Parallel Implementation of Data Balancing Algorithms

Nelson Lopes Silva
nelsonflsilva@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2015

Abstract

The goal of this thesis is to describe our work on the development of a parallel implementation for the Input-Output data balancing algorithm of Rodrigues. This application aims to out-perform sequential methods, providing quicker solutions. An analysis is made to find the must computational challenge operations of this algorithm, and discusses possible optimization points. To support out implementation we use PETSc, a parallel library which provides highly optimized routines for linear algebra using vectors and matrices. We describe in detail how this tool is used to fulfill our requirements. This document presents the principal problems found during development, addressing the different solutions. Scalability was the main issue in this project. To overcome this problem we experimented with processor load balancing and matrix permutation. The performance was evaluated in a computer cluster, logging and comparing execution times for various test problems. Real-life, as well as artificial problems were used. The input data is provided in Octave files. The application loads this input file, creating and distributing the data across all machines. Various matrix permutations are considered and computational scalability are tested for different matrix structures. To do this we executed our application, activating or deactivating such optimizations at run-time.
Keywords: Input-Output Analysis, data balancing, parallel computing, PETSc

1. Introduction

The need for information affects multiple areas in modern society. Disciplines, such as policy analysis, consumes vast quantities of data to study location patterns. In industry, researchers process large chunks of data to find competitive advantages. In environment, as global population continues to grow, Environmentally-extended input-output (EEIO) creates a need for data to evaluate environment impacts, and economic drivers for global carbon and nitrogen emissions, and natural resources footprints can be identified. Input-Output Analysis (IOA) is yet another area demanding large data-sets to update regional Input-Output (IO) tables, predicting how goods are used in different economic sectors.

Problems rise when some of these values are suppressed, due to confidentiality or surveys issues. Moreover, data may also contain value inconsistencies, making these analysis more difficult. In response to this issue, methods for values estimation were developed. Indeed, using available values constraints in faulty data, such methods are capable of estimate missing values, and reconcile inconsistencies. However, these techniques often requires dealing with large amounts of information, which naturally result in an increase of computational time.

In response to this issue, computer clusters and multi-processor machines become attractive, as these provide elevated computational power, compared with ordinary machines. Indeed, parallel computing takes advantage of such architectures in an attempt to provide fast solutions to problems demanding high computational power.

Hence, the present work is concerned with implementation, in the context of parallel/distributed computing, of algorithms that deal with uncertainty in balancing IOA data. This field deals with the compilation of data which describes the interactions among different components of the economy and the environment and, among other applications, with the assessment of the impact of a demand stimulus on carbon emissions. The guiding line connecting the algorithms implemented in this thesis is that they deal with the estimation and propagation of uncertainty, in large sparse systems.

1.1. Project Objectives

The main objective of this work is to speedup the computational process, finding quicker solutions, comparing with sequential methods. That said, in a nutshell, the goal of this work is to provide parallel implementations of algorithms to solve problems of such dimensions. Moreover, the new parallel algo-

rithms should be portable across computer clusters, and properly scale for real life IO data.

Also important is the group of problems which can be solved by our method. In other words, it is necessary to perform an analysis to find which problems work best in the parallelization technique. Since different topologies offer different matrix structures, data division and parallelization will change.

Currently there exists computer software capable of balancing IO data, implemented in high level languages, such as Octave. However, these solutions are limited to small systems, since it shows slow results for bigger systems. Furthermore, current solutions do not implement parallel operations, making it impossible to run many operations at the same time.

1.2. Proposed Solution

We propose a solution based on distributed memory architectures, as these provide the needed computational resources for this project. Implementation is done using Message Passing Interface (MPI), sending data across nodes within the cluster.

Objects, such as matrices and vectors, are divided by chunks of contiguous rows and sub-vectors, respectively. The algorithm proceeds by balancing input data, saving the final solution in an output file. To accomplish this, we use Portable Extensible Toolkit for Scientific Computation (PETSc) [1], an open-source library which provides parallel implementation of basic operations between matrices and vectors. Due to its dimensions, matrices are stored in sparse format, reducing memory requirements and entries to process.

Furthermore, our system is capable of permuting matrix rows. Some input matrices may present non-zero pattern, that impede parallel operations or demand high number of message to be exchanged. Hence, reshaping matrix structures can potentially increase performance in some cases.

1.3. Challenges

Parallelization of algorithms raises problems sequential implementations do not face. For instance, consider memory allocation and object creation. In traditional sequential solutions, large chunks of contiguous memory may be used, storing complete matrices and vectors. However, in parallel environments, data must be divided and distributed across machines. This task is crucial since all processors should have identical workloads for an optimal performance. Finding the best object division is a challenge, as different problems may require different approaches.

Another interesting aspect of parallel computing is the communication between machines. Since algorithms run in multiple machines, these require messages to be exchanged across the network. Considering that communication costs are considerable higher, comparing with processing times, this task introduces overhead to the overall process. Algorithms must aim for homogeneous object distribution and low communication overheads.

2. Problem Formulation

IOA was developed by Professor Wassily Leontief in the 1930s, as a framework to analyze interdependence of economic sectors [5]. The IO model is often referred as Leontief Model for this same reason. In its core, an IO model is a system of Linear Equations where, each represents the distribution of a given product through the economic sectors. In other words, it consists in a table showing how economy sectors's products (outputs) is used in other economy sectors (inputs).

Leontief attempted to manually perform the first IOA using a table representation of the US Economy in 1919. However, since it consisted in a 42-sector IO table, the required equation solving was performed by a calculation machine from John Wilbur at the Massachusetts Institute of Technology (MIT) [2].

Leontief model also been extended to other frameworks, dealing with employment and social account data, regarding industrial production, international and inter-regional flow of products and services. Furthermore it is also used in frameworks, analyzing energy consumption and environment pollution [5].

IOA may be extended to problems of different nature. For instance, consider the conciliation and update of Social-Accounting Matrix (SAM) matrices, which require coherent data. These tables often suffer from deficient row and column sums, i.e, these values do not add [7]. Moreover, IO tables may also present incomplete data, as some values may be missing. This may be due to various reasons, varying from inefficient industry surveys, as these may be costly, or suppression of confidential data [4]. In these cases, IOA may be used as a technique for both, estimate suppressed values and balance rows and columns sums in SAM.

EEIO is yet another field in which IOA is used to study relations between economic consumption activities and environment impacts [3]. Consider the consumption of beef in the modern society. In order to produce beef, inputs such as water and animal food must be consumed. Additionally, to provide water and food to, other activities must take place, such as the construction of water delivery structures, and food plantations. Continuing the same reasoning, it is possible to estimate global impacts, such as carbon, water and nitrogen footprints involved in beef production [3].

	Ag	Ma	$\mathtt{x}^{\mathtt{r}}$
Ag	8	5	13
Mg	4	2	6
$\mathtt{x}^{\mathtt{c}}$	12	7	

Table 1: IO table for a two sector economy.

2.1. Data Balancing

The specific problem being addressed in this work is the balancing of IO tables, such as SAM, with arbitrary structure, uncertainty estimates and multiple data source.

To further illustrate this idea, consider the example of the IO Table 1, based on EEIO explanation [3]. This table accounts for a two sector economy - Agriculture (Ag) and Manufacture (Ma) - where each sector sells and buys goods from the other.

In this example, business within sector Ag bought \$4 worth of goods and services from business in sector Mg, and business in sector Ag sold \$8 worth of services and goods for the same sector. Furthermore, $\mathtt{x}^{\mathtt{r}}$ and $\mathtt{x}^{\mathtt{c}}$ are the rows and columns sums, respectively. Note that the theses values must be coherent with one another, meaning that, column and row sums must add up to matrix values total. From this example, information can be divided in two vectors and a matrix, as expressed in Equation 1:

$$\mathbf{Z} = \begin{bmatrix} 8 & 5 \\ 4 & 2 \end{bmatrix}, x_r = \begin{bmatrix} 13 \\ 6 \end{bmatrix}, x_c = \begin{bmatrix} 12 \\ 7 \end{bmatrix}. \quad (1)$$

Essentially, to assure data consistency, IO values can be arranged in a single vector, as stated in Vector 2.

$$t = \begin{bmatrix} Z_{00} \\ Z_{01} \\ Z_{10} \\ Z_{11} \\ x_0^r \\ x_1^r \\ x_0^c \\ x_1^c \end{bmatrix} \quad (2)$$

where Z_{ij} is the value of $\mathbf{Z}$ at row i and column j, and t is the IO data.

This representation allows for a better manipulation of data, in a sense that, using the aggregation matrix $\mathbf{G}$, it is possible to assure data consistency, by verifying if $\mathbf{G} \times t = 0$.

As such, to obtain this effect in our example, $\mathbf{G}$ is constructed as follows.

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix},$$
$$t = \begin{bmatrix} 8 \\ 5 \\ 4 \\ 2 \\ 13 \\ 6 \\ 12 \\ 7 \end{bmatrix}, h = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 3 \\ 3 \end{bmatrix}, u = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.01 \\ 0.01 \\ 0.001 \\ 0.001 \end{bmatrix} \quad (3)$$

Furthermore, since t contains data from $\mathbf{Z}$, $\mathtt{x}^{\mathtt{r}}$ and $\mathtt{x}^{\mathtt{c}}$, three data sources are used. Additionally, using data uncertainty for each data point would result in the data structures presented in Equation 3.

In summary, a complete balance problem is a system composed of objects: aggregation matrix $\mathbf{G}$; initial IO data vector μ; IO data quality rank vector h; IO data uncertainty vector u.

According to the literature, an arbitrary structure is formulated by IO data arranged in a vector t of size n_t, and n_k accounting identities, described by [7]:

$$0 = \sum_{j=1}^{n_t} G_{ij} t_j + k_i \quad (4)$$

where $\mathbf{G}$ is an aggregation matrix, whose entries are either 0, 1 or -1, and k is a vector of numerical constrains. Additionally, accounting identities can be represented in matrix form as follows:

$$0 = \mathbf{G}t + k \quad (5)$$

However, in practice, the IO data is unbalanced, meaning that t is unknown, and what is available is a vector μ such that:

$$0 \neq \mathbf{G}\mu + k \quad (6)$$

In a nutshell, the objective is to compute t using μ, so Equation 5 is satisfied.

Initially, $\mathbf{G}$, μ and k are known objects, previously computed from IO data tables. Additionally, since IO data is also subject to uncertainty due to empirical measurements. Hence, it is also known an additional vector u of size n_t, which describes the relative uncertainty for each IO data unit.

Finally, to account for multiple data sources and conflicting information, a vector of size n_t is also available at input. Named h, this vector describes the rank of information quality for each IO data point.

In this project, this problem will be addressed using the Weighted Least-Squares (WLS) algorithm

of Rodrigues [7]. The basic idea is to use h to dictate which values will be changed and which will be stay constant. The algorithm starts by changing low quality rank values, and use high quality values as constraints. If no solution is found, the algorithm will evolve and adjust high quality data, computing successive result approximations in an Iterative Weighted Least-Squares (IWLS) process. Once the computation starts, the algorithm sets $t_{(0)} = \mu$ and solves the following [7]:

$$t_{(n+1)} = t_{(n)} + \hat{\mathbf{s}}_{(\mathbf{n})}\mathbf{G}^T\alpha_{(n)} \tag{7}$$

where $\mathbf{G}^T$ denotes the transpose of $\mathbf{G}$ and $\hat{\mathbf{s}}_{(\mathbf{n})}$ denotes a diagonal matrix built from $s_{(n)}$, the reliability index at iteration n. Additionally, $s_{(n)}$ is computed from a Hadamard Product $\odot$ (element-wise product), according to:

$$s_{(n)} = t_{(n)} \odot u \qquad \text{or} \qquad s_{(n)_i} = t_{(n)_i} \times u_i \tag{8}$$

Finally, $\alpha_{(n)}$, the Lagrange Parameter at iteration n, is obtained by computing one of the following methods:

Method 1.

$$\alpha_{(n)} = -(\mathbf{G}t_{(n)} + k) \oslash (\mathbf{G}\hat{\mathbf{s}}_{(\mathbf{n})}\mathbf{G}^T) \tag{9}$$

Method 2.

$$(\mathbf{G}\hat{\mathbf{s}}_{(\mathbf{n})}\mathbf{G}^T)\alpha_{(n)} = -(\mathbf{G}t_{(n)} + k) \tag{10}$$

Method 3.

$$(\mathbf{G}\hat{\mathbf{s}}^{\mathbf{2}}_{(\mathbf{n})}\mathbf{G}^T)\alpha_{(n)} = -(\mathbf{G}t_{(n)} + k) \tag{11}$$

where $\oslash$ denotes element-wise division.

The result from this algorithm is a balanced system, i.e, t is such that Equation 5 is verified.

It is important to notice that three loops are present in this algorithm: the first we refer as the hierarchy loop, since it loops trough data quality ranks from the lowest to the highest; the second is the main loop, called IWLS loop; and finally the third is the must inner loop and has the purpose of computing the next solution approximation.

2.2. Computational Challenges

While the mathematical concepts are not complex, performing such algebraic operations on high dimension objects still present considerable computational challenges. Indeed, due to such extensive objects, matrix and vector storage must be considered, since all operations will benefit from an optimized object manipulation. This is a well known optimization since in 1979 Duchin and Szyld [2] introduced several techniques for large sparse IO models.

Analyzing Equation 7, responsible for computing the next solution approximation, it is easily understood that a matrix-vector product is computed. Furthermore, the computational error, which describes whenever the algorithm should end, computed from $\epsilon = \mathbf{G}t_s + k$, also requires the computation of a matrix-vector product. Since these operations are executed once IWLS iteration, the algorithm requires the computation of at least two matrix-vector products per IWLS iteration.

To further illustrate this analysis, consider an estimation and harmonization of a Quarterly Census of Employment and Wage (QCEW) database example, discussed in the results chapter. This particular real-life case presents an aggregation matrix with 1 million rows per 2 million columns. Our experiments showed that, for a database of such dimensions and initial guesses, applying this algorithm requires around 3500 IWLS iterations to find a solution. Further, consider a maximum of 10 executions per iteration for Equation 7. Hence, the expected number of matrix-vector products in a single execution, p, determined as $p = 3500 \times (10 + 1)$ is $p = 38500$. This indicates a possible optimization challenge as, optimizing matrix-vector products will lead to a better overall performance.

Additionally, depending on the user's choice, the algorithm will compute the Lagrange Parameter α, following Equations 9, 10 or 11. The first performs an Hadamard division, which is a simple point-wise division of vectors. However, the second and third options will solve a system of linear equations, in which α is the solution.

Methods for solving linear systems of equations have been developed with extensive research in the past, since this is a well known operation in IOA and other areas, such as Life Cycle Assessment (LCA) [6]. Considering that solving systems of such dimension demand considerable computational resources, and that α must be computed every IWLS iteration, a conclusion is reached: α computation is one of the most computationally heavy operation in this algorithm and its optimization will have a large impact on the overall computation. Moreover, the algorithm parallelization and work distribution must be considered, since it is intended to run this algorithm in a cluster of machines.

To conclude, to successfully reduce computational time, we intend to focus on the parallelization of matrix-vector products and linear system solving. Considering the arguments above, the parallelization of such operations will heavily impact the overall algorithm performance and successfully reduce computational times.

3. Implementation

Scallability was the problem of greater relevance when developing this project. During testing phases, application performance did not scale as number of processors rise. Analyzing all possible

causes, hypothesis were considered. Moreover, this Chapter also explains how PETSc object representation, such as sparse matrices, can directly lead to these issues.

The first hypothesis for the low scalability was that PETSc was improperly configured, leading to bad performance.

PETSc provides benchmarks for cluster testing. When executed, these programs run for a varying number of machines, estimating the possible speedup each execution.

The cluster is composed by eleven machines, connected using ethernet. As expected, PETSc predicted a linear speedup, until the number of machines reaches eleven. After, the speedup decreases or, in some cases, maintains. In one way, this dismissed cluster configuration hypothesis, and thus, this was not the cause for bad scalability. Additionally, this also estimates the maximum number of machines and maximum speedup achievable using this cluster.

3.1. Load Balancing

Load balancing was the next hypothesis considered. Using the number of entries per row, it was trivial to compute the number of entries each processor had, allowing to see how matrix entries were divided across processors.

Since matrices are divided as chunks of rows, an unbalanced number of entries may lead to some processors need to process more information than others. For a better performance, all machines should have an identical amount of work.

Considering that it is necessary to remove columns and entries from matrices and vectors. This operation can also unbalanced work load, as some machines may have more columns than others, and thus, more matrix entries. That said, this problem can be divided in two sub problems.

3.1.1 Input Data Processing

One of the solutions is to process the input data. Since all data is loaded from disk into memory structures, it becomes more flexible and easy to change.

The approach is to use existent meta-data - number of entries per row - to reorder rows, in a way that all machines have the same amount of entries, in the same amount of rows. To accomplish this, a simple distribution algorithm it is used to assign rows to certain processors, depending on the total entries it processes.

To illustrate this further, consider an analogy of three bags, used to carry objects of different weights. Assigning each object to a bag, the goal is to have as much weight balance as possible across all bags. A simple approach would consist in assigning objects to each bag, considering object and bag weights. More specifically, placing the heaviest available object in the lightest available bag, prioritizing these with fewer objects. Repeating this process iteratively until no object is left, would result in a similar weight and number of objects across all bags.

In this particular case, rows are assigned to each processor, as objects to each bags. Additionally, denser rows are assigned to processors with fewer matrix entries and fewer rows.

In the end, this algorithm will balance work load across all processor as expected.

3.1.2 Object Distribution

The processor's work may become unbalanced after removing entries from vectors. To better understand this issue, consider two vectors, t and h, divided across two machines, `A` and `B`. A correct division would be:

$t =$

i0	i1	i2	i3	i4	i5
1	2	3	4	5	6

$h =$

1	1	1	2	2	2

where the left chunk of each vector belongs to machine `A`, and the remaining chunk to machine `B`.

Applying the same logic, the result would be:

$t =$

i0	i1	i2
1	2	3

$h =$

1	1	1

Notice that, machine `A` has three entries whereas machine `B` has none, since these indexes were be removed. Note that, after created and distributed, an object cannot change, and PETSc does not provide any automated way of preventing this unbalance.

The solution consists in removing indexes from the original data, distributing information without these indexes. At the end of each hierarchical iteration, the result is gathered and reintegrated in a single machine. In posterior iterations, this data is processed, to remove undesired indexes, and redistributed across machines.

This approach will solve the present problem, at the cost of additional communication at the beginning and end, of each hierarchical iteration.

This hypothesis was dismissed as the cause for low scalability, since the desired performance was not achieved after assuring proper load balancing between machines.

3.2. Matrix Structure Analysis

Lastly, matrix structure and object distribution are considered.

Firstly, it is important to consider that this thesis refers to two different matrices: $\mathbf{G}$ is a horizontal rectangular, matrix loaded from the input file. This matrix is used in matrix-vector products, and matrix-matrix multiplications; and $\mathbf{A}$, a square matrix, computed following $\mathbf{A} = \mathbf{G}\hat{\mathbf{s}}_{(\mathbf{n})}\mathbf{G}^T$. This matrix specifies the equations of a linear system, which is later solved, using Generalized Minimal RESidual (GMRES) or Conjugate Gradient (CG) iterative methods.

Matrices and vectors are naturally distributed across all machines, meaning that, operations such as matrix-vector products, which requires the use of complete vector in all processors, leads to machine communication.

The strategy to reduce communication overhead consists in performing local computations while communicating. To further explain this point, one must first understand how matrices and vectors are stored in PETSc. Matrices are stored in a two sub-matrix structure: one sub-matrix will store all values belonging to the matrix diagonal blocks, and another sub-matrix for the remaining entries. Furthermore, vectors are stored in dense format, distributed across machines as chunks of contiguous values.

3.2.1 Matrix-Vector Product Scalability

Distributed matrix-vector products are computed each IWLS iteration, using matrix $\mathbf{G}$. Once performing this operation, each processor starts by broadcasting its vector chunk. In other words, all nodes exchange its vector chunk, until the entire cluster has a complete copy of the same. However, while communicating, each processor starts computation using its local vector values, by multiplying its local vector chunk per its local sub-matrix holding diagonal values.

Analyzing this operation, it is possible to conclude that, matrices with denser diagonal blocks and sparser non-diagonal blocks, will require longer for each processor to compute its local part. As a consequence, local computational time will dominate the operation overall time, providing longer times for communication. In summary, the more diagonal the matrix, the less communication overhead.

This key idea will dictate the difference between scalable and non-scalable matrix-vector products. It also leads to one core conclusion: each problem should be analyzed so $\mathbf{G}$ is changed using the optimal permutation. Different problems have different matrices, and thus, each matrix structure requires a different permutation to reach an optimal non-zero pattern.

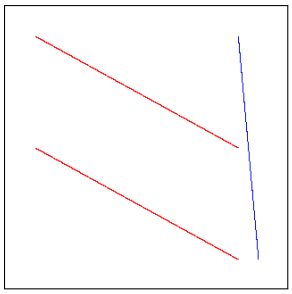

Figure 1: Sparse bi-diagonal rectangle matrix.

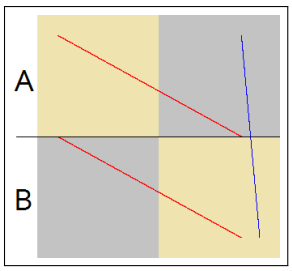

Figure 2: Sparse bi-diagonal rectangle matrix divided across two processors, A and B.

During the course of this project, it was possible to find a possible permutation for one type of matrix. Consider the matrix structure presented in Figure 1, in which each dot represents a non-zero matrix entry.

This matrix shows two sparse diagonals, in red, and a denser one, in blue. Distributing this matrix across two processors would result in the division presented in Figure 2. Both processors, `A` and `B`, contain the same amount of rows and same diagonal block sizes, presented in yellow.

However, one can see that some values in non-diagonal matrix, presented in gray. Indeed, for this particular matrix, 52% of entries are on the diagonal block for processor `A`, and 47% for processor `B`.

To transform this matrix into a more diagonally heavy one, it would be necessary to permute rows. The permutation would simply interlace the bottom part rows, with the top part rows. In other words, for a matrix with N rows, counting from the top, the permutation would sort rows similar to 0, $N/2$, 1, $(N/2)+1$, 2, $(N/2)+2$, and so on. The result is a diagonal denser structure, presented in Figure 3.

Note that, with this permutation, the matrix is composed of one dense diagonal, presented in red, and two sparse diagonals presented in blue.

This trick allows for denser diagonal blocks, containing 95% and 90% of entries for processors `A` and `B`, respectively, in this case. As a result, computational times decreased for this specific test case.

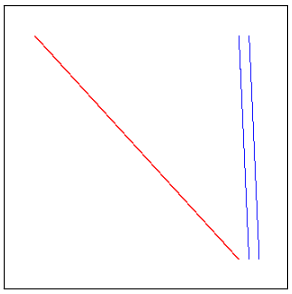

Figure 3: Permuted sparse bi-diagonal rectangle matrix.

3.2.2 Linear Solver Optimization

Yet another concern is to optimize the linear system solver. As expected, iterative linear solvers, such as GMRES and CG, will convergence faster if input matrix shows a highly diagonal structure.

Two possible solutions were used to diagonalize **A**: permute **A** using an algorithm such as Reverse Cuthill-McKee (RCM); or permute **G** rows so **A** is computed as permuted.

PETSc provides implementations for RCM and other permutation algorithms, that could be applied to permute **A** when created.

However, applying these permutations on distributed matrices leads to a local permutation on each processor. In other words, to avoid large communication overhead, PETSc will not permute lines between machines. The result is a permuted matrix which structure varies with the number of machines, losing the algorithm effect for large clusters. Another problem with this solution is that the matrix will be permuted each time it is created. A in a nutshell, a permutation is performed each IRLS iteration, requiring additional time.

The goal is to obtain **A**, distributed across all machines, and permuted as if in a single machine. To accomplish this, one should create **A** locally to a single processor, compute the permutation matrices and apply those to **G**. The result is a distributed **A**, computed as if permuted in a single machine. Using this technique it is also avoided to permute **A** once created, optimizing the overall process.

3.2.3 Overall Scalability

Since **G** is permuted in a way that **A** is diagonal, **G** becomes less desirable regarding non-zero structure, leading to a bad scalability in matrix-vector products. In other words, optimizing linear solver operation deteriorates matrix-vector product performance. Optimizing structure of **G** for matrix-vector products does not assure diagonal structure of **A**.

In order to have a diagonal **A** and a good **G**, it is necessary to use two **G** matrices: one permuted into a more diagonal heavy matrix, for matrix-vector products; and a second to **A** computation.

The only shared point between these two matrices is a single best guesses vector. However, this object would not require any extra communication, in a sense that it could be used by both matrices without any extra permutation.

3.3. Meta-data Distribution Optimization

For a better matrix assembly, PETSc requires meta-data, so the proper structures can be created.

The problem is that, when loading the input file, objects are yet to be created, and since diagonal block sizes change with matrix dimensions and processor number, it is not possible to know which values will be part of the diagonal or non-diagonal submatrix. In other words, it is necessary to know the sizes of each processor chunk do separate values into these categories.

This lead to the creation of two extra meta-data structures to count the number of non zeros per row that belong to the diagonal, and non-diagonal blocks. In a nutshell, after file loading and matrix creation, the application processes each matrix entry to determine of it belongs to any processor's diagonal block. The algorithm shows complexity $O(nnz \times np)$, where nnz is the number of matrix entries, and np number of processors. Naturally, this solution becomes less viable as number of matrix entries or machines increases.

Yet another problem is that, each processor needs to preallocate the structures for its part of the matrix values, meaning meta-data must be distributed across the processors. To accomplish this, the matrix global and local dimensions are used so the root machine can calculate which chunk of values belongs to each processor, and distribute the meta-data along the cluster.

4. Results

Evaluation is a very important phase in a project. In this part, the software is tested to investigate either fulfills the requirements, or not.

For this particular project, it is desirable that the algorithms run faster than sequential ones. On top of that, it should provide the same, if not better, results as the ones from an uniprocessor architecture. Further more, it should be capable of handling growing amount of work and do not use unnecessary memory.

4.1. Overview

Summing up, two qualities are desirable - performance and scalability.

The time a program takes to deliver a solution is one of the most important qualities, when developing a parallel scientific application. Indeed, one

of the main goals of using multi-processor architectures, shared or distributed memory, is to obtain faster results, compared with uniprocessor environments.

Since that is one of the main goals of this project, the final product should be evaluated in its performance.

To do so, the computation should be timed and compared with existing sequential solutions.

When developing parallel programs, scalability is another main concern in engineering. Scalability is very close related to performance, and means that, the program should be able to handle growing amount of work, and should be able to adapt in a capable manner to an enlargement to accommodate the growth of work.

This is a very important quality, since a bad scalability often reflect in a bad performance in large environments, usually due to the overhead in communication between identities or memory requirements.

As such, the memory requirements and the number of communications should be monitored.

4.2. Methodology

To test the parallel application, it was available an eleven machine cluster, configured with MPI. It was also possible to use various types of input problems, from real cases to artificial generated. These problems differ in topology, and most importantly, matrix structure.

Real case problems were available as octave data files. These files vary in problem size, allowing for debug usage or performance measurements.

Regarding artificial files, these were created using an octave program. These problems could be generated with various sizes, providing the expected solution for comparative purposes.

The application ran using `mpirun` command, providing the appropriated arguments.

4.3. Problem Topologies

Four problem topologies were tested. In practice, these problems vary in matrix structure. The tested topologies are listed in Figure 4. In each picture, a red dot represents a positive number, whereas a blue dot represents a negative entry.

Additionally, it is important to note that, topologies `A` and `B` are from BEA. Topology `D` is a QCEW problem and `C` is an artificial generated problem. Note that, since `Type C` topology is artificial generated, matrix non-zero values are equally distributed across rows.

4.4. Results

This section presents the results from various experiments. Each problem was executed using an increasing number of processors. In order to achieve

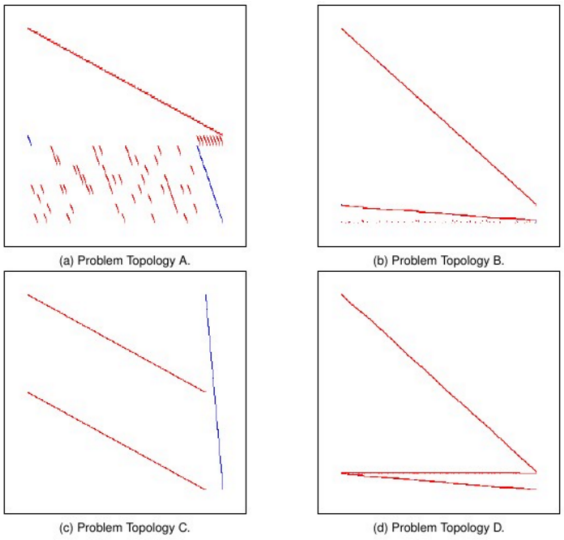

Figure 4: Different problems topologies used.

Type	Columns	Rows	Entries
A	6 840	2 286	15 792
B	364 572	73 998	799 830
C	8 800 000	800 000	16 800 000
D	2 039 681	1 098 939	5 171 901

Table 2: Comparative view between problem topologies.

computation scalability various hypothesis were considered. Hence, in this section we show the results from each optimization.

The results are presented in Figures 5, 6, 7, 8 and 9.

Computing in normal execution, i.e. without any changed in matrix structure or optimization, the results were as shown in the figures, as blue column. Note that, some values are not set. In these cases, execution took too much time and was aborted. Furthermore, note that `Type D` test case contains two different problems for the same topology, differing only on the IO data values. As a result, two columns are used for `Type D` execution times, each

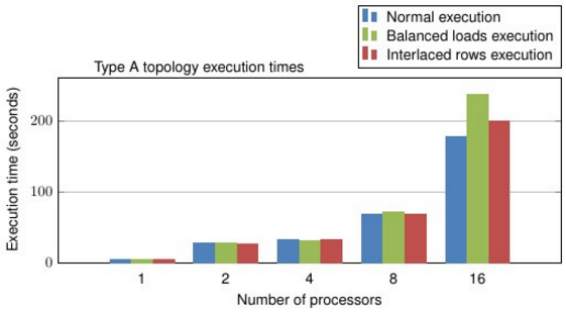

Figure 5: Type A topology execution times.

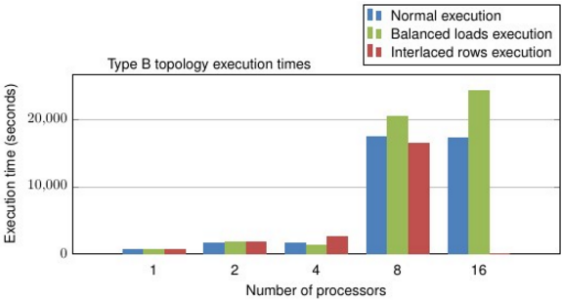

Figure 6: Type B topology execution times.

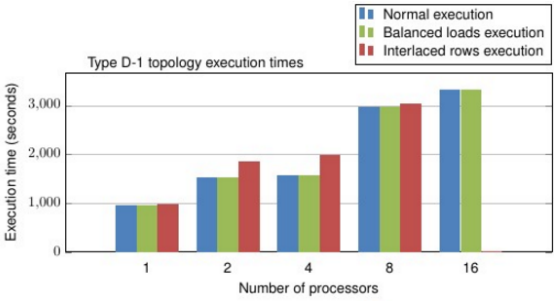

Figure 8: Type D-1 topology execution times.

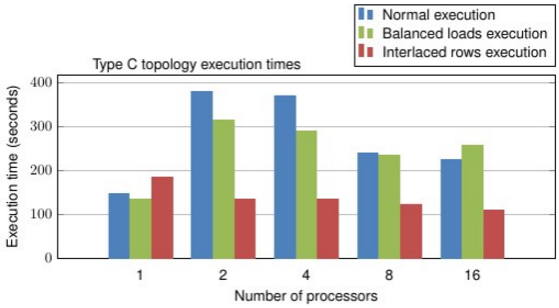

Figure 7: Type C topology execution times.

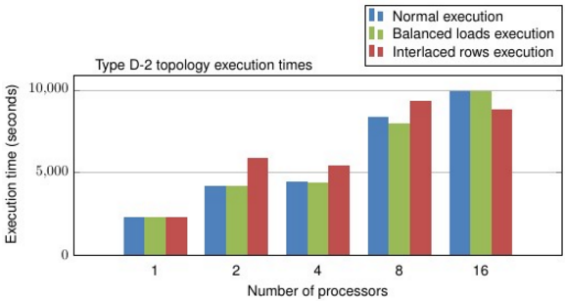

Figure 9: Type D-2 topology execution times.

referring to a different problem.

It is also important to note that, our results only display execution times for method 1 of our algorithm.

From the normal execution times, it is understood that our application does not scale well for increasing numbers of processors. In these cases the input data is not equally distributed across machines, limiting the maximum speedup achievable. Moreover, processor diagonal sub-matrices are typically not dense, reducing the potential parallelism.

Our next optimization equally distributes matrix rows and non-zero entries across all machines. The goal is to assign the same amount of work for each processor, maximizing the potential parallelism. The results are shown in the charts as the green column. `Type C` topology got better results, comparing with normal execution. However, `Type B` problem became slower for 8 processors. The remaining topologies had no change in execution times.

Considering the discussion in Section 3.2.1, and since this optimization does not account for the density of diagonal sub-matrices, the results were expected not to scale well. Additionally, `Type C` topology showed similar results to normal execution. This is expected because, since matrix non-zero entries are equally distributed across rows, processors will have the same amount of work without and balance needed.

Hence, the finally optimization we propose has the goal of transforming `Type C` matrix structure into a more diagonally heavy one. This is accomplished by interlacing matrix rows, previously addressed in Section 3.2.1. Using this optimization, `Type C` topology got better execution times and scalability. The red columns in the charts shows the resulting execution times for several problems.

This is expected because, as addressed in Section 3.2.1, diagonal sub-matrices became more dense, and non-diagonal sub-matrices became more sparse. This means that, by changing matrix structure, parallelism potential for matrix-vector products had increased.

Furthermore, since the number of matrix entries in diagonal matrix decreases with increasing number of processors, performance deteriorates after number of machines increases to 8 and 16.

5. Conclusions

This thesis attempts to provide a parallel implementation of a WLS algorithm of Rodrigues [7] to balance IO data, in order to reduce processing times. Accordingly to WLS algorithm of Rodrigues, three district methods to balance input data are supported. All methods share the same algorithm logic, differing on the method used to compute a Lagrange parameter α, needed each WLS iteration. Method 1 uses a simple element-wise division between two vectors. Methods 2 and 3 require the solution to a sparse linear system of equations. Essentially, the algorithm requires the computation of multiple matrix-vector products, as well as matrix-matrix products. In conjunction with linear sys-

tem solving, these are the main computational challenges in this project.

To support our implementation we decided to use the library PETSc, since it provides parallel implementations for all operations needed in this work. This was the best option because we did not intend to develop software that already existed, specially when these libraries provide well optimized functionality. Using PETSc we implemented our algorithm, targeting distributed memory machines.

The main challenge found when developing our program was the low computational scalability. Indeed, when testing, our application did not perform for an increasing number of machines. For this reason, three optimizations were considered and tested: the first balanced the number of matrix entries and rows, equally across all machines, attempting to assign the same work load in all machines; the second optimization focused on diagonalize the input matrix, changing its structure intro a more appealing format for palatalization; finally, the third optimization focused on permuting the input matrix with the goal of changing the sparsity of its diagonal and non-diagonal sub-matrices. With this we attempted to take advantage of PETSc distributed matrix structure to obtain better results in parallel matrix-vector products.

These tests helped us understand which matrix permutations work best for each case study. In a nutshell, different matrices require different permutations to achieve computation scalability. As such, from this analysis, we manage to obtain better results using the third optimization, for a case study with bi-diagonal matrix.

5.1. Future Work

Despite our attempt to find a scalable, multiprocessor solution for this problem, our efforts showed to be insufficient for all topologies. We think it is possible to improve computational performance using adequate transformations for each matrix type. Since our application does not scale for many problem topologies, it is important to research permutations to transform matrices into a more parallel efficient structures. This aspect is important as it dictates the difference between scalable and non-scalable executions.

Another issue to be investigated is the linear system solving. With our implementation we could not use algorithm methods 2 and 3 to find a solution for test problems because linear system solving was not providing the expected solutions. Future work should focus on finding a reason for this unwanted behavior.

Additionally, other important aspect to complete our solution is to account for ill composed input data Essentially, our algorithm does not account for input data with singular matrices. At present time, the user is responsible for eliminating the redundant rows from the input matrix. However, it is desirable that input data should be processed in order to make it usable by the application

Acknowledgements

Since the very beginning, this work counted with the help of various people. It is possible to say that, without those people, this work would not be possible. It is also important to note that, help did not always came from a technical perspective, but also from a word of encouragement and motivation.

I would like to thank professors José Monteiro and João Rodrigues, for the guidance, continuous help, advice and availability. During the development of this work their enthusiasm and ideas were very motivating and challenging.

Similarly, I would like to show my gratitude towards my course and work colleagues, for the exchange of ideas and motivational words in frustration times.

Last, but not least, I would like to express my gratitude towards my dear parents, brother and girlfriend. They were tireless during all my academic progress, providing an excellent environment so I could achieve this milestone and be prepared for future challenges.

References

[1] Petsc - portable, extensible toolkit for scientific computation. http://www.mcs.anl.gov/petsc/, 2015.

[2] F. Duchin and D. B. Szyld. Application of sparse matrix tecniques to inter-regional input-output analysis. *Economics of Planning*, 15:142–167, 1979.

[3] J. Kitzes. An introduction to environmentally-extended input-output analysis. *Resources*, 2(4):489, 2013.

[4] M. Lenzen, B. Gallego, and R. Wood. Matrix balancing under conflicting information. *Economic Systems Research*, 21(1):23–44, 2009.

[5] R. E. Miller and P. D. Blair. *Input-Output Analysis: Foundations and Extensions*. Cambridge University Press, 2009.

[6] G. Peter. Efficient algorithms for life cycle assessment, input-output analysis, and monte-carlo analysis. *Int J LCA*, 12 (6):373–380, 2007.

[7] J. Rodrigues. A bayesian approach to the balancing of statistical economic data. *Entropy*, 16:1243–1271, 2014.