# *SMArtOp*: A Java Library for Distributing High-dimensional Sparse-matrix Arithmetic Operations

Antonela Tommasel, Daniela Godoy, Alejandro Zunino

PII: S0167-6423(17)30126-0
DOI: http://dx.doi.org/10.1016/j.scico.2017.06.005
Reference: SCICO 2109

To appear in: *Science of Computer Programming*

Received date: 31 December 2016
Revised date: 8 June 2017
Accepted date: 10 June 2017

## Highlights

- Sparse-matrix operations are commonplace in computational science.
- Existing libraries are mostly unable to cope with high-dimensional data.
- SMArtOp is a Java library for distributing large-scale sparse-matrix arithmetic operations.
- SMArtOp relies on the characteristics of operations and the matrices involved.
- SMArtOp improved execution times and resource needs of popular algebra libraries.

# *SMArtOp*: A Java Library for Distributing High-dimensional Sparse-matrix Arithmetic Operations

Antonela Tommasel[a,*], Daniela Godoy[a], Alejandro Zunino[a]

[a] *ISISTAN, UNICEN-CONICET. Facultad Cs. Exactas. Campus Universitario, Tandil (B7001BBO), Argentina. Tel.: +54 (249) 4439682 ext. 35. Fax.: +54 (249) 4439681*

**Abstract**

Sparse-matrix operations are commonplace in computational science, and novel solutions for speeding-up them are essential for numerous applications. *SMArtOp* is a software for efficiently dividing and distributing the processing of large-scale sparse-matrix arithmetic operations. This software relies on both the distinctive characteristics of each type of arithmetic operation and the particular matrices involved to split the operations into parallel and simpler tasks. Experimental evaluation showed the speeding-up and resource consumption advantages of the proposed software, in comparison to other linear-algebra libraries.

*Keywords:* big-scale sparse-matrix, matrix arithmetic operation, distributed computing

## 1. Introduction

High-dimensional sparse-matrices are vital in scientific computing and engineering where matrices have thousands or millions of elements. Operating with such matrices represents the dominant cost when solving optimisation problems and processing images. Even though sparse-matrix operations need to be efficiently performed, managing them has proved to be a challenging task.

As technology evolves, both new multi-core and distributed architectures, and Graphics Processing Units (GPUs) are used for increasing computers' power, leading to a continuous demand of new tools to cope with these new architectures [1] and the huge volume of user generated data. However, most efforts are focused on developing centralised libraries that operate with dense matrices. Hence, the parallel processing of sparse-matrix operations in distributed architectures is needed to deal with increasing performance and resource requirements.

*Corresponding author.
*Email address:* `antonela.tommasel@isistan.unicen.edu.ar` (Antonela Tommasel)

| *Library* | *Data Type* | *Matrix Type* | *Execution Type* |
|---|---|---|---|
| PCOLT MKL | Double. Float. Long. Integer | Dense. Sparse | Multi-threaded |
| JAMPACK la4j COLT JAMA JBLAS Apache Common Maths | Double | Dense | Sequential |
| oj! Algorithms | Number. BigDecimal | Dense | Multi-threaded |
| UJMP | BigDecimal. Boolean. Byte. Double. Float. Int. Long. Object. Short. String | Dense. Sparse | Sequential |
| ND4j | Float | Dense | BLAS dependent. GPU parallel |
| CupSparse | Float. Double | Dense. Sparse | GPU parallel |
| numpy / scipy | int boolean float double long complex | Dense / Sparse | BLAS dependent |
| PETSc | double complex float long int | Dense. Sparse | Sequential, multi-threaded, multi-process |

Table 1: Linear Algebra Libraries Characteristics

This work presents *SMArtOp* (Sparse Matrix library for ARiThmetic Operations), one of the first Java libraries for operating with large-scale sparse-matrices on heterogeneous distributed environments, without needing specialised hardware. Experimental evaluation over a high-dimensional application showed that *SMArtOp* helped to significantly reduce both the storage and memory requirements, and the computing times of operations, when compared to other sequential, multi-thread and GPU-based linear-algebra libraries.

## 2. Problems and Background

Linear algebra has played a crucial role in computing since the rise of the first digital computers, hindering applications' performance due to the high computational complexity of its operations. Whilst dense-matrices have been intensively studied, sparse-matrices have received less attention. Also, diverse issues arise from the interactions between computer processors and data [1], as memory accesses continue to be relatively slow despite processors' speed. Thus, performance is usually limited by data synchronisation between processors and memory. The problem worsens if matrix's elements are repeatedly traversed. GPUs have been used as co-processors capable of handling large calculations in addition to CPUs. Generally, applications exploiting data level parallelism can attain good performance with GPUs [2]. However, obtaining the desired performance from GPUs is not simple, as it usually requires rewriting the algorithms [4], and specific and expensive hardware, which might be unavailable. Several linear algebra software libraries are available in programming languages, such as Fortran, C++, Python and Java. Table 1 presents the most commonly used linear algebra libraries[1], which mostly provide dense matrix implementations, unsuitable for sparse-matrices.

[1]A more comprehensive description of the libraries can be found in: `https://github.com/tommantonela/SMArtOp/wiki/Comparison-with-other-Software-Libraries`.

## 3. Software Framework

*SMArtOp* is designed for dividing and balancing the processing of large-scale sparse-matrix arithmetic operations for distributed execution. Operations are divided based on their intrinsic characteristics and the involved matrices, as in [6].

### 3.1. Software Architecture

*SMArtOp*'s architecture[2] addresses three flexibility concerns: matrix representation, operation implementation, and operation distribution. Matrix representations must comply with the `matrix.matrixImpl.Matrix` interface, which defines both access methods and operations that benefit from leveraging on direct access to data representations. Matrix operations are defined in `matrix.matrixComp.MatrixComputation`. Packages `matrix.reconstructionStrategy` and `matrix.policy` provide support to operation distribution by implementing the approach in [6]. The package `matrix.adapterDistribution` focuses on task execution and data sharing, providing an extension point for new distributed middlewares.

### 3.2. Software Functionalities

*SMArtOP* provides dense and sparse representations, and implements matrix addition, subtraction, multiplication, scalar multiplication, transpose, inverse and Laplacian. The inverse relies on the LU, QR or Cholesky decompositions. Operations support three execution modes: sequential, multi-thread, and distributed in a cluster. The sequential model (`MatrixComputationFD`) is a baseline implementation that does not leverage on matrix sparseness, whereas the multi-threaded (`MatrixComputationSparsePar`) and distributed (`MatrixComputationSparseDistributed`) ones do. `MatrixComputationSparseDistributed` considers different metrics to distribute operations, providing alternatives for balancing tasks' work load.

## 4. Implementation and Empirical Results

### 4.1. Implementation Details

*SMArtOp* is implemented in Java, as it is portable and can achieve similar performance to other optimised languages [3]. Portability is highly relevant in heterogeneous systems, as computer clusters with different operating systems or even architectures, where a Java application can be run without re-compiling it. Matrix's values have float precision to balance memory requirements. Sparse-memory structures aim to decrease network usage, storage and memory requirements. Although *SMArtOp* can be extended to support any middleware for distributing computations over a computer cluster, a default implementation is provided by using the Java Parallel Processing Framework (JPPF).

[2]Further architectural details can be found in `https://github.com/tommantonela/SMArtOp/wiki/Software-Architecture`.

### 4.2. Experimental Results

*SMArtOp*'s performance was tested for the feature selection approach in [5], which involved computing $B = XX^T + \beta F L_A F^T$. Evaluation was based on *Digg* data[3], with matrices between $8,546$ and $42,843$ rows, with 66.94% to 99.96% of sparseness[4]. Non-distributed executions were run on an i7-3820 with 32 GB RAM and a NVIDIA Quadro K2200. The computer cluster included 5 AMD Phenom II X6 1055T with 8 GB RAM, and 3 AMD FX-6100 with 16 GB RAM.

Figure 1 shows $B$ execution times. *SMArtOp* outperformed most of the libraries in Table 1. Distributed results include the times of sharing data, synchronising and accessing results. JAMA, COLT-dense, PCOLT, oj!Algorithms, JAMPACK, La4j-sparse, JBLAS, numpy and numpy-Theano (on GPU) resulted in excessive memory requirements, being unsuitable for high-dimensional matrix operations. PCOLT required more than 18 days to compute $XX^T$ and numpy-Theano 13 minutes, whereas *SMArtOp* required 12 minutes to compute $B$. ND4J required 22 (GPU) or 33 (CPU) minutes, and scipy required 29 minutes, which is approximately twice the best *SMArtOp* time. Cusp-C++ and PETSc obtained the best GPU and CPU results respectively outperforming *SMArtOp* and MKL. However, PETSc, Cusp and MKL require coding in C++, reducing portability. MKL is available for a restricted processor set. Although operations were fast, data access was slow as retrieving the matrices required more time than operating, which resulted in times close to the best *SMArtOp* time. Despite leveraging on GPU, Cusp-Java obtained worse results than *SMArtOp*, showing the coding effort needed and the difficulty for leveraging on GPUs. In both ND4J and Cusp-Java most part of the time was spent on sharing data and synchronising results, instead of actually performing the arithmetic operations. The Multi-thread and Cusp-C++ executions were replicated on an i7-6700HQ with 16 GB RAM and a NVIDIA GTX 960M. The Multi-thread execution required 32 minutes, while Cusp-C++ required 45 minutes. These results show that mid-end GPUs might not be sufficient for improving CPU-based solutions. Sparse Apache computed $B$ in 39 minutes. Our Multi-thread implementation outperformed most of the evaluated libraries, whilst our Sequential one was outperformed by Apache, ND4J, Cusp, scipy and MKL. In summary, the improvements offered by *SMArtOp* ranged between 44% and 99%, considering the smallest (ND4J-GPU) and highest (PCOLT) improvements.

## 5. Illustrative Examples

Listing 1 shows $B$'s implementation including the `IntData` (dataset), `Matrix` and `MatrixComputation` definitions. The project's wiki[5] includes comprehensive

[3] *Available in* `https://github.com/tommantonela/SMArtOp/tree/master/dataset`

[4] Further evaluation using a dataset with over 100,000 rows and columns can be found in [6].

[5] `https://github.com/tommantonela/SMArtOp/wiki`

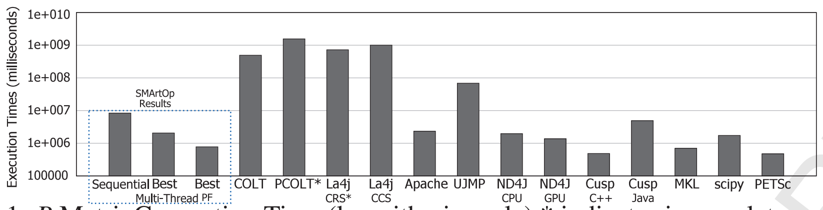

Figure 1: *B* MatrixComputing Time (logarithmic scale) * indicates incomplete execution

Listing 1: Code Example

```java
public Matrix computeB(IntData data){
    FactoryMatrixHolder.setFactory(new FactoryMatrixSparseHashPar()); //Matrix Type Definition
    //MatrixComputation defined according to the MatrixType
    MatrixComputation algebra = new MatrixComputationSparsePar();
    Matrix A = data.getPcoPost();
    A = algebra.multiplyByTranspose(algebra.transpose(A)); // Pt x P -- Matrix Multiplication I
    A = algebra.laplacian(A); // Laplacian A -- Addition-Subtraction I & Addition-Subtraction II
    Matrix aux = algebra.multiply(beta, data.getXpostFeature()); //beta x F
    aux = algebra.multiply(aux, A); //Matrix Multiplication II
    Matrix Ftrans = algebra.transpose(data.getFpostFeature());
    aux = algebra.multiply(aux,Ftrans); //Matrix Multiplication III
    Matrix X = algebra.multiplyByTranspose(data.getXpostFeature()); //Matrix Multiplication IV
    Matrix B = algebra.add(X, aux); //Addition-Subtraction III
    return B;
}
```

documentation regarding the integration to a Java project and code examples.

## 6. Conclusions

*SMArtOp* is a library for distributing high-dimensional sparse-matrix arithmetic operations on computer clusters by leveraging on multiple processors and their computational power. *SMArtOp* outperformed several popular linear algebra libraries regarding both the execution times and the required computational resources. Results confirmed *SMArtOp*'s feasibility and advantages for efficiently operating with high-dimensional sparse matrices in distributed environments.

[1] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Review*, 46(1):3–45, 2004.

[2] A. Lee, C. Yau, M.B. Giles, A. Doucet, and C.C. Holmes. On the utility of graphics cards to perform massively parallel simulation with advanced monte carlo methods. *J. Comp. Graph. Stat.*, 19(4):769–789, 2010.

[3] J. Moreira, S. Midkiff, M. Gupta, P. Artigas, P. Wu, and G. Almasi. The ninja project. *Communications of the ACM*, 44(10):102–109, October 2001.

[4] G. Oyarzun, R. Borrell, A. Gorobets, and A. Oliva. Mpi-cuda sparse matrix-vector multiplication for the conjugate gradient method with an approximate inverse preconditioner. *Computers & Fluids*, 92(0):244–252, 2014.

[5] J. Tang and H. Liu. Feature selection with linked data in social media. In *Proceedings of the 12th SDM*, pages 118–128. SIAM / Omnipress, 2012.

[6] Antonela Tommasel, Daniela Godoy, Alejandro Zunino, and Cristian Mateos. A distributed approach for accelerating sparse matrix arithmetic operations for high-dimensional feature selection. *Knowl Inf Syst*, 51(2):459–497, 2017.

## Required Metadata

## Current executable software version

| Nr. | (executable) Software metadata description | Please fill in this column |
|---|---|---|
| S1 | Current software version | v1.2 |
| S2 | Permanent link to executables of this version | `https://github.com/tommantonela/SMArtOp/releases/tag/v1.2` |
| S3 | Legal Software License | Apache Licence V 2.0 |
| S4 | Computing platform/Operating System | Java Compatible Platform. |
| S5 | Installation requirements & dependencies | Java Virtual Machine. Maven. Trove. JPPF for distribution support. |
| S6 | If available, link to user manual - if formally published include a reference to the publication in the reference list | `https://github.com/tommantonela/SMArtOp/wiki` |
| S7 | Support email for questions | antonela.tommasel@isistan.unicen.edu.ar |

Table 2: Software metadata (optional)

## Current code version

| Nr. | Code metadata description | Please fill in this column |
|---|---|---|
| C1 | Current code version | v1.2 |
| C2 | Permanent link to code/repository used of this code version | `https://github.com/tommantonela/SMArtOp/tree/v1.2` |
| C3 | Legal Code License | Apache Licence V 2.0 |
| C4 | Code versioning system used | Git |
| C5 | Software code languages, tools, and services used | Java. Eclipse IDE. Maven. Trove. JPPF for distribution support. |
| C6 | Compilation requirements, operating environments & dependencies | Java Compatible Platform. |
| C7 | If available Link to developer documentation/manual | `https://github.com/tommantonela/SMArtOp/wiki` |
| C8 | Support email for questions | antonela.tommasel@isistan.unicen.edu.ar |

Table 3: Code metadata (mandatory)