International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014,pp.66-74
ISSN 2356-5608

Efficient Recursive Implementations for Linear Algebra Operations

Ryma Mahfoudhi[1], Samy Achour[2], Zaher Mahjoub[3]

University of Tunis El Manar, Faculty of Sciences of Tunis
University Campus - 2092 Manar II, Tunis, Tunisia
[1]rimahayet@yahoo.fr
[2]sami.achour@fst.rnu.tn
[3]zaher.mahjoub@fst.rnu.tn

Abstract— ***'Divide and Conquer' (D&C) is a famous paradigm for designing efficient algorithms and improving the effectiveness of computer memory hierarchies. Indeed, D&C-based matrix algorithms operate on submatrices or blocks, so that data loaded into the faster memory levels are reused. In this paper, we design recursive D&C algorithms for solving four basic linear algebra problems, namely matrix multiplication (MM), triangular matrix system solving, LU factorization, dense matrix system solving. Our solution is based on the use of matrix block decomposition and Strassen MM algorithm in the top decomposition level and BLAS routines invocation in the bottom decomposition level. The theoretical complexity of our algorithms is O($n^{log_2 7}$). In an experimental part, we compared our implementations with the equivalent kernels in the BLAS library. This latter study achieved on different machines permits to evaluate the practical interest of our contribution.***

Keywords— **BLAS, Block decomposition, Divide and Conquer, Linear algebra, Recursive implementation, Strassen algorithm.**

I. INTRODUCTION

The optimization of linear algebra routines has an important interest for both sequential and parallel applications. Indeed, due to their cubic complexity, the linear algebra routines such as matrix multiplication or system solving are too time consuming for large sized matrices.

Since the performance of these routines is memory hierarchy dependent, a solution for their optimization consists in using the divide and conquer (D&C) paradigm. In fact, with this technique, we divide the data into small portions which are loaded and reused by fastest levels of memory hierarchy.The Strassen method for matrix multiplication [1] is a typical divide and conquer algorithm.

Recursion leads in fact to automatic matrix blocking for dense linear algebra algorithms and the recursive way in algorithm programming accelerates data access. For this and other reasons, recursion usually speeds up the algorithms. Our work deals with the development of fast algorithms for solving triangular matrix systems, LU factorization and solving dense matrix systems. The main idea focuses on how recursion can be applied in order to benefit from recursive Strassen matrix multiplication algorithm.

The remainder of our paper is organised as follows. In Section 2, we recall the well-known Strassen algorithm for matrix multiplication, introduce our recursive blocked algorithms for solving a triangular matrix system then for LU factorization and present two algorithms for solving a dense matrix system. In Section 3, we discuss different implementation issues, including when to terminate the recursion (optimal level) and a describe comparative study with the BLAS routines.

II. RECURSIVE LINEAR ALGBRA ALGORITHMS

A. Matrix Multiplication (MM)

Let *A*, *B* and *C* be real matrices of size n. The number of scalar operations required for computing the matrix product $C=AB$ by the standard method is $2n^3=O(n^3)$. Due to its regularity and stability, this method is implemented in the BLAS library as ***dgemm*** routine.

In [1] Strassen introduced an algorithm for matrix multiplication, based on the *D&C* paradigm, whose complexity is only $O(n^{\log_2 7})$. This algorithm is based on the block decomposition of matrix *A*, *B* and *C*. Hence, to calculate the matrix product $C=AB$ of size *n*, we need 7 matrix products and 18 matrix additions of size *n/2*. Therefore, the complexity recurrence formula is as follows:

$Str(n) = 7Str(n/2) + 18ADD(n/2) + O(n^2)$. Solving this recurrence leads to $Str(n) = O(n^{log_2 7}) = O(n^{2.807})$. Hence an algorithm better than the standard one.

Since the seminal work of Strassen, a series of other works tried to design faster algorithms. We may particularly cite the Coppersmith-Winograd algorithm whose complexity is

$O(n^{2,376})$. However, this latter is significantly more complicated and less stable than Strassen's [2], [3].

B. Triangular Matrix System Solving (TMSS)

We now discuss the design of solvers for a triangular matrix system with matrix right hand side AX=B (resp. left hand side XA=B) where A (a triangular matrix) and B (a dense matrix) are known.

This kernel is commonly named ***trsm*** in the BLAS convention. In the following, we will consider, without loss of generality, the resolution of a lower triangular matrix system with matrix right hand side (AX=B). Our approach is based on a block recursive algorithm in order to reduce the computation to matrix multiplication (MM) [4], [5].

To optimize this algorithm, we use a fast algorithm for dense MM i.e. Strassen algorithm.

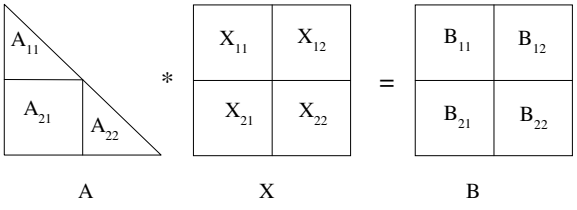

Fig. 1 Matrix Splitting for TMSS Algorithm

We split A, X and B as mentioned in figure 1. This recursive splitting is expressed as follows:

(1) $A_{11}X_{11} = B_{11}$ (3) $A_{21}X_{11} + A_{22}X_{21} = B_{21}$
(2) $A_{11}X_{12} = B_{12}$ (4) $A_{21}X_{12} + A_{22}X_{22} = B_{22}$

The procedure is recursively applied until reaching a size smaller than a fixed block size blks. Hence, solving a TMSS of size n requires 4 TMSS of size n/2 and 2 MM of size n/2. Thus, the resulting complexity recurrence formula is:

$$TMSS(n) = 4TMSS(n/2) + 2MM(n/2) + O(n^2)$$
$$= 4TSS(n/2) + O(n^{log_2 7}) = O(n^{log_2 7}).$$

So, we define for TMSS 2 levels (denoted (i,j), see figure 2) since there are 2 recursions.

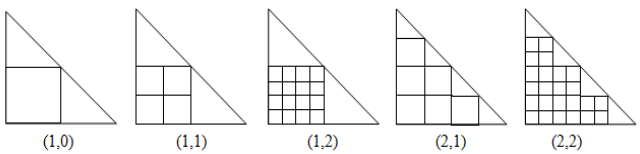

Fig. 2 Matrix Splitting for TMSS Algorithm- 2 levels

TMSS Algorithm

Begin
  If *(n<=blks)* ***Then***
    Trsm(A,X,B,n)
  Else */* split matrices into four blocks of sizes n/2*/*
    $X_{11} = TMSS(A_{11},B_{11})$
    $X_{12} = TMSS(A_{11},B_{12})$
    $X_{21} = TMSS(A_{22}, B_{21}\text{-}MM(A_{21},X_{11}))$
    $X_{22} = TMSS(A_{22}, B_{22}\text{-}MM(A_{21},X_{12}))$
  Endif
End

C. LU Factorization (LUF)

LU Factorization, called also LU decomposition, factorizes a matrix as the product of a lower triangular matrix (L) and an upper triangular one (U). It is generally used to solve square systems of linear equations, and is considered as a key step for matrix inversion or computing the determinant matrix. This kernel is commonly named ***getrf*** in the BLAS convention.

To reduce the complexity of LUF, blocked algorithms have been proposed since 1974 [6]. For a given matrix A of size n, the L and U factors verifying A=LU. After splitting A, L and U as presented in figure 3, we obtain the following equations:

(1) $L_1U_1 = A_{11}$ (3) $L_3U_1 = A_{21}$
(2) $L_1U_2 = A_{12}$ (4) $L_3U_2 + L_4U_4 = A_{22}$

Hence the LUF of matrix A of size n requires:

- One LUF of size n/2 i.e. (1): $L_1U_1 = A_{11}$ giving L_1 and U_1
- Solving 2 (lower) triangular matrix systems (TSS) i.e. (2): $L_1U_2 = A_{12}$ giving U_2 and $(3)^T$: $U_1^T L_3^T = A_{21}^T$ giving L_3
- One matrix multiplication (MM) i.e. L_3U_2
- One LUF of size n/2 i.e. (4): $L_4U_4 = A_{22} - L_3U_2$ giving L_4 and U_4.

Therefore, the complexity recurrence formula is as follows:

$$LUF(n) = 2LUF(n/2) + 2TMSS(n/2) + 1MM(n/2) + O(n^2)$$
$$= 2LUF(n/2) + O(n^{log_2 7}) = O(n^{log_2 7})$$

International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014,pp.66-74
ISSN 2356-5608

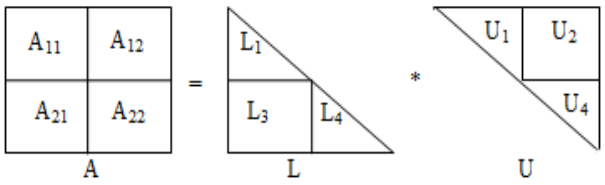

Fig. 3 Matrix Splitting for LUF Algorithm

LUF Algorithm

Begin
If *(n<=blks)* ***Then***
getrf (A,L,U,n)
Else */* split matrices into four blocks of sizes n/2*/*
$(L_1, [U_1, U_2]) = LUF([A_{11} A_{12}])$
${}^tU_1{}^tL_3 = A_{21}$
$H = A_{22} - L_3U_2$
$(L_4, U_4) = LUF(H)$
Endif
End

D. Dense Matrix System Solving (DMSS)

1) Brief Survey: Solving a linear system of equations is a basic kernel used in many scientific applications. Given its cubic complexity in terms of the matrix size, say n, several works addressed the design of practical efficient algorithms for this problem. Apart the standard Gaussian elimination (GE) algorithm, another algorithm namely LU factorization (LUF) with same complexity is often used due to its better stability. This algorithm is composed of two phases. The first consists in factorizing of the input matrix, say A, into a product of a lower triangular matrix L and an upper triangular one U i.e. A=LU. Afterwards, if Ax=b is the input system, where x and b are column vectors of size n and A is a square matrix of size n, we have to successively solve, in the second phase, two triangular systems i.e. Ly=b and Ux=y. We recall that the first phase costs $2n^2/3+O(n^2)$ and the second costs $2n^2+O(n)$. Thus an overall $2n^2/3+O(n^2)$ complexity [7].

Now, consider the matrix system (MS): AX=B where A, X and B are three dense square matrices of size n, A and B being known whereas X is unknown. Clearly, a straightforward approach for solving such a matrix system (MS) consists in solving n classical systems of size n. Obviously, this standard algorithm (SA) has a complexity $SA(n)=8n^3/3+ O(n^2)$ since we need only one factorization followed by solving n couples of triangular systems.

More precisely, solving the MS: AX=B by LUF requires one LUF i.e. A=LU, then solving two triangular matrix systems (TSS): LY=C and UX=Y i.e. 2 classical triangular systems of size n. Our aim is to optimize (through the D&C paradigm) both LUF and TSS kernels in order to obtain a fast algorithm for solving the MS.

2) Recursive Algorithm Using Blocked Decomposition: We introduce now another algorithm for solving the MS: AX=B. The main idea consists in decomposing both matrices A, X and B into 4 submatrices of size n/2 as shown in figure 4.

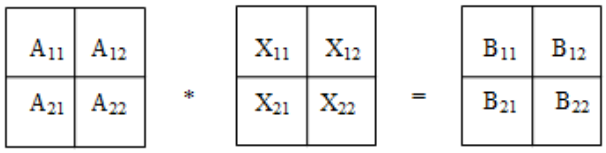

Fig. 4 Matrix Splitting for Recursive Blocked Algorithm

This splitting leads to the following equations:
(1) $X_{11} + A_{12}X_{21} = B_{11} \rightarrow X_{11} = A_{11}^{-1}(B_{11} - A_{12}X_{21})$
(2) $A_{11}X_{12} + A_{12}X_{22} = B_{12} \rightarrow X_{12} = A_{11}^{-1}(B_{12} - A_{12}X_{22})$

(3) $A_{21}X_{11} + A_{22}X_{21} = B_{21} \rightarrow A_{21}A_{11}^{-1}(B_{11} - A_{12}X_{21}) + A_{22}X_{21} = B_{21}$

(4) $A_{21}X_{12} + A_{22}X_{22} = B_{22} \rightarrow A_{21}A_{11}^{-1}(B_{12} - A_{12}X_{21}) + A_{22}X_{22} = B_{22}$

To ensure that the complexity of the Recursive Blocked (RB) algorithm does not exceed that of the standard algorithm (SA) i.e. $8n^3/3 +O(n^2)$, we must choose the most suitable kernels. We have shown in a previous paper [8] that we have to solve two dense matrix systems and six triangular systems of size n/2 along with five dense matrix multiplication of size n/2. So we get the following complexity recurrence formula:

$$RB(n) = 2RB(n/2)+ 6TMSS(n/2)+5MM(n/2) + O(n^2)$$
$$= 2RB(n/2) + O(n^{log_2 7}) = O(n^{log_2 7})$$

Clearly, if any MM algorithm of $O(n^{log_2 7})$ complexity is used, then the algorithms previously presented both have the same $O(n^{log_2 7})$ complexity instead of $O(n^3)$ for the corresponding standard algorithms.

To conclude our theoretical study on the four kernels, we have to precise that their $O(n^{\log_2 7})$ complexity requires that the recursive decomposition procedure (RDC) is repeated until reaching elementary problems of size O(1) [9], [10] However, this is never the case it practice since the RDC is usually stopped at a size equal to $n/2^r$ where $r < k = \log_2 n$ leading to the best execution time.

The parameter ***r*** will hence be considered as the optimal level in the RDC. On the other hand, if we define 1 optimal ***level*** for the MM algorithm, 2 optimal levels will be defined for TMSS since there are 2 recursions (1 for TMSS and 1 for MM called by TMSS), 4 for LUF since there are 4 recursions

International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014,pp.66-74
ISSN 2356-5608

(1 for LUF, 1 for MM and 2 for TMSS both called by LUF), 6 for the DMSS algorithm based on LU factorization (noted RDLUF) since there are 6 recursions (1 for RDLUF, 1 for MM and 4 LUF both called by RDLUF), and 4 for RB algorithm (1 for RB, 2 for TMSS and 1 for MM called by RB).

III. Experimental Study

This section presents experiments of our implementations for the different algorithms described above. We precise that we used BLAS library [11] in the last recursion level of any algorithm. Four machines (nodes) from the Grid'5000 were targeted [12] (see Table 1). We used the g++ compiler under a Linux Debian wheezy distribution. All execution times are the means of several runs.

We have to underline the importance of the determination, for each algorithm used, of the optimal number of recursive levels (***nrl***) i.e. the one leading to the best execution time. Indeed, the optimal ***nrl*** depends on both matrix size and target machine architecture and has to be determined experimentally. It is well known that the execution tie decreases for increasing ***nrl*** until a precise threshold, and then increases [13].

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 [1024 16384] and the input matrices involving real floating point elements were randomly generated. For sake of simplicity and without loss of generality, we chosed *N* as a power of 2. We recall that when this is not the case, there are techniques known in the literature proposing efficient strategies (e.g. padding, dynamic peeling) leading to the power-of-2 case without increasing the complexity order [14].

TABLE I
MACHINES' CHARACTERISTICS

	CPU	***RAM size***	***Cache size***
Node1	AMD Opteron@1.7GHz	47 GB	0.5 Mo
Node2	Intel Xeon@2.93GHz	23 GB	8 Mo
Node3	Intel Xeon@2.27GHz	23 GB	8 Mo
Node4	Intel Xeon@2.53GHz	15 GB	8 Mo

A. Matrix Multiplication

We compare in this section our recursive implementation for matrix multiplication (RIMM) and the BLAS routine ***dgemm*** for the same operation. We denote by Time and **Level** the minimum execution time (given in seconds) of Strassen Algorithm with the then level = *'**Level**'*. We also give the ratio i.e. execution time of ***dgemm***/execution time of RIMM.

TABLE III
MM: RIMM VS DGEMM

N	**dgemm (s)**	**RIMM**		**Ratio**
		Time (s)	**Level**	
	Node 1			
1024	2.75	1.51	2	1.82
2048	22.43	10.68	3	2.10
4096	178.93	76.64	4	2.33
8192	1449.63	545.02	5	2.66
16384	11553.63	3843.05	6	3.00
	Node 2			
1024	0.96	0.71	3	1.35
2048	7.82	5.02	4	1.55
4096	67.4	35.8	5	1.88
8192	534.66	253.92	6	2.03
16384	4430.28	1788.54	7	2.47
	Node 3			
1024	1.16	0.92	3	1.26
2048	9.62	6.45	4	1.49
4096	83.45	46.04	5	1.81
8192	664.81	326.27	6	2.03
16384	5548.39	2298.19	7	2.41
	Node 4			
1024	0.93	0.74	3	1.25
2048	7.76	5.31	4	1.46
4096	68.67	38.03	5	1.80
8192	547.67	270.04	6	2.02
16384	4579.54	1904.74	7	2.40

We can notice that for any target machine and any matrix size, RIMM is better than BLAS. The corresponding speed-ups increase with N. Indeed, for N=16384, RIMM is 3 times better than BLAS with node 1 and around 2.4 times better with the three other nodes. We have to add that the recursion is terminated when the size of remaining subproblems to be solved is smaller than the machine block size, which is the only architecture-dependent parameter in our algorithms.

B. Triangular Matrix System Solving

We compare in this section our recursive implementation for triangular matrix system solving (RITSS) and the ***dtrsm*** BLAS routine resolving the same problem. The two induced recursion levels are denoted '***Level(1,2)***' where the first is that of RITSS and the second is that of RIMM (called by RITSS for the resolution of the encountered MM).

International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014,pp.66-74
ISSN 2356-5608

TABLE IIIII
TRIANGULAR MATRIX SYSTEM SOLVING: RITSS VS DTRSM

N	Dtrsm(s)	RITSS		Ratio
		Time (s)	Level(1.2)	
	Node 1			
1024	1.22	0.91	(1,1)	1.34
2048	12.16	6.8	(2,2)	1.79
4096	104.06	50.56	(3,3)	2.06
8192	1202.44	373.71	(4,3)	3.21
16384	9672.9	2756.7	(4,4)	3.51
	Node 2			
1024	0.59	0.47	(2,1)	1.25
2048	5.05	3.34	(4,1)	1.51
4096	41.59	24.27	(4,2)	1.71
8192	332.92	174.29	(4,3)	1.91
16384	2827.28	1252.79	(4,4)	2.26
	Node 3			
1024	0.75	0.6	(2,1)	1.25
2048	6.4	4.3	(4,1)	1.49
4096	52.69	31.27	(4,2)	1.68
8192	420.77	224.51	(4,3)	1.87
16384	3591.94	1611.47	(5,4)	2.23
	Node 4			
1024	0.59	0.49	(2,1)	1.20
2048	5.06	3.44	(4,1)	1.47
4096	42.25	25.07	(4,2)	1.68
8192	337.79	180.57	(4,3)	1.87
16384	2907.67	1298.52	(4,4)	2.24

We can determinate for any node the level for which RITSS becomes better than ***dtrsm***. The corresponding Ratio increases with N. Indeed, for N=16384, RITSS is 3.5 times better than BLAS with node 1 and around 2.25 times better with the other machines.

C. LU Factorization

We compare in this section our recursive implementation for the LU factorization routine RILUF and the BLAS routine for LU factorization ***dgetrf***. We denote the four induced recursion levels by '***Level(1,2,3,4)***' where level 1 is the LUF recursion level, level 2 et 3 are those of RITSS (called by RILUF) and level 4 is the recursion level of RIMM (called by RILUF).

TABLE IVV
RILUF VS DGETRF

N	dgetrf (s)	RILUF		Ratio
		Time (s)	Level(1,2,3,4)	
	Node 1			
1024	0.63	0.6	(2,0,4,3)	1.05
2048	4.98	4.16	(3,0,4,3)	1.20
4096	40.25	31.58	(3,0,4,3)	1.27
8192	421.41	249.68	(4,3,3,4)	1.69
16384	5631.42	2202.88	(3,3,3,3)	2.56
	Node 2			
1024	0.3	0.28	(2,0,3,3)	1.07
2048	2.26	1.9	(4,0,3,3)	1.19
4096	19.06	14.23	(3,0,4,3)	1.34
8192	150.84	108.88	(3,0,4,4)	1.39
16384	1337.44	808.54	(4,1,4,4)	1.65
	Node 3			
1024	0.37	0.32	(4,0,0,0)	1.16
2048	2.87	2.4	(4,0,0,1)	1.20
4096	24.14	18.19	(3,0,3,3)	1.33
8192	192.68	138.92	(3,0,4,4)	1.39
16384	1803.46	1038.77	(4,1,3,4)	1.74
	Node 4			
1024	0.3	0.25	(4,0,1,0)	1.20
2048	2.27	1.9	(3,0,1,1)	1.19
4096	19.07	14.57	(3,0,4,3)	1.31
8192	152.55	111.3	(4,0,4,3)	1.37
16384	1338.21	844.33	(4,0,4,4)	1.58

We can determinate for any target machine the level for which RILUF becomes better than ***dgetrf***. The corresponding ratio increases with N. Indeed, for N=16384, RILUF is 2.5 times faster than BLAS with node 1 and around 1.7 better with the other machines. This shows the importance of determining a suitable level.

D. Dense Matrix System Solving

We named our routine RIDLUF and RIDRB (see section 2.4). The BLAS routine, where the routine ***dtrsm*** was used in combination with the factorization routine ***dgetrf*** to solve dense systems, is denoted ***dmss***. We precise that we denote by 'Ratio1' (resp.Ratio2) the ratio execution time of RIDLUF/***dmss*** (resp. RIDRB/***dmss***). For RIDLUF, the six induced recursion levels are denoted '***Level(1…6)***', where

International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014,pp.66-74
ISSN 2356-5608

levels 1…4 correspond to LUF (which is called by RIDLUF) and levels 5,6 correspond to RITSS.

TABLE V
SOLVING DENSE MATRIX SYSTEM: RIDLUF VS DTRSM

N	dmss (s)	RIDLUF		Ratio1
		Time (s)	Level(1…6)	
	Node 1			
1024	3.15	3.3	(2,0,2,2,3,3)	0.95
2048	29.44	19.9	(3,0,2,2,3,3)	1.48
4096	250	135.1	(3,1,3,3,3,3)	1.85
8192	3063.05	1029.05	(3,0,3,3,4,4)	2.98
16384	25546.01	7756.44	(4,1,3,4,4,4)	3.29
	Node 2			
1024	1.45	1.47	(2,0,2,2,3,3)	0.99
2048	12.37	9.13	(2,0,2,2,3,3)	1.35
4096	102.48	63.55	(3,0,3,3,3,3)	1.61
8192	850.81	459.83	(3,1,4,3,4,3)	1.85
16384	7017.24	3348.38	(4,0,4,4,4,4)	2.10
	Node 3			
1024	1.47	1.61	(3,0,2,2,3,3)	0.91
2048	12.5	9.77	(2,0,2,2,3,3)	1.28
4096	104.41	66.47	(3,0,3,3,3,3)	1.57
8192	849.34	482.97	(3,0,3,3,3,3)	1.76
16384	7310.57	3486.21	(3,1,3,3,4,4)	2.10
	Node 4			
1024	1.88	1.88	(3,0,2,2,3,3)	1.00
2048	15.77	11.75	(3,0,2,2,3,3)	1.34
4096	130.65	81.74	(3,0,3,3,3,3)	1.60
8192	1073.95	600.72	(3,0,3,3,4,4)	1.79
16384	9009.67	4371.1	(3,1,3,3,4,4)	2.06

We can determinate for any node the level for which RIDLUF becomes better than ***dmss***. The corresponding ratio increases with N. Indeed, for N=16384, RIDLUF is 3.29 times better than BLAS with node 1 and around 2 times better with the other nodes.

For RIDRB, we denote the four induced recursion levels by '***Level (1…4)***', where level 1 is the recursion levels of RIDRB, levels 2 and 3 the recursion levels of RITSS (which is called by RIDRB) and level 4 is the recursion level of RIMM.

We remark that for any target machine, there is a level when RIDRB becomes faster than ***dmss***. For a matrix of size 16384, RIDRB is 3.33 times better than BLAS with node 1 and around 2.2 times better with the other nodes. Furthermore, an important loss of performance is observed for BLAS when N increases.

TABLE VI
SOLVING DENSE MATRIX SYSTEM: RIDRB VS DMSS

N	dmss (s)	RIDRB		Ratio2
		Time (s)	Level(1…4)	
	Node 1			
1024	3.18	2.1	(1,2,1,1)	1.51
2048	29.57	16.09	(1,2,1,1)	1.84
4096	248.6	127.66	(2,2,2,2)	1.95
8192	3067.71	957.64	(2,3,3,3)	3.20
16384	25471.57	7653.75	(3,3,4,4)	3.33
	Node 2			
1024	1.49	1.03	(1,2,1,1)	1.45
2048	12.42	7.46	(1,3,1,1)	1.66
4096	102.77	58.13	(1,3,2,2)	1.77
8192	850.77	430.7	(2,3,3,3)	1.98
16384	7150.95	3217.06	(2,4,4,4)	2.22
	Node 3			
1024	1.48	1.06	(1,2,1,1)	1.40
2048	12.49	7.68	(1,2,1,1)	1.63
4096	104.52	59.55	(1,4,1,1)	1.76
8192	849.05	453.67	(2,2,3,3)	1.87
16384	7304.99	3278.79	(2,3,4,4)	2.23
	Node 4			
1024	1.87	1.32	(1,2,1,1)	1.42
2048	15.76	9.60	(1,3,1,1)	1.64
4096	130.42	74.67	(1,4,2,2)	1.75
8192	1072.29	553.33	(1,3,3,3)	1.94
16384	9013.12	4894.0	(2,2,3,3)	2.19

E. Remarks related to machine architecture

The experimental study achieved on different nodes enables us to make the following remarks as far as the node architecture is concerned.

1) Different behaviors between nodes: Figure 5 and figure 6 depict the variations of the Ratio in terms of matrix size for MM and TMSS. We can notice that the behaviours of the three Intel nodes (2, 3 and 4) are quite similar contrary to node 1. In fact, the designed algorithms are more efficient on this latter.

This phenomenon is due to the decreasing of BLAS performance for node 1. In fact, the optimization approach adopted by BLAS basically involves the optimal use of cache by simultaneously operating on several columns of a matrix. On machines with high speed and large cache memory, these operations can provide a significant speed advantage [15]. Since for the AMD processor (node 1), the cache memory size is 0.5 Mo, so the BLAS performance decreases and the ratio is

larger than that found for other nodes .i.e. for Intel processors having 8Mo of cache size (see figure 7).

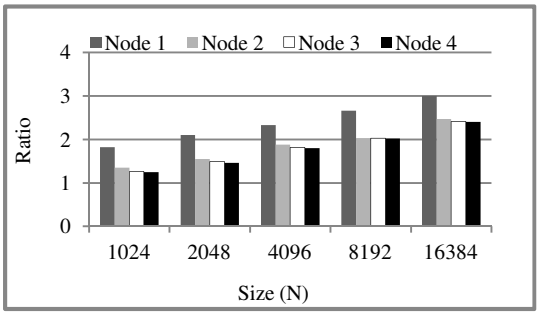

Fig. 5 Ratiovariations in terms of matrix size n – RIMM vs DGEMM

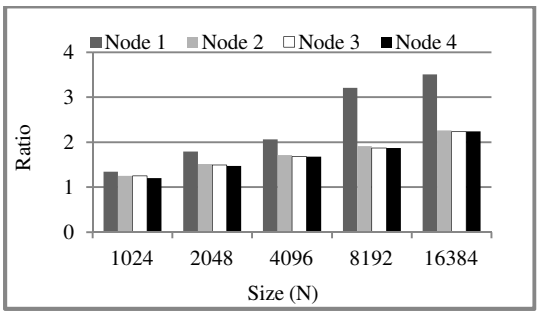

Fig. 6 Ratio variations in terms of matrix size n – RITSS vs DTRSM

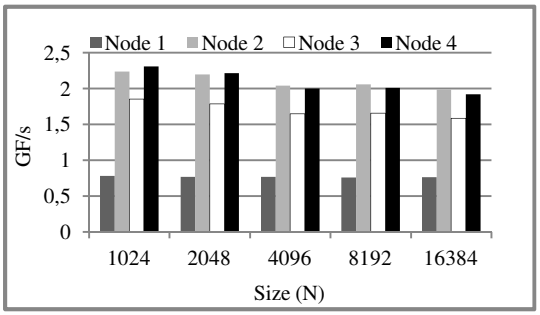

Fig. 7 Performance variation in terms of matrix size n – DGEMM

2) *RIDLUF vs RIDRB:* In Figure 8 the execution time ratio of RIRB/ RILUF is depicted. We can notice that RIDRB is more efficient than RIDLUF. For increasing matrix sizes, the two algorithms become very similar (improvement factor decreases from 1.57 for *N=1024* to 1.01 for *N=16384*).

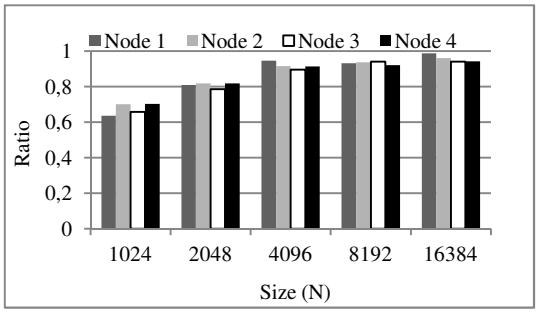

Fig. 8 Ratio variations in terms of matrix size n - RIDLUF vs RIDRB

IV. CONCLUSION AND FUTURE WORK

The fast recursive algorithms we designed for both matrix multiplication, LU factorization, triangular and dense matrix systems solving have been proven enough satisfactory in practice and could outperform some BLAS routines. These performances were tightly related to the target machines and the optimal number of recursion levels. Indeed, this occurs at a threshold reached when the remaining sub-problems to be solved are smaller than the optimal memory block size of the target machine. Pursuing recursion until a lower size would in general leads to an important overhead and a drop in the overall performance. It has to be noticed that our (recursive) algorithms essentially benefit from both (recursive) Strassen matrix multiplication algorithm, recursion and the use of BLAS routines in the last recursion level. Furthermore, the performance was reached, particularly thanks to (i) efficient reduction to matrix multiplication where we optimized the number of recursive decomposition levels and (ii) reusing numerical computing libraries as much as possible.

The results we obtained lead us to precise some attracting perspectives we intend to study in the future. We may particularly cite the following points:

- Design of specific heuristics for the determination of the multiple optimal recursion levels in the different discussed implementations.
- Generalize our approach to other linear algebra kernels such as rectangular matrix system solving and multiplication.

REFERENCES

[1] V. Strassen, "Gaussian elimination is not optimal", Numerische Mathematik, 1969, 13, pp. 354-356.
[2] D. H. Bailey , K. Lee , H. D. Simon, "Using Strassen's Algorithm to Accelerate the Solution of Linear Systems, " 1991, J. Supercomputing, 1991,(4), pp.357-371.
[3] J. Demmel, O. Holtz and R. Kleinberg, "Fast linear algebra is stable," Numerische Mathematik., 2007, 108(1), pp. 59-91.
[4] R. Mahfoudhi, "A fast triangular matrix inversion," Proceedings of the 2012 International Conference on Applied and Engineering Mathematics, ICAEM 2012, London, 2012

[5] R. Mahfoudhi, and Z. Mahjoub, "A fast recursive blocked algorithm for dense matrix inversion, " Proceedings of the 12th International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2012, La Manga, Spain, 2012.

[6] A. V. Aho, J. E. Hopcroft, and J.D. Ullman," The design and analysis of computer algorithms," Addison-Wesley, Reading, Mass, 1974.

[7] P. Lascaux and R. Théodor, "Analyse numérique matricielle appliquée à l'art de l'ingénieur, Tome 1, " Dunod, Paris, 2000.

[8] R. Mahfoudhi, and Z. Mahjoub, "On fast algorithms for matrix system solving, " Proc. International Conference on Control, Engineering & Information Technology, CMMSE 2012, CEIT'13, Sousse, Tunisia, 2013.

[9] F. Song, Sh. Moore, and J. Dongarra, "Experiments with Strassen's algorithm: from sequential to parallel, " International Conference on Parallel and Distributed Computing and Systems, PDCS06, Dallas, Texas,2006.

[10] V. Valsalam, and A. Skjellum, "A framework for high-performance matrix multiplication based on hierarchical abstractions, algorithms and optimized low-level kernels," Concurrency and Computation: Practice and Experience, 14 (10), pp.805-839, 2002.

[11] (2013) The BLAS website. [Online]. Available: www.netlib.org/blas/

[12] (2013) The Grid 5000 website. [Online] Available : https://www.grid5000.fr

[13] S. Huss-Lederman, E.M. Jacobson, J.R. Johnson, A. Tsao and T.Turnbull, "Strassen's algorithm for matrix multiplication: Modeling, analysis, and implementation," Technical Report, Center for Computing Sciences, Bowie, Maryland, 1996.

[14] M. Thottethodi, S. Chatterjee and A. R. Lebeck, "Tuning Strassen's matrix multiplication for memory efficiency," Supercomputing '98 Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, Orlando, Florida, 2012.

[15] (2013) Mathworks. [Online]. Available: http://www.mathworks.com/company/newsletters/articles/matlab-incorporates-lapack.hl