for some values of n, the values of both k^*, $n^*=n/2^{k^*}$ (the corresponding size) and the time ratio ext(RCA_0.5_q)/ext(RCA_0.5*_k*). We deduce that the threshold size $n^*=n/2^{k^*}$ from which RCA becomes more efficient is around 32.

- Decomposition factor λ. We remark (see Fig 9) that, for fixed n and k, the execution time (ext) is minimum for $\lambda=0.5$ thus confirming our theoretical complexity analysis (see section 2.2). In fact, the variations of ext in terms of λ (for fixed n and k) are quadratic as precised in Table I. Let us add that the largest value of ext (in terms of λ) is at most equal to 1.15 (resp. 1.05) times its lowest one obtained for $n=1024$ (resp. 6000). Hence this ratio decreases when n increases and seems to converge to 1 for very large values of n.
- For (the dynamic) algorithms RCA_$\lambda_{1...k}$_k where a different decomposition ratio λ_i is chosen at each decomposition level i ($=1...k$), quite similar results were obtained. The same results occur for RIA_λ*_k and RCA_λ*_k as well for RIA_$\lambda_{1...k}$_k and RCA_$\lambda_{1...k}$_k.

TABLE II. VARIATIONS OF THE RATIO, K* AND N* IN TERMS OF MATRIX SIZE

N	400	512	800	1024	1600	2048	3200	4096	5000	6000
Ratio	1.14	1.26	1.34	1.41	1.46	1.56	1.62	1.73	2.02	2.26
k*	3	4	4	5	5	6	6	7	7	7
n*	50	32	50	32	50	32	50	32	36	49

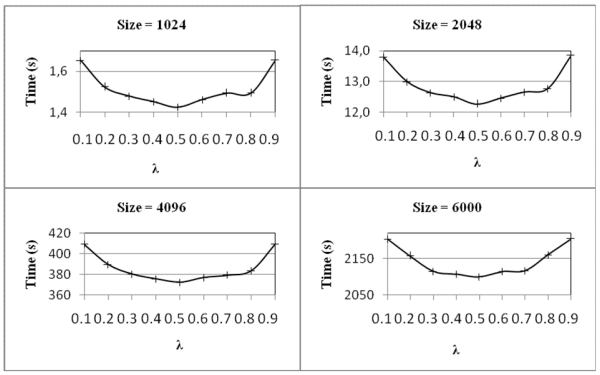

Fig 8. Execution time in terms of the decomposition factor λ for (static) RCA_λ*_k

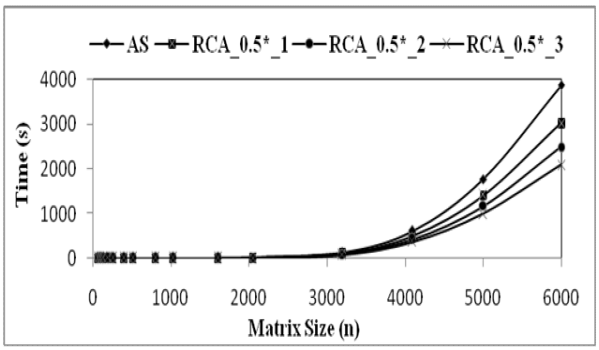

Fig 9. Execution time ext in terms of matrix size *n*

B. *Parallel Algorithms*

Two series of experiments were achieved. The first was done on a shared memory environment i.e. a Dell T5400 2.5 GHz quad core biprocessor. In order to have a flexible range of different speeds, we used a technique of changing clock frequencies of processors on Linux [15][16]. The second targeted a distributed platform constituted by 16 node cluster connected by heterogeneous network. C, OpenMp [17][18] and MPI [19][20] under Ubuntu 9.04 OS were used.

1)First Platform: Shared Memory Parallel Machine: We defined three scenarios with heterogeneous characteristics (frequencies are imposed) explained in Table III below.

TABLE III. SCENARIOS OF THE SHARED MEMORY ENVIRONMENT

	Scenario 1	***Scenario 2***	***Scenario 3***
Core 1	*2.5*	*2.5*	*2.5*
Core 2	*2.5*	*2.5*	*2.5*
Core 3	*2.5*	*2.5*	*2.0*
Core 4	*2.5*	*2.5*	*2.0*
Core 5	*1.7*	*1.2*	*1.7*
Core 6	*1.7*	*1.2*	*1.7*
Core 7	*1.7*	*1.2*	*1.2*
Core 8	*1.7*	*1.2*	*1.2*
Virtual Global Speed (vgs)	*16.8*	*14.8*	*14.8*

The performance evaluation consists in measuring the cost of the parallel algorithm $\mathcal{C}_p=(\sum_{i=1}^{p} v_i)T_p$. We then deduce the corresponding efficiency, denoted $E_p=W/\mathcal{C}_p$. Remark that for each n and p, successive runs were achieved and the mean run time is chosen. Notice that vi corresponds to the speed of P_i, T_p is the execution times of the parallel algorithm and W is the execution time of the sequential algorithm running on a processor with speed 1 (using the same technique of changing frequency explained above).

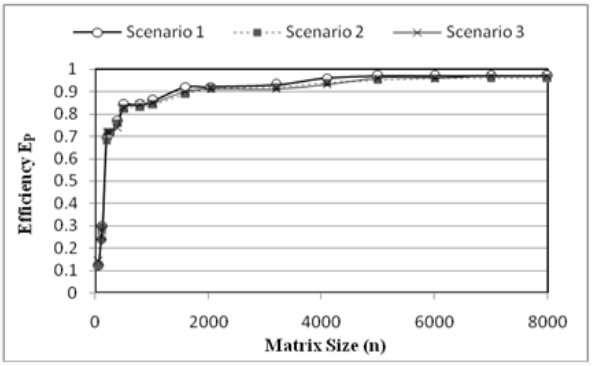

Fig 10. Efficiencies for the different scenarios in terms of matrix size *n* – Shared memory case

The analysis of the results confirming the theoretical study (see Fig 10), leads to the following remarks.

- For fixed scenario, E_p increases with n reaching 97% of the optimal value i.e.100%.
- For fixed n, the efficiencies of the three scenarios are very similar.
- The scalability of our algorithm is good for this type of architecture.

2)*Second Platform: Distributed Memory Cluster:* A preliminary work was necessary before the implementation of our algorithm. It is a measure of the real performance of each processor used (see Fig 11). Table IIIV reports the relative speeds of the heterogeneous processors in the cluster (We measure their relative speed with the core computation of the algorithm). Note that the relative speed does not depend on the size of problem for the wide range of matrix sizes used in our experiments [21].

TABLE IV. CLUSTER FEATURES

	Nodes		
Cluster	C_1: 6 Nodes	RAM	1 Go
		Frequency	2.5 GHz
		Relative speed	2.21
	C_2: 5 Nodes	RAM	1 Go
		Frequency	3 GHz
		Relative speed	3.39
	C_3: 5 Nodes	RAM	512 Mo
		Frequency	1 GHz
		Relative speed	1

We defined six scenarios with heterogeneous characteristics depicted in Table V below.

TABLE V. SCENARIOS OF THE DISTRIBUTED PLATFORM

	C_1	C_2	C_3	Virtual global speed (vgs)	Processors' number
Scenario 1	5	5	6	35	16
Scenario 2	5	5	0	20	10
Scenario 3	5	0	6	20	11
Scenario 4	0	5	6	30	11
Scenario 5	1	1	1	6.5	3
Scenario 6	2	2	2	13	6

As previously done, the efficiency *E(p,n)* is depicted (see Fig 12) for different scenarios. The analysis of the results leads to the following remarks.

- For fixed scenario, E_p increases with n reaching 93% of the optimal value i.e.100%.

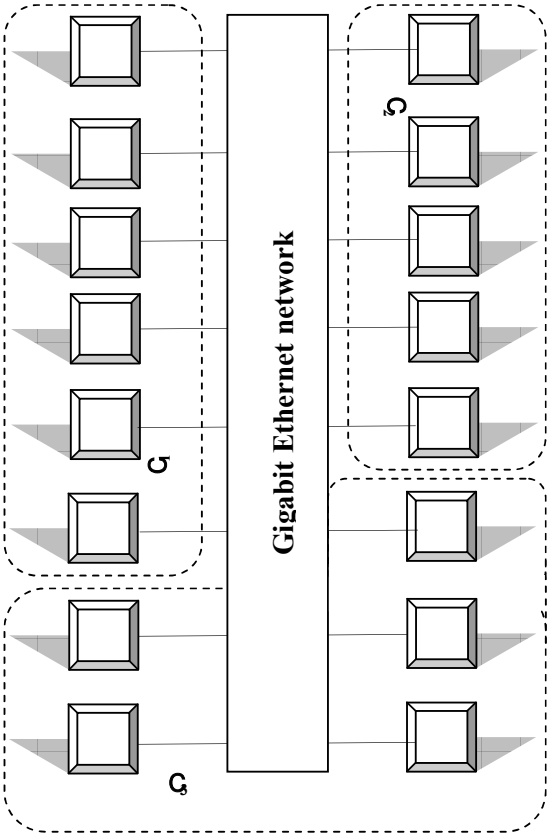

Fig 11. Cluster schema

- For fixed n, the efficiencies of the six scenarios are very similar.
- The scalability of our algorithm is good for this type of architecture.

Let us precise that an important performance criterion for a parallel algorithm is its scalability on a target architecture i.e. its ability to achieve performance proportional to the number of processors [19]. Our algorithm is indeed scalable since we obtained efficiencies around 95% and increasing with the number of processors as well as the matrix size.

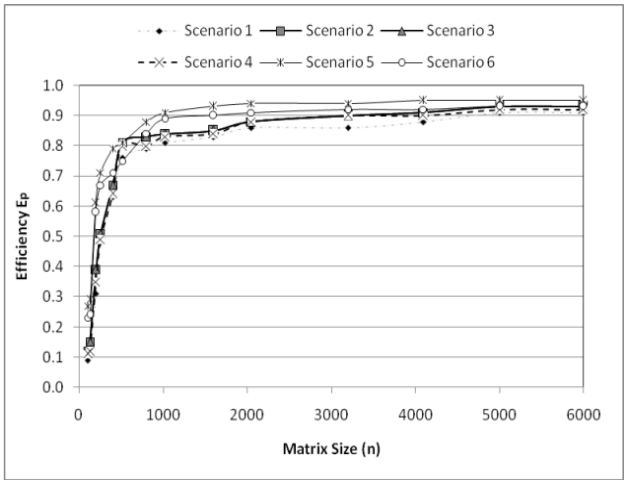

Fig 12. Efficiencies for the different scenarios in terms of matrix size n – Distributed memory case

III. CONCLUSION

We addressed in this paper the parallelization of an important kernel in scientific computing, namely triangular matrix inversion. Choosing a modified version of the recursive algorithm of Heller based on the 'Divide and Conquer' (D&C) paradigm, we made use of original versions of the latter in order to design optimal parallel communication-free algorithms for heterogeneous processors. The theoretical study was validated by a series of experiments achieved on two platforms: a shared memory one constituted by 8 cores and a distributed memory one involving 16 nodes. The results we obtained were interesting since we reached about 95% of the optimal speed-up and efficiency values for large matrix sizes. Furthermore, since we have shown that our algorithm is scalable, it may easily be adapted to larger parallel environ-ments, thus delivering high performances.

This leads us to precise some attracting perspectives we intend to study in the future. We may particularly cite the following points.

- Achieve an experimental study on larger platforms involving a higher number of (heterogeneous) processors.
- Achieve an experimental study on heterogeneous multicore CPU/GPU systems.
- Apply our non equitable and incomplete D&C approach on other algorithms in order to design optimal parallel algorithms where communications vanish or at least are minimised.

REFERENCES

[1] A. Quarteroni, R. Sacco and F. Saleri, "Méthodes numériques. Algorithmes, analyse et applications," *Springer*, Milano, 2007.
[2] D. Heller, "A survey of parallel algorithms in numerical linear algebra, " *SIAM Review 20*, pp. 740–777, 1978.
[3] J. J. Modi, "Parallel Algorithms and Matrix Computation," *Oxford Univ. Press*, Oxford, 1988.
[4] J. JáJá, *An Introduction to Parallel Algorithms*. Addison–Wesley, Reading, 1992.
[5] A. Schikarski, D.Wagner, "Efficient parallel matrix inversion on interconnection networks," *Journal of Parallel and Distributed Computing* 34, pp. 196–201, 1996.
[6] W. Nasri, Z. Mahjoub and D. Trystram, "Computing the inverse of a triangular matrix on heterogeneous clusters," *in Algorithms and Tools for Parallel Computing on Heterogeneous Clusters*, F. Desprez and al. ed., Nova Sc. Publ, pp. 67–78, 2007.
[7] W. Nasri, and Z. Mahjoub, "Design and implementation of a general parallel Divide and Conquer algorithm for triangular matrix inversion," *International Journal of Parallel and Distributed Systems and Networks* 5(1), pp. 35–42, 2002.
[8] L. Karlsson, "Computing explicit matrix inverses by recursion," MS thesis, Umea University, Department of Computing Science, Sweden, 2006.
[9] L. Keqin, "Fast and highly scalable parallel computations for fundamental matrix problems on distributed memory systems," *The Journal of Supercomputing*, http://www.springerlink.com/content/x03424q12666w3t4/fulltext.pdf, 2009.
[10] R. Mahfoudhi, Z. Mahjoub, and W. Nasri, "Une Nouvelle Méthode de Parallélisation Optimale pour l'Inversion de Matrice Triangulaire," *Renpar'20* ; Saint Malo, 2011.
[11] M. Gengler, S. Ubéda and F. Desprez, *Initiation au parallélisme: concepts, architectures et algorithmes*. Masson, Paris, 1996.
[12] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker and R. C. Whaley, "A proposal for a set of parallel basic linear algebra subprograms," TR CS– pp. 95–292, Computer Science Dept. University of Tennesse, Knoxville, TN, 1995.
[13] M. Marrakchi, "Conception et analyse d'ordonnancements efficaces pour algorithmes parallèles d'algèbre linéaire," Doctoral thesis, Faculty of Sciences of Tunis, 2001.
[14] N. Karmarkar, R. M. Karp, G.S. Luekerand A. M. Odlyzko, "Probabilistic Analysis of Optimum Partitioning," *J. Appl. Prob*. 23, pp. 626–645, 1986.
[15] Ubuntu Geek, http://www.ubuntugeek.com/howto–change–cpu–frequency–scaling–in–ubuntu.html (accessed Jan. 20, 2010).
[16] Ubuntu Guide, http://ubuntuguide.net/change–and–monitor–cpu–frequency–scaling–in–ubuntu–11–04–with–indicator–cpufreq (accessed Jan. 20, 2010).
[17] J. Chergui, "OpenMP : Parallélisation multitâches pour machines à mémoire partagée," Course, Institut du développement et des ressources en informatique scientifique, France, 2006.
[18] OpenMP, http://www.openmp.org (accessed Feb. 25, 2010).
[19] M. Creel and W. L. Goffe, "Multi–core CPUs, Clusters, and grid computing," *Kluwer Academic Publishers*, Dordrecht, 2007.
[20] Message Passing Interface Forum, http://www.mpi–forum.org
[21] A. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to parallel computing: design and analysis of algorithms*. Addison–Wesley, Reading, 1994.