Performance Evaluation of Algorithms for Sparse-Dense Matrix Product

Olfa Hamdi-Larbi[†][‡], Elwsaef Rim[†]

[†]University of Tunis El Manar - Faculty of Sciences of Tunis, Tunis, Tunisia
[‡]Taibah University-College of Business Administration, Madinah, KSA

DOI: 10.29322/IJSRP.10.10.2020.p10647
http://dx.doi.org/10.29322/IJSRP.10.10.2020.p10647

Abstract- In this paper we address the sparse-dense matrix product (SDMP) problem where the first (resp. second) matrix is sparse (resp. dense). We start with an initial DO loop nest structured algorithm corresponding to the most used sparse matrix compressed formats i.e. DNS, CSR and COO. Then, we derive other versions by applying optimization techniques such as scalar replacement, loop invariant motion and loop unrolling. In addition, we use different compiler optimization options such as -O0, -O1, -O2, -O3, -O4, -Os and -Funroll-loops. We particularly focus on the GAXPY-Row body kernel where the matrices are accessed row-wise. A theoretical multi-fold performance study permits to establish accurate comparisons between the different versions. Our contribution is validated through a series of experiments achieved on a set of real matrices, having different sizes and densities on Grid5000 Intel Xeon Processors. Our aim is to detect the optimal version for each format as well as the best compression format giving the best performances for SDMP.

Index Terms- compression formats, loop unrolling technique, Intel Xeon architecture, algorithm optimization.

I. INTRODUCTION

Sparse linear algebra concerns the study of matrix algorithms processing large size sparse matrices. These latter are very frequent in real world applications covering diverse domains such as electromagnetism, semiconductors, image processing, networks, graphs, molecular dynamics, fluid dynamics, etc [1, 2, 3, 4, 5, 6, 7]. A matrix is called sparse if it has a large (resp. small) number of zero (resp. nonzero) elements [3, 4, 5, 6, 7]. The mostly used kernels in these applications are (i) Sparse Matrix-Vector Product (SMVP) [4,5,6,7,8], (ii) Sparse-Dense Matrix Product (SDMP) [2],[4], (iii) The symmetric case of the SDMP problem, i.e. Dense-Sparse Matrix Product (DSPM) where the first matrix is dense and the second is sparse [3], and (iv) Sparse Matrix Product (SMP) where both input matrices are sparse [5,6]. We address here the case of Sparse-Dense Matrix Product (SDMP) denoted C=A.B where A (resp. B) is sparse (resp. dense). On the other hand, we underline that processing large sparse matrices requires, for reasons of space-time complexity reduction, the use of storing formats. These latter may be either general i.e. adapted to any sparse structure e.g. DNS (DeNSe), CSR (Compressed Sparse Row), CSC (Compressed Sparse Column) and COO (COOrdinate), or special i.e convenient for a matrix structure as MSR (Modified Storage Row) for triangular structure, BND (BaND) for band structure, DIA (Diagonal) for diagonal structure [3,4,5,6,7], etc. Our aim here is (i) to determine the best optimization techniques for each version corresponding to a compressed format. (ii) to determine the best SCF for SDMP i.e. leading to the best performances. The remainder of the paper is organized as follows. In section 2, we present the related work. In section 3, we recall some useful concepts. Then in section 4, we present a theoretical study of the intra and inter algorithms optimization for the SDMP corresponding to the three chosen compression formats. Section 5 is devoted to an experimental study validating our theoretical contribution.

II. RELATED WORK

Many works are interested in solving the problem of Matrix Product such as the Sparse Matrix-Vector Product (SMVP) [4, 5, 6, 7, 8], the Sparse-Dense Matrix Product (SDMP) [2], [4], the symmetric case of the SDMP problem, i.e. the Dense-Sparse Matrix Product (DSPM) [3] and the Sparse Matrix Product (SMP) [5,6]. Few works in the state-of-the-art concern the SDMP optimization especially for the sequential version of the algorithm. Indeed, most of the works are dealing with the parallelization of the kernel.

In [10], the authors use two different workstations with a total of eight CPU cores: one Xeon workstation that holds two quad core processors (2,66 GHz), and a Barcelona test platform that holds two AMD Opteron 2347 processors (Barcelona, quad core, 1.9 GHz). We note that they use the respective block-oriented data structure to store sparse matrices [10]. However, our goal is to study the SDMP versions corresponding to different SCFs.

In [11], the authors study the SDMP parallelization. However, we are interested in the sequential version.
Note that the present work is a continuation of the works carried out and published in [5],[16]. The used techniques are inspired from the work on the SMVP optimization presented in [7,8].

III. GENERAL CONCEPTS

A. *Compression Formats for Sparse Matrices*

We recall that a matrix is called sparse (dense) if it has a large (resp. small) number of zero (resp. nonzero) elements [3, 4, 5, 6, 7]. Let NNZ be the number of nonzero elements. As previously mentioned, processing sparse matrices requires using special SCFs restricted to the nonzero elements. In this paper, we are especially interested in three among the most used storage formats namely DNS, CSR and COO. We underline that for storing a sparse matrix, say A of size N having NNZ nonzero elements, DNS corresponds to a 2D array where the whole N^2 elements are stored. However, CSR corresponds to data structure consisting of three arrays denoted AA, JA and IA, where AA [1...NNZ] is a real array for row-wise storing the NNZ nonzero elements of A. JA [1…NNZ] is an integer array to store the column position of the elements in AA, and finally, a pointer array IA [1…N+1], where the ith entry points to the beginning of the ith row in arrays AA and JA [4], [7]. As to the COO format, it also consists of three arrays, each of which is of size NNZ i.e. a real array AA containing the nonzero elements, an array IA (resp. JA) of integers containing their row (resp. column) indices [4].

B. *Sparse-Dense Matrix Product (SDMP)*

The standard algorithm for multiplying two-dimensional N*N matrices, is given by Algorithm 1 below:

```
Algo-IJK (A, B, C, N)
DO i=1, N
    DO j=1, N
        DO k= 1, N
            C (i,j)+= A(i,k)* B(k,j)
        ENDDO
    ENDDO
ENDDO
END Algo-IJK
```

Algorithm 1. Standard Matrix-Matrix Product (MMP)

```
Algo-IKJ (A, B, C, N)
DO i=1, N
    DO k=1, N
        DO j= 1, N
            C (i,j)+= A(i,k)* B(k,j)
        ENDDO
    ENDDO
ENDDO
END Algo-IKJ
```

Algorithm 2. Standard algorithm GAXPY-R kernel

Let us recall that we address in this paper the case of the Sparse-Dense Matrix Product (SDMP) denoted C=A.B where the first (resp. second) matrix is sparse (resp. dense). We point out that the standard algorithm where A, B and C are stored in 2D matrices is a perfect 3-loop nest, denoted IJK (see Algorithm 1). It corresponds to the DOT-R body kernel and has a cubic complexity as mentioned above.

We can derive from algorithm 1 five other versions by permutation of the loops i, k and j. Hence, the permutations IKJ, JKI, KIJ, KJI, IJK and JIK respectively correspond to GAXPYs (GAXPY-R and GAXPY-C), AXPYs (AXPY-R and AXPY-C) and DOTs (DOT-R and DOT-C) kernels. We recall that there are two variants (Row, denoted R, and Column, denoted C) of each body kernel depending on the access mode to matrices A, B and C. We rely on the work of Zouaoui [4], who studied the different kernels and she found that the GAXPY-Row (GAXPY-R) kernel where all the matrices are accessed raw-wise is the best among the other kernels since it provides the best performances, improves data locality and reduces cache misses. In Algorithm 2, we present the structure of the GAXPY-R Kernel.

IV. A THEORETICAL STUDY OF SDMP

A. *Application of optimization techniques*

In order to optimize the SDMP loop nest structured algorithms, we have applied particular techniques such as (i) scalar replacement (SR) i.e. an array element (indirect memory access) is replaced by a scalar, (ii) loop invariant motion (LIM) where useless operations are avoided, and (iii) loop unrolling (LpU) by duplicating the loop body u times where u is an integer named LpU factor [6, 7]. In addition, we use different compiler optimization options denoted by -Oi (i=0 4, s) and Funroll-loops [7], [12].

B. Intra- algorithm study

In this paper, we are limited to formats storing matrices row-wise such as DNS, CSR and COO since nonzero elements may be stored and accessed row-wise. Various optimization techniques are applied to the obtained codes corresponding to these formats, namely, SR, LIM techniques (Algorithms 3, 6 and 9). In addition, in (Algorithms 4, 7 and 10) we apply the LpU technique.

For each studied format, we achieve a comparative study between different optimized versions for the SDMP. To make a comparative study in order to detect the most optimal optimization techniques, we consider (i) the number of operations (i.e. arithmetic and logic), (ii) the number of used memory words (i.e. integers and doubles), (iii) the number of indirect access (i.e. access to an array element) (iv) the number of indirect nested access (i.e. where the index of an array element is an array element) and (v) the number of reads and writes, as comparison criteria for the different SDMP versions (Tables 1, 2 and 3). Notice that unrolling optimizations have no impact on the values of these criteria. However, they have influence on the memory cache behaviour [6, 7].

1) SDMP-DNS versions: in Table 1, we compare the optimized version V2 (using tests, SR and LIM) to the non-optimized version (V0 version) of the SDMP-DNS algorithm.

SDMP_DNS version	#operations	#memory words	#Indirect Access	#Indirect nested Access	#Memory I/O
Non optimized (V0)	$2N^3$	$2N^2$	$4N^3$	0	$4N^3$
optimized (V2)	$2N*NNZ+ N^2$	$2N^2$	$N^2+3N*NNZ$	0	$2N^2 +3N*NNZ$

Table 1. Comparative study of SDMP-DNS versions

```
SDMP-DNS-V2 (A, B, C, N)
DO i=1, N
    DO k=1, N
        s = A(i,k)
        IF (s ≠ 0) THEN
            DO j= 1, N
                C(i,j)+ = s * B(k,j)
            ENDDO
        ENDIF
    ENDDO
ENDDO
END SDMP-DNS-V2
```

Algorithm 3. SDMP-DNS version (V2) optimized with logical tests, SR and LIM

```
SDMP-DNS-u2 (A,B,C,N)
m = N mod 2, ne = N-m
DO i=1, N
    DO k=1, N
        s = A(i,k)
        IF ( s ≠ 0) THEN
            DO j= 1, ne,2
                C(i,j)+ = s * B(k,j)
                C(i,(j+1))+= s * B(k,(j+1))
            ENDDO
            DO j = ne+1, N
                C(i,j)+= s * B(k,j)
            ENDDO
        ENDIF
    ENDDO
ENDDO
END SDMP-DNS-u2
```

Algorithm 4. SDMP-DNS optimized with logical tests, SR, LIM and LpU, factor u=2

2) SDMP-CSR versions: in Table 2, we present a comparative study for the SDMP-CSR algorithms (the non-optimized version V0 and the optimized version V2 (using tests, SR and LIM)).

SDMP_CSR version	#operations	#memory words	#Indirect Access	#Indirect Nested Access	#Memory I/O
Non optimized (V0)	2N*NNZ	$2NNZ+N+N^2$	3N*NNZ	N*NNZ	4N*NNZ
Optimized (V2)	2N*NNZ	$2NNZ+N+N^2$	2N+2NNZ+3N*NNZ	0	4N+4NNZ+3N*NNZ

Table 2. Comparative study of SDMP-CSR versions

```
SDMP-CSR-V0 (IA, JA, A, B, C, N)
DO i=1, N
    DO k=IA(i), IA(i+1)-1
        DO j= 1, N
            C(i,j)+= A(k) *B(JA(k),j))
        ENDDO
    ENDDO
ENDDO
END SDMP-CSR-V0
```

Algorithm 5. SDMP-CSR version (V0) non optimized

```
SDMP-CSR-V2 (IA, JA, A, B, C, N)
DO i=1, N
    ia1=IA(i), ia2= IA(i+1)
    DO k= ia1, ia2-1
        s= JA(k), s1= A(k)
        DO j= 1, N
            C(i,j)+= s1 *B(s,j)
        ENDDO
    ENDDO
ENDDO
END SDMP-CSR-V2
```

Algorithm 6. SDMP-CSR version (V2) optimized with SR and LIM

```
SDMP-CSR-u2 (IA, JA, A, B, C, N)
m=N mod 2, ne=N-m
DO i=1, N
    ia1=IA(i), ia2= IA(i+1)
    DO k= ia1, ia2-1
        s= JA(k), s1= A(k)
        DO j= 1, ne-1, 2
            C(i,j)+= s1 *B(s,j)
            C(i,(j+1))+= s1 * B(s,(j+1)))
        ENDDO
        DO j= ne+1, N
            C(i,j)+= s1 *B(s,j)
        ENDDO
    ENDDO
ENDDO
END SDMP-CSR-u2
```

Algorithm 7. SDMP-CSR optimized with SR, LIM and LpU, factor u=2

3) SDMP-COO versions: in Table 3 we present a comparative study for the SDMP-COO algorithms (Algorithm 8 and Algorithm 9).

SDMP_COO version	**#operations**	**#memory words**	**#Indirect Access**	**#Indirect Nested Access**	**#Memory I/O**
Non optimized (V0)	2N*NNZ	3NNZ+N^2	N*NNZ	3N*NNZ	4N*NNZ
optimized (V2)	2N*NNZ	3NNZ+N^2	3NNZ+3N*NNZ	0	6NNZ +3N*NNZ

Table 3. Comparative study of SDMP-COO versions

```
SDMP-COO-V0 (IA, JA, A, B, C, N)
DO i=1, NNZ
    DO j=1, N
        C(IA(i),j)+= A(i)* B(JA(i),j)
    ENDDO
ENDDO
END SDMP-COO-V0
```

Algorithm 8. SDMP-COO version (V0) non optimized

```
SDMP-COO-V2 (IA, JA, A, B, C, N)
DO i=1, NNZ
    iai= IA(i), jai=JA(i), s=A(i)
    DO j=1, N
        C(iai,j)+= s * B(jai,j))
    ENDDO
ENDDO
END SDMP-COO-V2
```

Algorithm 9. SDMP-COO version (V2) optimized with SR and LIM

```
SDMP-COO-u2 (IA, JA, A, B, C, N)
m=N mod 2, ne=N-m
DO i=1, NNZ
    iai= IA(i), jai=JA(i), s=A(i)
    DO j=1, ne-1, 2
        C(iai,j)+= s * B(jai,j))
        C(iai,(j+1))+= s * B(jai,(j+1))
    ENDDO
    DO j= ne+1, N
        C(iai,j)+= s * B(jai,j))
    ENDDO
ENDDO
END SDMP-COO-u2
```

Algorithm 10. SDMP-COO optimized with SR, LIM and LpU, factor u=2

C. Inter-algorithm study

In this section, we compare the optimized SDMP algorithms corresponding to the formats DNS, CSR and COO in order to detect the optimal format which provides the best performances. Notice that in Table 4, we don't present the nested indirect access number because they are equal to zero for the SDMP optimized algorithms.

Algorithm version	#operations	#memory words	# Indirect Access	# Memory I/O
SDMP_DNS	$2N*NNZ + N^2$	$2N^2$	$N^2+3*NNZ$	$2N^2+3N*NNZ$
SDMP_CSR	$2N*NNZ$	$2NNZ+N+N^2$	$2N+2NNZ+3N*NNZ$	$4N+4NNZ+3N*NNZ$
SDMP_COO	$2N*NNZ$	$3NNZ+N^2$	$3NNZ*(N+1)$	$6NNZ+3N*NNZ$

Table 4. Optimized SDMP versions corresponding to different formats

Thus, we first compare the number of operations. In the case of equality, we compare the number of indirect access and the number of memory I/O. Indeed, we remark that the SDMP-DNS has the largest number of operations. So, it is lastly classified. Concerning SDMP-CSR and SDMP-COO, they have the same number of operations. So, we compare their numbers of indirect access. Hence, we compute the difference of the indirect access numbers (resp. memory I/O numbers). Then, we study the sign of the obtained quantity which leads to three cases: 2N < NNZ, 2N > NNZ and 2N=NNZ. Tables 5, 6 and 7 present the classification of the SDMP versions corresponding to DNS, CSR and COO formats.

Algorithm version	#operations	#memory words	# Indirect Access		# Memory I/O	Final rank
			Reading	Writing		
SDMP_DNS	**3**	**3**	**3**	**1**	**3**	**3**
SDMP_CSR	**1**	**1**	**1**	**1**	**1**	**1**
SDMP_COO	**1**	**1**	**2**	**1**	**2**	**2**

Table 5. Comparative SDMP study for 2N > NNZ

Algorithm version	**#operations**	**#memory words**	**# Indirect Access**		**# Memory I/O**	**Final rank**
			Reading	Writing		
SDMP_DNS	**3**	**3**	**3**	**1**	**3**	**3**
SDMP_CSR	**1**	**1**	**2**	**1**	**2**	**2**
SDMP_COO	**1**	**1**	**1**	**1**	**1**	**1**

Table 6. Comparative SDMP study for 2N < NNZ

Algorithm version	**#operations**	**#memory words**	**# Indirect Access**		**# Memory I/O**	**Final rank**
			Reading	Writing		
SDMP_DNS	**3**	**3**	**3**	**1**	**3**	**3**
SDMP_CSR	**1**	**1**	**1**	**1**	**1**	**1**
SDMP_COO	**1**	**1**	**1**	**1**	**1**	**1**

Table 7. Comparative SDMP study for 2N=NNZ

We notice that:

- **In the case of 2N > NNZ**; SDMP-CSR and SDMP-COO have the same number of operations. However, SDMP-COO accomplishes more indirect access and memory I/O. Thus, SDMP-CSR is first classified, and SDMP-COO is second classified. the SDMP-DNS algorithm is lastly classified because it provides the worst theoretical performances.
- **In the case of 2N < NNZ**; SDMP-CSR and the SDMP-COO have the same number of operations. However, SDMP-CSR accomplishes more indirect access and memory I/O. Thus, SDMP-COO is first classified, and SDMP-CSR is second classified. The SDMP-DNS algorithm is lastly classified because it provides the worst theoretical performances
- **In the case of 2N=NNZ**; SDMP-CSR and SDMP-COO algorithms are first classified because they have the same number of operations, the same number of memory access and the same number of memory I/O. The SDMP-DNS algorithm is lastly classified because it provides the worst theoretical performances.

V. EXPERIMENTAL WORK

Architectural Models	**CPU frequency**	**L3 Cache memory**	**RAM**	**Sites**	**Clusters**
Intel Xeon E5-2660	2.20GHz	20Mo	**64 GB**	Nantes	Econome
Intel Xeon E5-2650	2.00GHz	20Mo	**252 GB**	Nancy	Graphite
Intel Xeon X5570	**2.93GHz**	8Mo	24GB	Rennes	Parapide
Intel Xeon E5520	**2.27GHz**	8Mo	24GB	Grenoble	Edel
Intel Xeon E5520	2.27 GHZ	**8Mo**	32GB	Sophia	Suno
Intel Xeon E5-2630	2.30 GHZ	**15Mo**	32GB	Lyon	Orion

Table 8. The characteristics of the used INTEL XEON models

In order to evaluate the performances of the SDMP versions, a series of experimentations is accomplished on Intel Xeon processors. These last belong to Grid5000 platform and have different architectures (see Table 8). To evaluate the performances of the algorithm versions and validate our theoretical study, we use three storage formats for storing matrix A. For each format we generate eight algorithm versions relative to the different optimization techniques. In total, the number of tested algorithms is 24. For each version, we apply seven different compiler options, we test ten matrices and we use six different target machines with different characteristics to achieve the experimentations. The total number of tests is equal to 15075.

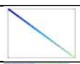
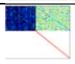
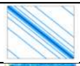
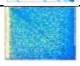
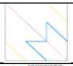
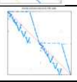
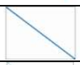
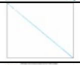
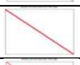
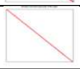
	Matrices	**N**	**NNZ**	**d**	**Structures**
2N<NNZ	cry10000	10000	49 699	0.05	
	FA	10617	72 176	0.06	
	Trefethen_20000	20000	287 233	0.07	
	mult_dcop_03	25187	193 216	0.03	
	bloweybl	30003	70 001	0.008	
	lhr34	35152	764 014	0.06	
	obstclae	40000	118 804	0.007	
2N>NNZ	bcsstm25	15439	15 439	0.006	
	qpband	20000	30 000	0.007	
	bcsstm37	25503	14 765	0.002	

Table 9. The characteristics of the real matrices

For each compressed format (SCF) (i.e. CSR, COO and DNS) used to store the sparse matrix A, we test different versions (i) SCF V0, the non-optimized version, (ii) SCF V1, the optimized version with scalar replacement (SR), (iii) SCF V2, the optimized version with scalar replacement (SR) and loop invariant motion (LIM) (iii) SCF ui (i=2, 8, 16, 32,36,40), the version SCF V2 optimized using the loop unrolling (LpU) such that ui correspond to the unrolling factors. In the same way, we include logical tests in the SDMP-DNS algorithm which improves its performances. In addition to the manual optimizations (i.e. SR, LM, and LpU), we use at compile time different options that control various hinds of optimizations in the GNU Compiler Collection (GCC) such as the '-O' options (the allowed forms are -O0, '-O1', '-O2', '-O3', and '-Os'), and the "-Funroll-loops ".

For this purpose, we use a set of matrices from real applications which belong to Tim Davis and Matrix Market collections [13][14]. The size (N) of the matrices is in the range [10000, 40000] and the density (%) is varying in ([8.10^{-3}, 7.10^{-2}]) (see Table 9).

A. Intra- algorithm study

Let Ratio be defined as:

$$\text{Ratio} = (1\text{-tx/ty}) * 100 \qquad (1)$$

Where tx is the runtime of the SDMP version x, ty is the runtime of the SDMP version y, with tx < = ty. In this section, we validate the optimization study of the SDMP versions corresponding to the DNS, CSR and COO formats. To optimize the SDMP-DNS, SDMP-CSR and SDMP-COO algorithms, we include logical tests, SR, LIM, and LpU. The experiments are accomplished on six different machines of Grid5000, using ten matrices from real applications. For each format, we present the experimentation results we obtained when applying (i) SR and LIM optimizations, (ii) loop unrolling technique and (iii) compiler optimization options.

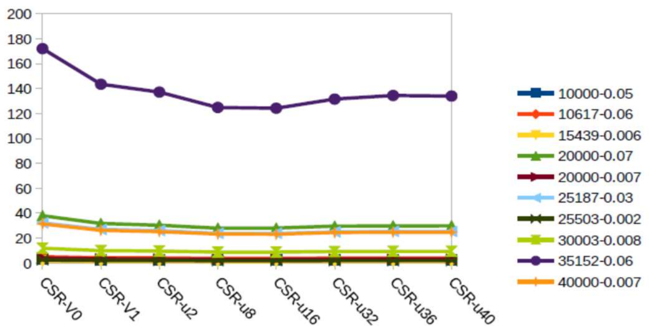
1) SR and LIM techniques: SR and LIM techniques improve the performance of the different algorithms SDMP-DNS, SDMP-CSR and SDMP-COO and this is true for all the tested matrices and on the six used architectures (see Figure 1.(a) and Figure 1.(b)). In the figures, each curve corresponds to the running time for a couple (matrix size/matrix density).

2) Loop unrolling technique: we apply the LpU technique with different factors in the range [2...40] and we obtain interesting results. Indeed, the loop unrolling technique when combined with the SR and LIM techniques increases the performances for SDMP-DNS, SDMP-CSR and SDMP-COO. We precise that the best performances are given by the unrolling factors u8, u16 and u32 for the three formats. This result is true on the six tested architectures. For lack of space reasons, we only present the results related to three architectures (Grenoble Edel, Nantes Econome and Lyon Orion). For the mentioned unrolling factors:

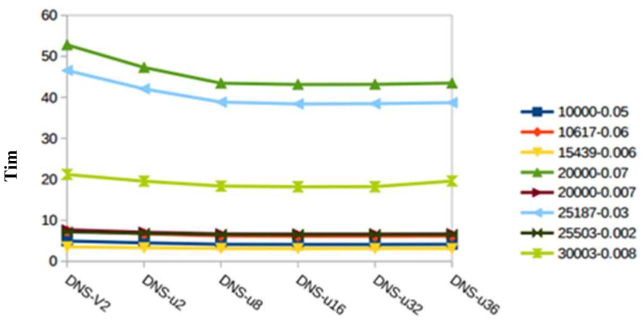
(i) for SDMP-DNS, the performances obtained on the different machines are equivalent. Indeed, the difference does not exceed a ratio of 1% on the Grenoble Edel processor and 6% on both Nantes Econome and Lyon Orion (see Figure 1.(c)).

(ii) for SDMP-CSR, the performances obtained on the different machines are also equivalent. The difference does not exceed a ratio of 3% on the Grenoble Edel processor, 6% on Nantes Econome and 5% on Lyon Orion (see Figure 1.(d)).

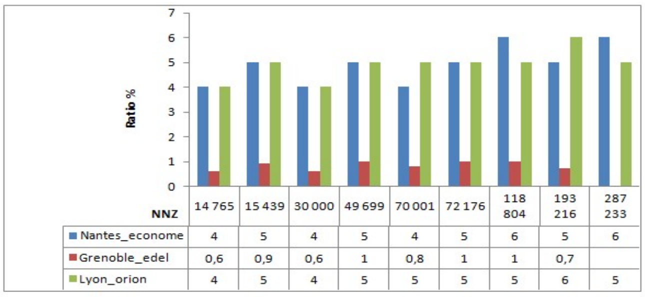
(iii) for SDMP-COO, the performances obtained on the different machines are also equivalent. The difference does not exceed a ratio of 6% on Grenoble Edel processor, 10% on Nantes Econome and 11% on Lyon Orion (see Figure 1.(e)).



(a). **Performance study of SDMP_CSR optimized with (SR, LIM and LpU) on Grenoble_Edel**

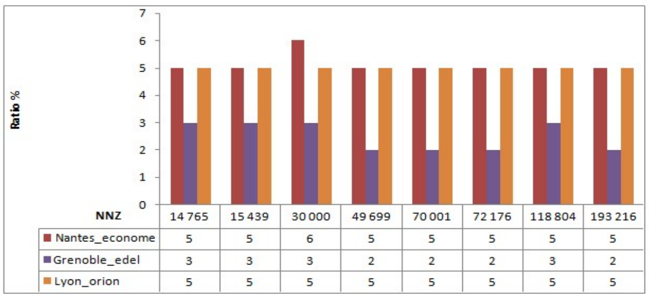


(b). **Performance study of SDMP_DNS optimized with (Test, SR, LIM and LpU) on Grenoble_Edel**



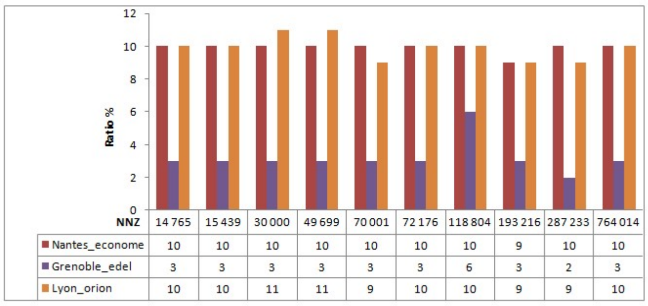
NNZ	14 765	15 439	30 000	49 699	70 001	72 176	118 804	193 216	287 233
Nantes_econome	4	5	4	5	4	5	6	5	6
Grenoble_edel	0,6	0,9	0,6	1	0,8	1	1	0,7	
Lyon_orion	4	5	4	5	5	5	5	6	5

(c). **Comparison for SDMP_DNS of the performances obtained
by using unrolling factors (u8 ... u32)**



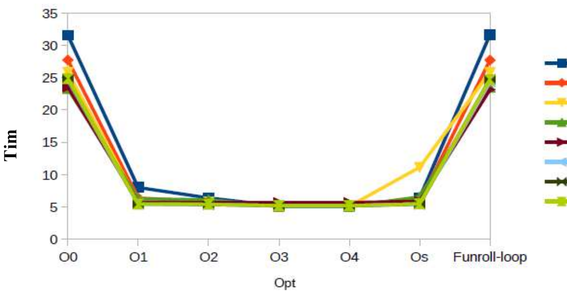
NNZ	14 765	15 439	30 000	49 699	70 001	72 176	118 804	193 216
Nantes_econome	5	5	6	5	5	5	5	5
Grenoble_edel	3	3	3	2	2	2	3	2
Lyon_orion	5	5	5	5	5	5	5	5

(d). **Comparison for SDMP_CSR of the performances obtained
by using unrolling factors (u8 ... u32)**



NNZ	14 765	15 439	30 000	49 699	70 001	72 176	118 804	193 216	287 233	764 014
Nantes_econome	10	10	10	10	10	10	10	9	10	10
Grenoble_edel	3	3	3	3	3	3	6	3	2	3
Lyon_orion	10	10	11	11	9	10	10	9	9	10

(e). Comparison for SDMP_ COO of the performances obtained by using unrolling factors (u8 ... u32)

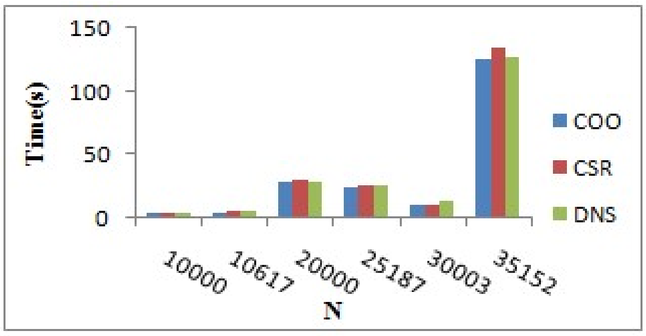


(f). Performance study of SDMP_ CSR optimized using compiler options where N=10000 on Nantes_Econome

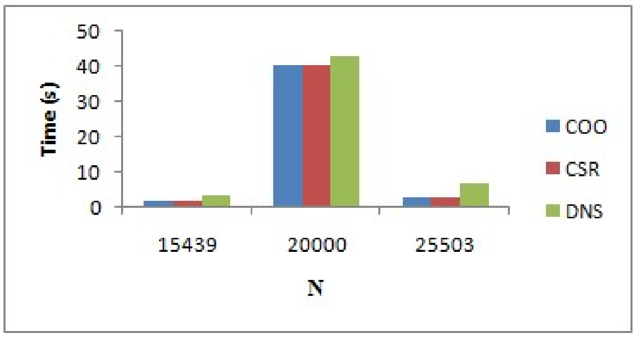
Figure 1: Experimental results for intra-algorithm comparison

Recapitulation: we conclude that SDMP optimization using techniques RS, LIM, tests and LpU improves the algorithm performances with a ratio reaching 22%. These results are true for all the formats DNS, CSR and COO used for storing matrix A and are valid for the six architectures.
For the three studied formats, the best performance of the SDMP is obtained when we combine manual optimizations and compiler options.

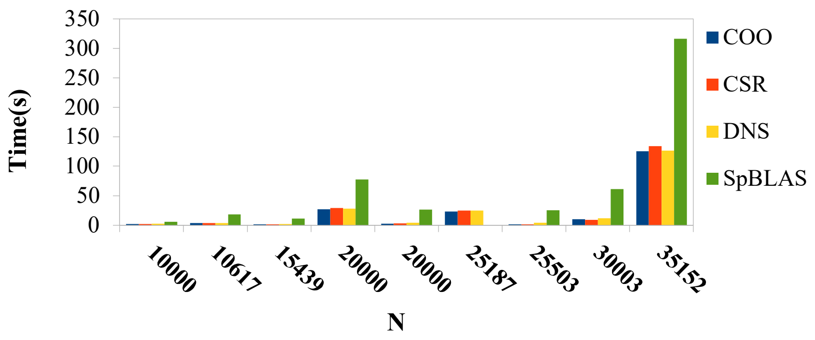
B. *Inter-algorithms study*



(a). A comparison of optimized SDMP-DNS, SDMP-COO and SDMP-CSR when 2N < NNZ on Lyon_Orion architecture



(b). A comparison of optimized SDMP-DNS, SDMP-COO and SDMP-CSR when 2N > NNZ on Grenoble_Edel architecture



(c). A comparison of the performances of SDMP optimized with LpU to the performances of Sparse BLAS on Lyon_Orion architecture

Figure 2: Experimental results for inter-algorithms comparison

The experimental results validate the theoretical results. Indeed,

- **In the case of 2N > NNZ**; the SDMP-DNS algorithm is the last classified. This is true on different tested architectures. On Grenoble Edel, Rennes Parapide and Sophia Suno, the SDMP-CSR algorithm is first classified followed by SDMP-COO algorithm (see Figure 2.(b)) which is conform with the theoretical study. However, on Nantes Econome, Nancy Graphite and Lyon Orion, the SDMP-COO algorithm is first classified followed by SDMP-CSR algorithm. Nevertheless, on the six different architectures, both algorithms have close performances.
- **In the case of 2N < NNZ**; the SDMP-DNS algorithm is the last classified. This is true on the different tested architectures. On Nantes Econome, Nancy graphite and Lyon Orion, the SDMP-COO algorithm is first classified followed by the SDMP-CSR algorithm (see Figure 2.(a)). However, on Grenoble Edel, Rennes Parapide and Sophia Suno, the SDMP-CSR algorithm is first classified followed by SDMP-COO algorithm. Nevertheless, on the six different architectures, both algorithms have close performances.

- **In the case of 2N=NNZ**; we don't find real matrices that respect this case.

In addition to manual optimizations, we use compiler optimization options, we notice that SDMP-COO and SDMP-CSR optimized with option O1 or O2 give the best running time on the six different architectures.
The obtained results are compared to the Sparse BLAS library, and we find that our optimized algorithms outperform the Sparse BLAS with a ratio of 13% (see Figure 2.(d)).

Recapitulation: When we compile SDMP-DNS, SDMP-COO and SDMP-CSR algorithms with compiler options O1 or O2, performances are improved, and the versions order is maintained.

VI. CONCLUSION

In this paper, we study several algorithm versions of sparse-dense matrix product (SDMP) corresponding to three storage formats (DNS, CSR and COO) using particularly the GAXPY-R body kernel. Various optimization techniques are applied and lead to interesting improvement, namely manual optimization techniques (i.e. scalar replacement, loop invariant motion and loop unrolling) and compiler optimization options (i.e. -O1, -O2, -O3, -O4, -Os and the -Funroll-loops).

With manual optimizations, the best performances are obtained when we optimize the SDMP versions using LpU factors u=8, u=16 and u=32. On the other hand, when we add the compiler optimizations, we find that the options -Oi (i=1, 2) give the best performances. This result is true for SDMP-DNS, SDMP-CSR and SDMP-COO versions and on all the tested Intel Xeon architectures. Concerning the optimal format, when the algorithms are optimized using manual optimizations, the best performances are obtained by the COO format, after applying the LpU, on Nantes Econome, Nancy Graphite and Lyon Orion architectures. Whereas, the CSR format is the best on Rennes Parapide, Grenoble Edel and Sophia Suno architectures. When we add the compiler optimizations, the performances of the algorithms are improved, and the order of the versions is maintained. To conclude, our work arises some interesting points which may constitute a second step we intend to study soon. We may cite:

- studying the SDMP for regularly structured sparse matrices (triangular, band, …) and the corresponding formats.
- studying the sparse-dense matrix on other architectures.
- processing very large sized matrices by parallelizing SDMP algorithms.

REFERENCES

[1] Emad, N., Hamdi-Larbi, O., Mahjoub, Z.: On sparse matrix-vector product performance evaluation for efficient distribution on large scale systems. In The 9th Hellenic European Research on Computer Mathematics and its Applications Conference (HERCMA09), Athens, Greece (2009).

[2] Buluc, A., Gilbert, J. R.: On the representation and multiplication of hyper sparse matrices. In Parallel and Distributed Processing, IEEE International Symposium on (pp. 1-11), IPDPS (2008).

[3] Ezouaoui, S., Hamdi-Larbi, O., Mahjoub, Z.: Dense-Sparse Matrix Multiplication: Algorithms and Performance Evaluation, In ACECS'14, pp.8794, Sousse, Tunisia, (2014).

[4] Ezouaoui, S., Mahjoub, Z., Mendili, L., Selmi, S.: Performance Evaluation of Algorithms for Sparse Dense Matrix Product, In IMEC13, pp. 257262, Kowloon, Hong Kong, (2013).

[5] Ezouaoui, S., Hamdi-Larbi, O., Design, Analysis and Performance Comparison of Sparse Matrix Product Algorithms, University of Tunis El Manar Faculty of Sciences of Tunis, Tunis, Tunisia, (2015).

[6] Ezouaoui, S., Hamdi-Larbi, O., Mahjoub Z., Towards Efficient Algorithms for Compressed Sparse-Sparse Matrix Product., HPCS 2017, Genoa, Italy, July 17-21, (2017).

[7] Hamdi-Larbi, O.: Etude de la Distribution sur Système Grande Echelle de Calcul Numérique Traitant des Matrices Creuses Compresses , Thesis, University of Tunis El Manar, Faculty of Sciences of Tunis, (2010).

[8] Hamdi-Larbi, O., Emad, N., Mahjoub, Z.: On Sparse Matrix-Vector Product Optimization, In AICCSA05, Cairo, Egypt, (2005).

[9] Howell, G.W.: Wide or Tall and Sparse Matrix Dense Matrix Multiplications, Proceedings of the 19th High Performance Computing Symposia, CA, USA, (2011).

[10] Bader, M., Heinecke, A.: Cache oblivious dense and sparse matrix multiplication based on peano curves, In Proceedings of the PARA, 8, Germany, (2008).

[11] Koanantakool, P., Azad, A., Buluc, A., Morozov, D., Oh, S., Oliker, L., Yelick, K.: Communication-Avoiding Parallel Sparse-Dense Matrix-Matrix Multiplication, IEEE International Parallel and Distributed Processing Symposium (IPDPS), USA, (2016).

[12] Hoste, Kenneth, Eeckhout, l.: Cole: compiler optimization level exploration, Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization. ACM, (2008).

[13] Matrix Market, www.math.nist.gov/MatrixMarket, 2020.

[14] Tim Davis Matrix, www.cise.ufl.edu/research/sparse/matrices, 2020.

AUTHORS

First Author – Olfa Hamdi-Larbi, University of Tunis El Manar - Faculty of Sciences of Tunis, Tunis, Tunisia, oarbi@taibahu.edu.sa
Second Author – Elwsaef Rim, Taibah University-College of Business Administration, Madinah, KSA, elwsaefrim@gmail.com