Understanding the performances of SMVP on multiprocessor platform

I. MEHREZ[1,2], O. HAMDI-LARBI[2], T. DUFAUD[1,3] and N. EMAD[1,3]
[1]Université de Versailles St-Quentin, Université Paris-Saclay
Li-Parad, 78035, Versailles, France
[2]Université de Tunis El Manar, Faculté des Sciences de Tunis
URAPOP, 2092, Tunis, Tunisie
[3]Maison de la Simulation, 91191, Saclay, France

Abstract— *Sparse Matrix Vector Product (SMVP) is an important kernel in many scientific applications. In this paper we study the performances of this kernel on multiprocessor platform using four different compression format (CSR, CSC, ELL and COO). Our aim is to extract runtime environment parameters, matrix characteristics and algorithm parameters that impact performances. This work is in the context of implementing an auto-tuner system for Optimal sparse Compression Format (OCF) selection.*

Keywords: sparse matrix, memory access, compression format, data parallel

1. Introduction

Several applications in scientific computing handle large sparse matrices with regular or irregular structures. To reduce both spatial and temporal complexities, these matrices require the use of a particular data storage format as well as the use of parallel or distributed target architectures. For this purpose, many particular structures for sparse data compression are known in the literature. The used compression format may highly affect the performance of the sparse application. The choice of the most appropriate one generally depends on several factors including: the matrix structure, the architecture of the target platform, the numerical method and the parallel programming model. Given the diversity of these factors, an optimized choice for one set of input data can lead to poor performance for another. Hence the interest of using a system allowing the automatic selection of the optimal Compression Format (OCF) by taking into account these different factors. In this context, we presented the modeling of a auto-tuned system which, given a sparse matrix, a numerical method, a parallel programming model and an architecture, can automatically select the OCF and propose its associated implementation [1][2]. In a first step, we validated our modeling with a case study involving (i) Horner scheme, and Sparse Matrix Vector Product, as a numerical method, (ii) CSC, CSR, ELL, and COO as compression formats, (iii) parallel data as a programming model and (iv) a multiprocessor platform as target architecture. This study allows us to extract a set of metrics and parameters that affect the selection of the OCF [1][2]. We showed that metrics extracted from the data parallel model analysis are not enough to make a decision (selection of the OCF). Therefore, we defined new metrics involving the number of operations and the number of indirect accesses (access to an array element). Thus, we proposed a new decision process that takes into account both the analysis of the data parallel programming model and the analysis of the algorithm [2].
In this paper, we focus on extracting the main parameters that affect the choice of the OCF and that are related to runtime environment. For that, we choose Sparse Matrix Vector Product (SMVP) as a case study since it represents an important kernel in many scientific applications that involve iterative operations. We study its performances using four different compression formats namely CSR, CSC, ELL and COO on a multiprocessor platform. We start with applying data parallel model to our computing kernel for each format. Therefore, data (nonzero elements) are mapped to a virtual geometry where each element of the geometry represents a virtual processor. A set of virtual processors are then affected to a physical one that is involved in the computing. For assigning virtual processors to physical processors, we use the Sorted Generalized Fragmentation with Balanced Number of Non-zeros algorithm (S-GBNZ) [8].
This paper is organized as follow: in section 2 we present a state of the art on SMVP drawbacks defined in the literature. In section 3, we present sparse compression format used in our study. In section 4 we present SMVP algorithm for each studied compression format. Section 5 is devoted for data parallel model presentation. It explains how data are mapped to processors. Experimental results are presented in section 6.

2. SMVP drawbacks

In this section we try to summarize the most commonly known drawbacks that impact the SMVP performances. we summarize these in the following points:

- Short vectors: in sparse matrices, generally, rows and columns have small number of nonzero elements which create an overhead with significant cost [6][9][10].

- Indirect memory access: the use of compressed format causes a large number of indirect access. This is due to the use of array elements to compute the position of a non-zero element at each iteration [6][9][11].
- Irregular memory access: this this due to the irregular access to vector X which depends on the sparcity structure of the matrix[9].

3. Sparse Compression Format

Several formats are proposed in literature to optimize sparse matrix storage. In this paper we study the performances relative to four different formats:

- Due to its simplicity and effectiveness, the CSR (Compressed Sparse Row) format remains the most used. To store an $n \times n$ sparse matrix A with nnz nonzero elements, CSR format uses three arrays: val, of size nnz, to store the nonzero elements of A on a row-wise (from row 1 to row n), col_ind, of size nnz, to store column indices of each stored element in val. and row_ptr, of size $n+1$, to store pointers on the head of each row in arrays val and col_in with $row\_ptr(n) = nnz$ (Fig.1).
- In opposite of CSR, the Compressed Sparse Column (CSC) uses the arrays val to store the nonzero elements of A on a column-wise. By analogy, it uses two other arrays: row_ind, to store row indices of each stored element in val. and col_ptr, of size $n + 1$, to store pointers on the head of each column in arrays val and row_ind with $col\_ptr(n) = nnz$ (Fig.2).
- The COO (COOrdinate) format uses also three array to store sparse matrix: val, of size nnz, to store the nonzero elements of A, row_ind and col_ind, of size nnz, to store respectively row and column indices of each stored element in val (Fig. 3). In this paper, we assume that non zero elements are stored in row (COO_R) or column (COO_C) order.
- The ELLPACK/ITPACK(ELL) format uses two 2D arrays, val (the ELLPACK matrix)and ind, of size $n \times nzmaxR$, to store a matrix with row or column compression. In this paper, we present the ELL format with a row compression (Fig. 4). Each row i of val is used to store the compressed i^{th} row of A. the second array ind stores the corresponding column indices of each element in val ($nzmaxR$ is the maximum number of nnz elements per row). Extra elements are padded with zeros.

4. Sparse Matrix Vector Product (SMVP)

Sparse Matrix Vector Product is an important computing kernel on sparse numerical methods. Let A be a $(n \times n)$ sparse matrix with nnz nonzero elements. Our aim is to compute $Y = A \times X$, where X is an input dense array with n elements and Y the result array. The SMVP algorithm depends on the used compression format for storing the sparse matrix. Thus, we evaluate four different optimized implementations of the SMVP kernel.

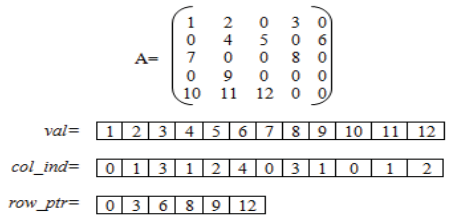

Fig. 1: CSR format

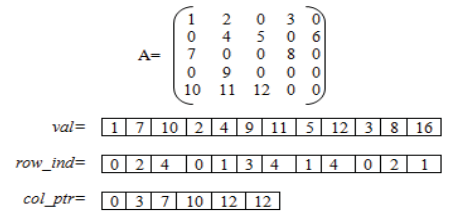

Fig. 2: CSC format

5. Data parallel programming model

In this paper, we present our study using the Data Parallel (DP) programming model. Indeed, we apply the same computing kernel (SMVP) on different matrix portions. According to this programming model, data is structured onto a virtual geometry representing the layout of virtual processors. In [4], some parameters are presented to evaluate and compare data parallel algorithms. We use extracted parameters to select the OCF [1][2].
Let:

- p be the number of virtual processors in the virtual geometry (1D, 2D, ...),
- α be the number of virtual processors concerned by a data parallel operation,
- cx be the number of data parallel operations (other than communications).

The computation ratio for a data parallel operation ops is defined by $\frac{\alpha_{ops}}{p}$ then, the average computation ratio is:

$$\Phi = \frac{1}{cx} \sum_{\alpha_{ops}=1}^{\alpha_{ops}=cx} \frac{\alpha_{ops}}{p} \quad (1)$$

The best data parallel algorithm to solve a scientific problem is the one with the smallest cx (the numerical aspect is supposed correct). Thus, the data parallel algorithm which maximizes Φ is often chosen [4].
In this paper, we use SMVP as a computing kernel. Let A be a $(n \times n)$ sparse matrix. We organize a one-dimensional

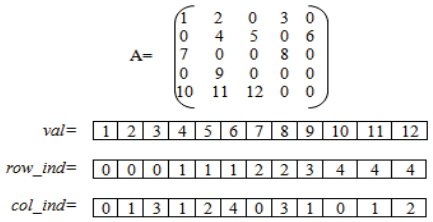

Fig. 3: COO format

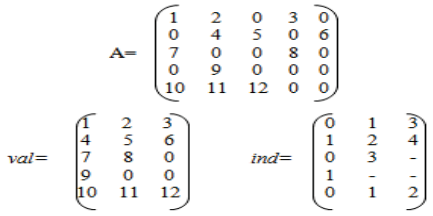

Fig. 4: ELLAPCK/ITPACK format

virtual geometry (with n virtual processors), on which we map and distribute data relatively to the used compressed format.

5.1 Row-wise format

When nonzero elements are stored in row order (CSR,ELL and COO_R), each i^{th} row of the matrix is mapped on a virtual processor i. Vector X is aligned on each element of the geometry. Each i^{th} virtual processor of the geometry performs a dot product of row i of the matrix by vector X using only one data parallel vector operation ($\alpha_{\times}$=1). In worst (respectively best) cases, the cost of this data parallel operation is $nzmaxR$ (respectively $nzminR$). $nzmaxR$ (respectively $nzminR$) is the maximum (respectively minimum) number of nnz elements per row. The i^{th} virtual processor of the geometry stores the i^{th} component of $A \times X$. Then partial results are gathered to have the final result Y.

5.2 Column-wise format

When nonzero elements are stored in column order (CSC or COO_C), each j^{th} column of the matrix with the corresponding element of x (j^{th} element) are mapped on the j^{th} virtual processor of the geometry. Each virtual processor j updates the vector Y: it performs a "$saxpy$" operation involving the j^{th} column of the matrix ($val(:, j)$) and the j^{th} element of the vector X ($X(j)$). In worst (respectively best) cases, the cost of parallel $saxpy$ operation is $nzmaxC$ (respectively $nzminC$). $nzmaxC$ (respectively $nzminC$) is the maximum (respectively minimum) number of nnz elements per column. Then, we have to sum the partial results with data parallel reduction with vector addition operation (with complexity $O(log_2(n))$)
Table 1 represents metric extracted from applying data parallel model on SMVP computing kernel.

Algorithm 1: SMVP for CSR format

```
1 for i=1 to n do
2     s = 0;
3     e = row_ptr[i + 1];
4     for j= row_ptr[i] to e do
5         indX = col_ind[j];
6         s = s + val[j] × X[indX];
7     end
8     Y[i] = s;
9 end
```

Algorithm 2: SMVP for CSC format

```
1 for i=1 to n do
2     x = X[i];
3     e = col_ptr[i + 1];
4     for j= col_ptr[i] to e do
5         indR = row_ind[j];
6         Y[indR] = Y[indR] + val[j] × x;
7     end
8 end
```

6. Experimental study

This section is devoted to the presentation of experimental study and the interpretation of results.

6.1 Experimental setup

6.1.1 Matrices dataset

Our matrices dataset is composed of:

- Real matrices selected from Tim Davis collection of sparse matrices (structured and unstructured matrices) [5]. These matrices cover a wide spectrum of domains such as structural engineering, computational fluid dynamics, electromagnetic, optimization, circuit simulation etc,
- Generated structured sparse matrices (diagonal and triangular matrices),
- Pathological matrices (Figure 5),

Table 2 represents statistics on matrices used in our study.

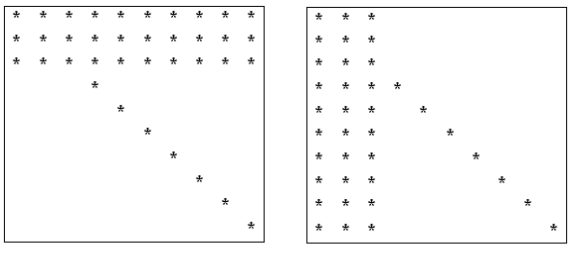

Fig. 5: Example of pathological matrices used in our tests

Algorithm 3: SMVP for ELL format

```
for i=1 to n do
    s = 0;
    j = 0;
    while j <nzmaxR and 0 <val[i][j] do
        IND = ind[i][j];
        s = s + val[i][j] × X[IND];
        j++;
    end
    Y[i] = s;
end
```

Algorithm 4: SMVP for COO format

```
for i=1 to nnz do
    indR = row_ind[i];
    IND = col_ind[i];
    Y[indR] = Y[indR] + val[i] × X[IND];
end
```

6.1.2 Experimental platform

We run SMVP on a multiprocessor platform (Table 3) with 100, 140, 160 and 200 physical processor.

6.1.3 Data scheduling

To assign virtual processors to physical processors, we use the S-GBNZ approach. This later is proposed in [8] for the SMVP partitioning on a large scale distributed systems. The objective of this algorithm is to decompose a sparse matrix into noncontiguous row blocks by balancing the number of nonzero elements between blocks (Figure 6). The S-GBNZ approach consists of three phases as follows:

- Phase 0 is a sorting phase; it consists in sorting the input matrix rows in decreasing number of nonzero elements.
- Phase 1 is based on LS heuristic (List Scheduling). In the first step, the first p rows of the input matrix are assigned respectively to the p partitions (p is the number of fragments): each row $i(i = 1...p)$ is assigned to the i^{th} partition. Thereafter, the remaining rows are assigned to the fragments based on their load (number of nonzero elements).
- Phase 2, which is a phase of improvement, is an iterative heuristic. It allows, through successive refinements, to improve the previous partitioning, leading to a better balance. The chosen criterion is IF (Imbalance Factor) which is defined as the difference between the maximal and the minimal loads. This phase consists on making exchanges or transfers between successive rows of the partition with the maximal load and the least loaded one.

Table 1: Vector data parallel metrics

	$CSR/ELL/COO_R$	CSC/COO_C
cx_{vect}	1	1
p_{vect}	n	n
$\alpha_{\times}$	n	n
$cost_{Vops}$	$nzmaxR$	$nzmaxC$

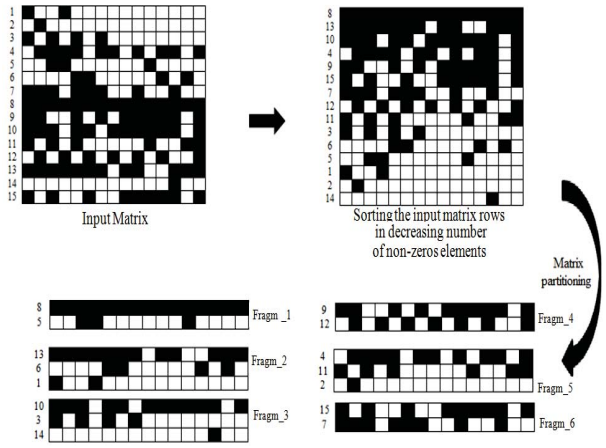

Fig. 6: S-GBNZ algorithm

6.2 Results Interpretation

Statistics presented in this section represents the results of 3000 different SMVP using different compression format namely CSR, CSC, ELL and COO (COO_R and COO_C). Table 4 represents notations used in this section.

6.2.1 Data access

Let PP_{load_mx} be the most loaded processor (with maximum nnz elements) and PP_{tmax} the processor with the maximum execution time ($tmax$). We make statistics on SMVP performances. Figure 7 shows that, in the case of the four studied format, it is very uncommon that the most loaded process be the one with worst performances (PP_{load_mn} =PP_{tmax}). We conclude that the load of the processor (nnz elements on which depends operation numbers), is not a sufficient criterion to describe its computational load. Analyzing SMVP algorithm (Algorithms 1,2,3 and 4), we remark that the increasing number of read/write data and indirect access at the same time affect its performances (Table 5). Therefore, we choose to attribute a $cost$ to SMVP that represents PP_{tmax}. Since each format has its own cost, we define:

$$\begin{aligned} CSR_{cost} = (3 \times nrPP_{tmax} + 6 \times nzPP_{tmax}) \times R_{cost} \\ + (4 \times nrPP_{tmax} + 3 \times nzPP_{tmax}) \times W_{cost} \\ + (2 \times nrPP_{tmax} + 4 \times nzPP_{tmax}) \times AC_{cost} \\ + 2 \times nzPP_{tmax} \times OP_{cost} \end{aligned} \quad (2)$$

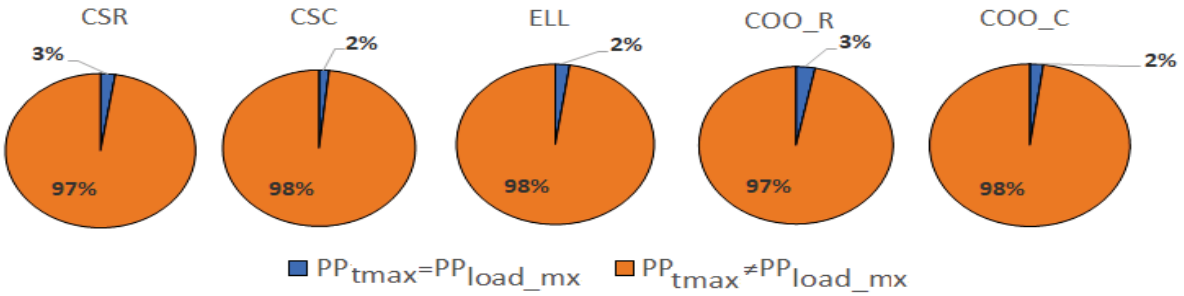

Fig. 7: Statistic

Table 2: Statistics on Matrix Dataset

n	nnz	Density
[362: 2.99E+06]	[1000:2.10E+09]	[0.0001:80]%

Table 3: Experimental platform

Cluster within Grid5000 platform	
#Node	5, 7, 8 and 10
Processor	Intel Xeon E5-2630 v4
CPU/node	2
Core/CPU	10

$$CSC_{cost} = (3 \times ncPP_{tmax} + 6 \times nzPP_{tmax}) \times R_{cost} + (3 \times ncPP_{tmax} + 3 \times nzPP_{tmax}) \times W_{cost} + (2 \times ncPP_{tmax} + 5 \times nzPP_{tmax}) \times AC_{cost} + 2 \times nzPP_{tmax} \times OP_{cost} \quad (3)$$

$$ELL_{cost} = (2 \times nrPP_{tmax} + 6 \times nzPP_{tmax}) \times R_{cost} + (4 \times nrPP_{tmax} + 3 \times nzPP_{tmax}) \times W_{cost} + (1 \times nrPP_{tmax} + 4 \times nzPP_{tmax}) \times AC_{cost} + 2 \times nzPP_{tmax} \times OP_{cost} \quad (4)$$

$$COO_{cost} = (6 \times nzPP_{tmax}) \times R_{cost} + (4 \times nzPP_{tmax}) \times W_{cost} + (6 \times nzPP_{tmax}) \times AC_{cost} + 2 \times nzPP_{tmax} \times OP_{cost} \quad (5)$$

where

- AC_{cost} is the cost of an array element address computing (Indirect Access),
- R_{cost} is the cost of a Reading access,
- W_{cost} is the cost of a Writing access,
- OP_{cost} is the cost of an arithmetic operation (multiplication or addition).

We note that these costs highly depend on the used hardware (cache levels, CPU frequency, etc.) and the load of PP_{tmax}. Thus, they can not be estimated in advance. Indeed, the proposed costs serve only to explain that the processor load only is not sufficient to predict the one with minimum execution time.

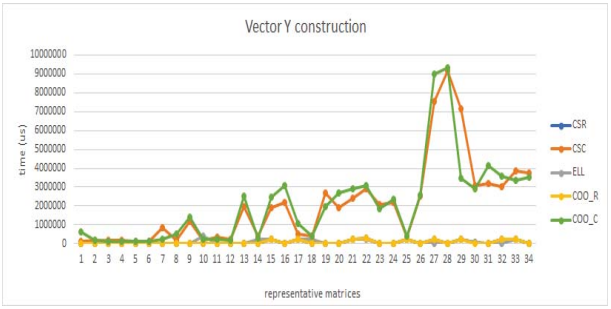

Fig. 8: Final result construction

6.2.2 Final result construction

The construction of the final results Y depends on the used compression format. It is either a *reduce* or a *gather* operation which have different costs. Figure 8 illustrates this difference using 34 representative matrices from our dataset (n > 200000).

- Matrix partitioning in row blocks: the case of CSR, ELL and COO_R format. A *gather* operation is performed to collect partial results from each processor (on average, sending an array of size n/PP). A sequential operation of array elements reordering is performed at the end (Figure 10).
- Matrix partitioning in column blocks: the case of CSC and COO_C format. A parallel *reduce* operation is performed to sum partial results from each processor (O($log_2(n)$) operation '+'). See Figure 9.

6.2.3 MPI process synchronization

Theoretically, PP physical processors start a task in the same date t_0. Which is also the case of synchronization region ($Barrier$). Each processor can then execute tasks with different size/time (Figure 11 (a)). Although, experiments show that all processor don't really start together and we don't obtain an 100% synchronization(Figure 11 (b)). This is due to the used execution environment that we can't predict or control its behavior. Figure 12 illustrate the gap between first and last processor beginning. This gap can produce a false prediction of results (predict the OCF which best

Table 4: Notation used in equation.

Notation	Meaning
PP	the number of physical processors
PP_{load_mx}	the most loaded processor (with maximum nnz elements)
PP_{tmax}	processor with the maximum execution time ($tmax$)
$nzPP_{tmax}$	number of nnz element in PP_{tmax}
$nrPP_{tmax}$	number of rows in PP_{tmax}
$ncPP_{tmax}$	number of columns in PP_{tmax}
AC_{cost}	the cost of an array element address computing (Indirect Access)
R_{cost}	the cost of a Reading access
W_{cost}	the cost of a Writing access
OP_{cost}	the cost of an arithmetic operation (multiplication or addition)

Table 5: Data Access

Format	#Read	#Write	#Indirect Access	#Operations
CSR	$3 \times nrPP_{tmax} + 6 \times nzPP_{tmax}$	$4 \times nrPP_{tmax} + 3 \times nzPP_{tmax}$	$2 \times nrPP_{tmax} + 4 \times nzPP_{tmax}$	$2 \times nzPP_{tmax}$
CSC	$3 \times ncPP_{tmax} + 6 \times nzPP_{tmax}$	$3 \times ncPP_{tmax} + 3 \times nzPP_{tmax}$	$2 \times ncPP_{tmax} + 5 \times nzPP_{tmax}$	$2 \times nzPP_{tmax}$
ELL	$2 \times nrPP_{tmax} + 6 \times nzPP_{tmax}$	$4 \times nrPP_{tmax} + 3 \times nzPP_{tmax}$	$1 \times nrPP_{tmax} + 4 \times nzPP_{tmax}$	$2 \times nzPP_{tmax}$
COO	$6 \times nzPP_{tmax}$	$4 \times nzPP_{tmax}$	$6 \times nzPP_{tmax}$	$2 \times nzPP_{tmax}$

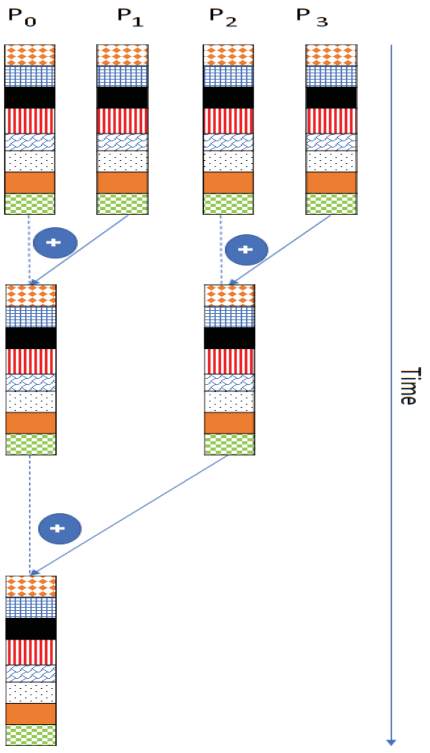

Fig. 9: Reduce operation

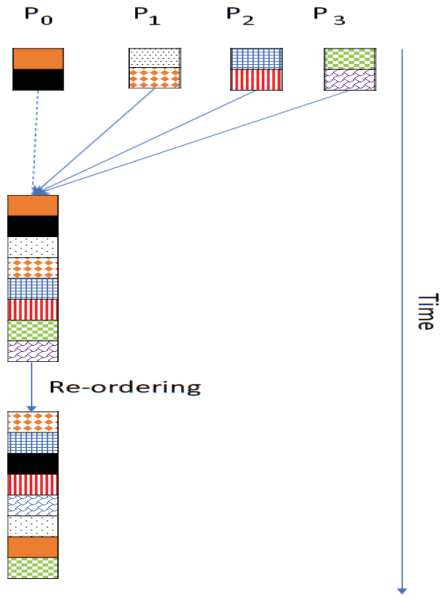

Fig. 10: Gather operation

SMVP performances).

7. Conclusions

In this paper, we present performance issue of Sparse Matrix Vector Product on multiprocessors platform. Our goal is to study SMVP and extract parameters that impact its performances. We do not look at optimizing the kernel but to define metrics that have an impact on the sparse

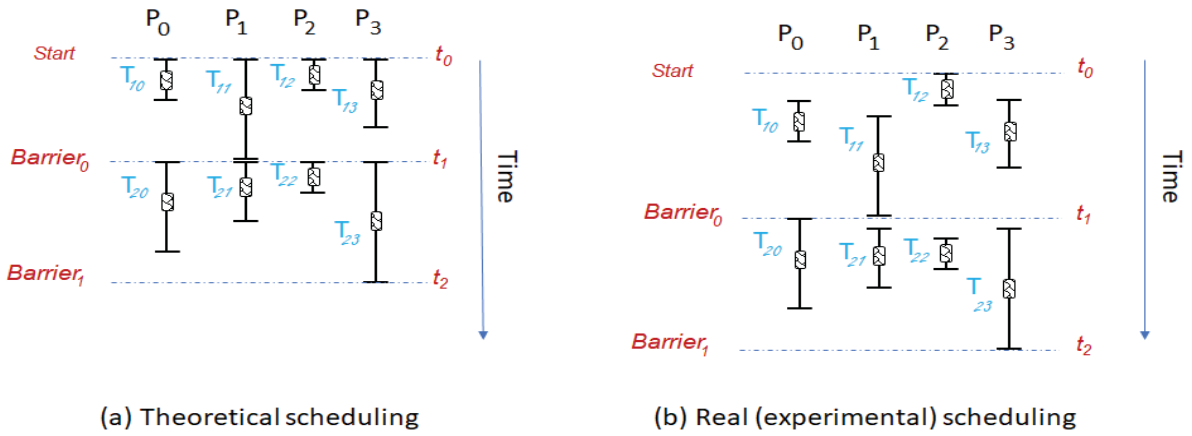

Fig. 11: Parallel process scheduling

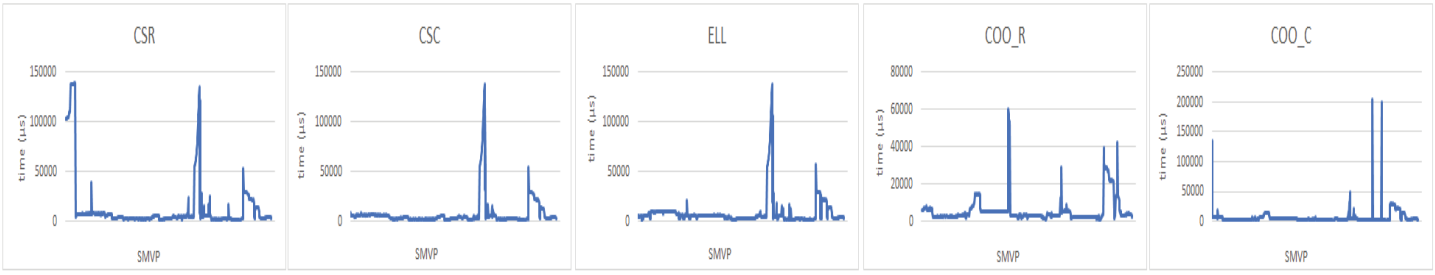

Fig. 12: Gap between first and last processor

compression format selection process. Indeed, the final aim of this work is to design and implement an auto-tuner system for Optimal sparse Compression Format selection. Our study suggests that it is possible to extract important parameters that influence performances from the data parallel programming model. Although, the analysis of algorithms leads to other models that are dependent on hardware. Thus, in further work, we propose to derive an approach using a machine learning method and improve this method by selecting features from a theoretical study of models involved in the implementation (for example the data parallel programming model).

Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] I. Mehrez, O. Hamdi-Larbi, T. Dufaud and N. Emad, "Towards an auto-tuning system design for optimal sparse compression format selection with user expertise," in *Proc. AICCSA*, 2016.

[2] I. Mehrez, O. Hamdi-Larbi, T. Dufaud and N. Emad, "Understanding the Performances of Sparse Compression Formats Using Data Parallel Programming Model," in *Proc. HPCS'17*, 2017, p. 667–674.

[3] O. Hamdi-Larbi, Z. Mahjoub and N. Emad,"Load balancing in re-processing of large-scale distributed sparse computation," in *Proc. LCI Conference on High Performance Clustered computing*, 2007.

[4] S. Petiton and N. Emad, "A data parallel scientific computing intro-duction," *Computer Science.*, vol. 1132, pp. 45–60, 1996.

[5] T. A Davis and Hu. Yifan, "The University of Florida Sparse Matrix Collection," *ACM Trans. Math. Softw.*, vol. 38, pp. 1–25, Dec. 2011.

[6] R. Shahnaz, A. Usman and I. R. Chughtai, "Review of Storage Techniques for Sparse Matrices," *Proc. 2005 Pakistan Section Multitopic Conference*, 2005.

[7] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., 2013.

[8] O. Hamdi,*Etude de la distribution, sur systeme e grande echelle, de calcul numerique traitant des matrices creuses compressees*, PhD report, FST (Tunisia)-UVSQ (France), March 2010.

[9] Georgios I. Goumas, Kornilios Kourtis, Nikos Anastopoulos, Vasileios Karakasis and Nectarios Koziris "Performance evaluation of the sparse matrix-vector multiplication on modern architectures," *The Journal of Supercomputing*, vol. 50, pp. 36–77,2008

[10] A. Buttari, V. Eijkhout, J. Langou, and S. Filippone. "Performance optimization and modeling of blocked sparse kernels", Innovative Computing Laboratory, University of Tennessee, Tech. Rep ICL-UT-04-05, 2005.

[11] Pinar Ali, and Heath Michael T., "Improving Performance of Sparse Matrix-vector Multiplication", in *Proc. Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*,1990, New York, NY, USA.