AVU-GSR Gaia Mission. An hybrid solution for HPC and Grid-MPI infrastructures

M. Bandieramonte(1,2)  
U. Becciani(2)  
Dept. of Physics and Astronomy, University of Catania(1)  
Astrophysics Observatory, INAF(2), Catania, Italy  
Email: {marilena.bandieramonte, ugo.becciani}@oact.inaf.it

A. Vecchiato, M. Lattanzi  
and B. Bucciarelli  
Astrophysics Observatory - INAF  
Torino, Italy  
Email: {vecchiato, lattanzi, bucciarelli}@oato.inaf.it

Abstract—Gaia is an ambitious space mission of the European Space Agency which will chart a three-dimensional map the Milky Way to study the composition formation and evolution of our Galaxy. Our research team is developing the AVU-GSR verification module, aiming to obtain a reconstruction of the celestial sphere using a subset of GAIA observations. The authors propose a hybrid solution for HPC and Grid-MPI infrastructures which utilizes a modified LSQR – a conjugate gradient-based algorithm – to solve the system of equations for the sphere reconstruction. The proposed solution has been selected as pilot test for porting HPC applications in Grid.

Index Terms—LSQR; sparse linear matrix; Grid; HPC

I. INTRODUCTION

Gaia is an ESA (European Space Agency) cornerstone mission, expected to be launched in mid-2013. The main goal of this mission is the production of a 5-parameters astrometric catalog (i.e. including positions, parallaxes and the two components of the proper motions) at the μarcsecond-level for about 1 billion stars of our Galaxy, by means of high-precision astrometric measurements conducted by a satellite sweeping continuously the celestial sphere during its 5-year mission. From a mathematical point of view, the satellite observations translate into a large number of equations, linearized with respect to the unknown parameters around known initial values, while the catalog production, which is called Global Sphere Reconstruction, requires the solution of the resulting linear system. The number of observations is much larger than that of the unknowns, so that the solution of the system in the least-squares sense eventually provides the catalog with its errors. In the Gaia mission these tasks are done by the Astrometric Global Iterative Solution (AGIS) but, given the absolute character of these results, the DPAC (Data Processing and Analysis Consortium, i.e. the international consortium which is in charge of the reduction of the Gaia data) decided to prepare a verification module which produces an independent sphere reconstruction using a subset of the Gaia observations. This is called AVU-GSR. The proposed application is for the development and testing of the part of the AVU-GSR software (Astrometric Verification Unit - Global Sphere Reconstruction) which is most demanding in terms of computational resources and at the same time most complex from the point of view of a parallelized problem.

The uniqueness of the problem stands on several factors, the main being the system dimensions which are of the order of \(10^{10} \times 10^8\). A brute-force solution of such system would take about \(10^{27}\) FLOPs, a requirement which cannot be decreased at acceptable levels even taking into account the sparsity ratio of the reduced normal matrix, which is of the order of \(10^{-6}\). It is therefore necessary to resort to iterative algorithms. By using additional hypotheses on the correlations among the unknowns which are reflected on the convergence properties of the system, AGIS implements a block-iterative adjustment of the astrometric, attitude, instrument calibration, and global parameters, allowing the use of an embarrassingly parallel algorithm. The starting hypotheses, however, can hardly be proved rigorously, and have only been verified “a posteriori” by comparing the results with simulated true values - a situation which cannot hold in the operational phase with real data. Moreover, such method does not provide the analytical covariance between the different types of unknowns, which constitute another unique characteristic of this problem. These considerations lead to the choice of an independent approach as the one adopted by AVU-GSR, which uses a modified LSQR – a conjugate gradient-based algorithm – to solve the system of equations. In its baseline configuration, AVU-GSR is meant to handle a more limited set of stars and observations than the one treated by AGIS.

The paper is organized as follows: section II illustrates the adopted solution for the parallelization of the Sphere Reconstruction problem; section III describes the data model and parallelization techniques; sections IV and V present a few results on the performance of the proposed solution and some discussions on the porting on Grid-MPI infrastructures, respectively; finally, conclusions and future works are given in section VI.

II. THE SYSTEM OF EQUATIONS OF THE SPHERE RECONSTRUCTION AND ITS SOLUTION ALGORITHM

The goal of AVU-GSR is to produce a Global Sphere Reconstruction using a subset of the Gaia observations. We employed a modified version of the PPN-RAMOD model used in [1] in which:

- space-time is represented by the PPN approximation of the Schwarzschild metric of the Sun;
• observations are the Gaia-like abscissae along the satellite scanning direction, computed with respect to the satellite’s reference frame;
• unknowns are:
  - the astrometric unknowns, represented by the spatial coordinates of the stars along with their proper motions:
    \[ \varpi^* = a_\beta/r^*, \quad \alpha^*, \quad \delta^*, \quad \mu_{\alpha^*} = d\alpha^*/dt, \quad \mu_{\delta^*} = d\delta^*/dt; \]
  - the attitude unknowns, given by an appropriate B-spline representation of the Rodrigues parameters of the satellite covering the whole mission duration [2];
  - the instrumental parameters;
  - the global parameter \( \gamma \) of the PPN (Parametrized Post-Newtonian) formalism, used to test General Relativity against other alternative theories of gravity.

In general, an astrometric model like the above results in a non-linear equation

\[
\cos \psi_i \equiv F_i \left( \begin{array}{c}
\alpha_s, \delta_s, \varpi_s, \mu_{\alpha_s}, \mu_{\delta_s},
\end{array} \right)
\]

expressing the \( i \)-th observed abscissa \( \psi_i \) as a function \( F_i \) of very few (typically some tens out of \( 10^9 \)) of the unknowns of the problem, and several billions of such equations (observations) are accumulated during the five years of the mission lifetime. Therefore, the final result will be a large and sparse system of equations of the order up to \( \sim 10^{10} \times 10^8 \) in the case of Gaia.

Since the solution of such systems of non-linear equations is not practially feasible, the observation equations (2) are linearized about a convenient starting point, represented by the current best estimation of the unknowns, by a first-order Taylor series.

\[
-\sin \psi_i \, d\psi_i \equiv \frac{\partial F_i}{\partial \alpha_s} \delta \alpha_s + \frac{\partial F_i}{\partial \delta_s} \delta \delta_s + \cdots
\]

The problem is therefore converted into that of the solution of a large and sparse system of linear equations

\[
b = Ax
\]

where \( b = \{- \sin \psi_i \, d\psi_i \} \) is the vector of known terms obtained from the satellite measurements. The vector of the unknowns \( x \) is formed by the corrections \( \delta x_j \) to the starting catalog values \( \bar{x}_j \), while matrix \( A \in \mathbb{R}^{m \times n} \), which we will call Total Matrix, has \( m \gg n \) (the number of equations is much larger than that of the unknown parameters) whose elements are

\[
a_{ij} = \left. \frac{\partial F_i}{\partial a_j} \right|_{\bar{a}_j}.
\]

As already mentioned, despite this simplification the system is still too large to be solved with direct methods. The number of unknowns, in fact, is mainly driven by the number of stars \( n_s \) included in the system, and by the accuracy with which the attitude has to be reconstructed, the latter determining the number of degrees of freedom associated to the satellite attitude. The characteristic dimensions of the matrix associated to this system can vary according to the specific goal of the corresponding sphere reconstruction. For this reason the number of stars can be \( 10^9 \lesssim n_s \lesssim 10^9 \), while the degrees of freedom \( n_d \) of the attitude can range between \( \sim 10^6 \) and \( \sim 3 \times 10^7 \). Since each star has up to 5 unknowns (three components for the position and two for the proper motion) and each degree of freedom for the attitude carries three unknowns, the total number of unknowns \( n \) can be roughly estimated as

\[
10^7 \lesssim n = 5 n_s + 3 n_d \lesssim 6 \cdot 10^8.
\]

The number of observations \( m \), instead, is simply proportional to \( n_s \), since each star will be observed an average of \( 7 \cdot 10^2 \) times at the end of the 5 years of the foreseen mission lifetime, and therefore we have

\[
7 \cdot 10^8 \lesssim m \lesssim 10^{11}.
\]

Moreover, the number of non-zero coefficients of the matrix \( A \) can be easily estimated considering that each observation will involve less than 30 parameters, for a total number of non-zero coefficients of

\[
2 \cdot 10^{10} \lesssim \text{ncoeff} \lesssim 3 \cdot 10^{12}.
\]

To solve such a system, the authors use a hybrid implementation of PC-LSQR [3], an iterative method for solving large and sparse linear equations, with the aid of some parallelization techniques and of an ad-hoc compression algorithm of the sparse system matrix \( A \).

The LSQR method was originally proposed by Paige and Saunders [4] and it consists of a conjugate-gradient type algorithm which is equivalent to compute, at each iteration \( (i) \), an approximate solution

\[
x^{(i)} = (A^T A)^{-1} A^T b^{(i-1)},
\]

and then evaluates the vector of residuals

\[
r^{(i)} = b - Ax^{(i)}
\]
which has to be minimized in the least-squares sense, according to suitable convergence conditions defined by the algorithm itself. Among the possible stopping conditions we have:

- the residual vector has a 2-norm lower than a threshold value
- a fixed maximum number of iterations is reached

The PC-LSQR method uses a pre-conditioning technique, which basically consists in a renormalization of the columns of $A$, made to improve the speed of convergence of the system.

As reported in [4], the number of multiplications per iteration needed by this algorithm is $3m + 7n$; therefore, from this consideration and from the quantities of formulae (4) to (6), it is easy to understand that this problem calls for the use of parallel computers even in the smallest case which is going to be treated for the Gaia Sphere Reconstruction.

The data reduction and the data distribution strategy of our implementation will therefore be targeted to limit as much as possible the memory requirement and at the same time to minimize the communications between the processes, as described in the following sections.

III. DATA MODEL AND PARALLELIZATION STRATEGY

For each observation, the Total Matrix stores the astrometric, attitude, instrumental parameters and a Global Value coefficient. As it is showed in Fig. 1, $A$ is a sparse matrix, being populated primarily with zeros. It can be divided into four different sub-blocks:

- the first one contains the astrometric parameters: 5 astrometric coefficients are foreseen for each observation and all the observations related to the same star are placed in consecutive rows, thereby producing a block-diagonal structure for this sub-block;
- the second one refers to the attitude unknowns: 12 different coefficients are placed in three equally spaced groups, each containing four non-zero elements
- the instrumental parameters: for each row 6 non-zero instrumental coefficients are spread among a long series of zeros
- at the end there is the Global Value that represents the relativistic $\gamma$.

A. The data reduction

The particular structure of this matrix lead the authors to design an ad-hoc compression algorithm, which translates the original matrix into three mono-dimensional vectors: $SystemMatrix$, $MatrixIndex$, $InstrIndex$.

- $SystemMatrix$: it is a double-precision vector that contains only the non-zero coefficients for each observation. In conclusion, there are only 24 coefficients for a single observation: 5 astrometric parameters of the observed star, 12 attitude parameters, 6 instrumental parameters and one Global Value, and each row of $A$ is translated into a sequence of 24 values of $SystemMatrix$.
- $MatrixIndex$: it is a vector of indices pointing to the position that each element of $SystemMatrix$ had in the original $TotalMatrix$. It is made of two long: the first one is used to recover the index of the first astrometric parameter different from zero; the second one to keep track of the position of the first non-zero attitude parameter.
- $InstrIndex$: it is used to determine the position of the instrumental parameters by means of the formulas from which its elements are derived. It is composed of four int that represent: the Field of view (FoV), the CCD, the pixel and the time interval.

We distribute the Total Matrix (Fig. 1) between the Processing Elements (PEs) of the parallel application so that each task has a continuous set of rows (observations). The main advantage of using this kind of compression is that it is possible to split the satellite data into different observation files that are placed in a shared filesystem, so every Processing Element (PE) can access them in parallel because they are independent. It could happen sometimes that two different PEs have to access the same file, but this is not a problem for the application because usually the observations read by a PE are not in the same file read at that time by any other PE.

B. The Data and the Load distribution

The purpose of this task is to distribute the data uniformly among the available physical memory. It represents the most important issue for the parallel code and it is done to maintain the computation as local as possible, in order to maximize the performances. The larger arrays are distributed, while other smaller arrays and solutions are replicated on each PE. Thanks to the ad-hoc compression algorithm designed for the Total Matrix $A$, we are able to distribute $A$ among processes with the three vectors $SystemMatrix$, $MatrixIndex$ and $InstrIndex$: each process has an equal portion of the system given by the ratio indicated in the Fig. 2. For example, each PE will have a portion of $SystemMatrix$ with:

$$\frac{nObs \times nParam}{nPEs}$$

(7)

where $nObs$ is the number of observations $nParam$ is the number of parameters and $nPEs$ is number of processes. With this subdivision a subset of stars is assigned to each PE.
to perform the LSQR computation. The known-term vector $b$ is distributed with a congruent map:

$$\frac{n\text{Obs}}{n\text{PEs}}$$ (8)

The unknown terms $x$, the preconditioning vector $x_{\text{preCond}}$ and the other vectors used to determine the solution (five in total) are partially replicated on each PE. For example, the $x$ vector portion necessary to maintain the computations as local as possible consists of:

- the attitude parameters
- the instrumental parameters
- the Global parameter

These portions are replicated on each PE, whereas the first portion, which is proportional to the number of stars that are computed by a single PE, is regularly distributed. Auxiliary arrays are distributed or replicated with the same above-mentioned logic. Using this kind of distribution, duplicated data are very small arrays in comparison with distributed arrays for large star numbers. We have to underline that duplicated arrays are updated and re-distributed at each iteration, with a typical MPI AllReduce call to combine all contributions. The load balancing among the processes (PEs) is very important in order to maximize performance, and with the data distribution and the replicated data described above, not only a balanced load is obtained, but each PE can run using local data only at each iteration. After the local computation is ended, a reduction operation is done to accumulate the overall results. The load balancing for the computational phase is ideal since each PE works on the same portion of data.

**C. Memory Request**

The memory request to solve the AVU-GSR module depends on the number of stars, the number of observations and the number of PEs available in the system. The overall requested memory is calculated as the sum of the distributed arrays and the replicated ones. Assuming to have $n^* = 10^7$ stars and to observe each star 720 times, the number of observation will be $n\text{Obs} = 720 \times 10^7$.

The replicated portion of data is fixed between 0.35 GB and 0.71 GB per task for each vector used to determine the solution (there are five of them). The difference between these two values is in the sampling interval of the attitude parameters: in the first case $\Delta T\text{Attitude}=10$ sec., in the second one $\Delta T\text{Attitude}=5$ sec., so at the end of the mission, assuming to observe $n^* = 10^7$ stars, we would have 5*0.35= 1.75 GB or 5*0.75=3.55 GB of replicated data on each CompNode.

The distributed data depends on the number of tasks and on the number of PEs. Still, by the end of the mission we will have:

- System_Matrix 1.287/nPEs GB
- Matrix_Index 107/nPEs GB
- Instr_Index 107/nPEs GB
- KnownTerms 54/nPEs GB

of distributed data on each PE.

In Fig. 3 we report how the total required amount of memory varies with the number of stars observed. The graph shows memory occupancy with changes in the number of stars and in the number of Computing Nodes having a fixed number of threads - equal to four - and $\Delta T$ of 10 sec for the attitude. The trend seems to grow in a reasonable way and memory requirements scale well with the number of Computing Nodes. As the number of stars grows, it becomes mandatory to increase the number of Computing Nodes because otherwise the required memory would be too high (above 16 GB per node). Using a computing node with 4GB of RAM we should limit to five million stars with a number of Computing Nodes at least equal to 512; however, in the operative phase of GAIA we expect to use the BlueGene/Q (BGQ) supercomputer system, which will be released during 2012, and which will provide 16 GB memory per node. The graph shows that with a BGQ system the AVU-GSR could be able to solve a system of 100 million stars with 1,500 Computing Nodes.

In Fig. 4 we plot the RAM memory request for a run as function the number of stars observed (5, 30, 50 and 100 million of stars) and of the sampling period of the attitude parameters $\Delta T\text{Attitude}$ (5/60, 6/60 and 10/60). The calculations have been carried out assuming to have 1,500 nodes. This configuration is significant in order to assess the problem because at the end of the mission we will have 1,500 nodes available to run the application. It can be seen that with a BGP system, which has 4 GB RAM, it is possible to elaborate up to 5 million stars with $\Delta T\text{Attitude}$ equals to 10/60, whereas with a BGQ system the application could even work with 100 million stars. This is an important result considering that our target is to reach at least 10 million

<table>
<thead>
<tr>
<th>Distributed Vector</th>
<th>Number of elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>System_Matrix</td>
<td>$n\text{Obs}^* n\text{PEs}$</td>
</tr>
<tr>
<td>Matrix_Index</td>
<td>$(n\text{Obs}/n\text{PEs})^*2$</td>
</tr>
<tr>
<td>Instr_Index</td>
<td>$(n\text{Obs}/n\text{PEs})^*4$</td>
</tr>
<tr>
<td>Known Terms</td>
<td>$n\text{Obs}/n\text{PEs}$</td>
</tr>
</tbody>
</table>
stars, and possibly up to 30-50 million. The system responds well to the changes and the memory requirement decreases consistently with decreasing sampling frequency.

D. Code portability

The use of MPI allows us to run the code on the most common computational resources. With the adopted strategy for data distribution we mainly need only some synchronism points (MPI_Barrier), some reduction operations (MPI_Allreduce mainly in the LSQR algorithm), and few data communications (MPI_Bcast). Also OpenMp is used where available, i.e. BGP and BGQ architecture and on the INFN Pisa Grid data center.

IV. THE PERFORMANCES

This section briefly discusses the results obtained by preliminary simulations to evaluate the AVU-GSR software. As said before, the aim of the module is to solve a system of at least 10 million stars in an acceptable time. The algorithm has been tested and works on IBM BlueGeneP (BGP) type system with 512 nodes, but during the operational phase of the mission we expect to run it on 1,500 nodes of a IBM BlueGeneQ system constituted by 10,000 nodes in total, 8,500 of which dedicated to large scientific runs. Fig. 5 shows the trend of the execution time (ET) of an iteration (in seconds) as function of the number of Computing Nodes and the type of system (with SMP or without SMP) and the number of stars. The considered scenarios make reference to a IBM BlueGeneP supercomputer. Each BlueGeneP computing chip contains four PowerPC 450 processor cores, running at 850 MHz. The cores are cache coherent and the chip can operate as a 4-way symmetric multiprocessor (SMP). A computing card contains a BlueGeneP chip with 4GB DRAM, constituting a Computing Node (CompNode). For the measurements the authors considered three different configurations:

- 256 computing nodes without SMP - 1 task per node, 1 thread per node
- 256 computing nodes with SMP - 1 task per node, 4 threads per node
- 512 computing nodes with SMP - 1 task per node, 4 threads per node

As it is well known, the speed-up of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program. From the system behavior one can notice the presence of a sequential fraction of the program, which does not take advantage of parallelization, amounting to approximately 10 seconds of computing time. This is due to some serial operations we must perform for the computation of attitude parameters. Subtracting this offset to each result, the computing time show a behavior approaching the theoretical maximum speed-up curve: as it can be noticed from the Fig. 5, by duplicating the number of CompNodes, the time is halved. Moreover, the switch from 1 to 4 threads resulted in an improvement of up to 25% for larger number of stars. This is due to the fact that the iterations of the LSQR algorithm mainly consist of two operations: one that calculates $r^{(i)}$, with indices depending on the observation number and therefore completely parallelizable; the other one that computes $x^{(i)}$: since this can be potentially critical (due to index overlap), at each iteration we must use a critical section which serializes the threads. Thus, 50% of each iteration is perfectly parallelizable with openMP and 50% is in fact serialized. Since 95% of the workload comes precisely from these two operations, the measured computing time obtained with above considerations make a perfect match. For this reason, we can also make reliable and improved predictions for BGQ. From the analysis carried out it can be argued that the system shows an extremely low communication level between the processes, as it was desirable, and that it has good scalability properties, keeping a good performance with increasing computational load. Nevertheless, the number of stars which can be processed is limited by the main memory requirements for each node. BGP can provide up to 4 Gb/CompNode which makes it possible to process $n^* = 3 \times 10^6$ stars only using 512 computing nodes at least. The Fig. 6 shows a projection of the ET of an iteration on BGQ system: each BGQ compute chip contains 16 PowerPC A2 processor cores used for computing, 4-way simultaneously multithreaded (SMP), running at 1.2 GHz with 16 GB DRAM. The projection was obtained by scaling the measured values by a factor of BGQ/BGP in GFlop/s considering the following
values: 3.4 GFlop/s per core BGP and 12.81 GFlop/s per core BGQ. The values obtained for 3 million stars were then projected on 30, 50, 75 and 100 million stars and scaled to the number of processors considering the trend shown in the chart above and using a conservative factor of 20% lower than the estimated one. The figure shows a projection of the costs of executing a single iteration: considering $3 \times 10^6$ stars and 750 nodes, we will have ET for an iteration $\sim 55$ sec and then assuming to obtain a solution in 3,000 iterations we get a time equal to 55 sec $\times 3,000 = 165,000$ sec., i.e. $\sim 46$ h. We notice also that with 1,500 nodes the solution of 100 million stars would be feasible.

V. GRID PORTING

The code has already been ported on Grid infrastructure: the application reads data from the catalog and write the solution to it. It is also restartable, then we can run and restart the execution from where it stopped the previous time. Currently we have the availability of the Cometa INAF grid infrastructure with 70 nodes, each one with two processors AMD Opteron 2218 dual core 3.2 GHz and 2GB per core, for a total of 280 cores, with a peak of 5.2 GFlop/s. The Grid middleware is based on gLite v. 3.2. We have performed some preliminary tests to calculate the expected performances of the Cometa INAF infrastructure doing a comparison with the single processor GFlop/s of BGP. Due to space limitation we cannot report here the detailed results, but we just want to emphasize that the analysis made show that we could run on Cometa for the first two years of mission with 10 million stars and with $\Delta$TAttitude not less than 7 seconds. The estimated execution time per iteration with only 3,000,000 stars will be of $\sim 63$ seconds and a complete run (3,000 iterations) will be performed in 189,000 seconds ($\sim 52.5$ hours). We can conclude that the grid infrastructure will be used for the first few cycles of mission and for solutions with very few stars (e.g., 1 million). Cometa will be used as an infrastructure to support and to determine alternative solutions, for example with different values of $\Delta$TAttitude ($\sim 5/60$). From May 2012, we will also have the availability of the INFN Pisa node with 1,000 nodes and OpenMP. This consists of 256 processors AMD Opteron 2356 quad core 2.3 GHz, for a total of 1,024 cores with 1 GB RAM per core. For larger solutions and at the end of the mission we will use BGQ system, for which we already have a preliminary agreement with CINECA.

VI. CONCLUSION AND FUTURE WORK

In this paper we have presented a hybrid solution which uses a modified PC-LSQR, a conjugate gradient-based algorithm, to develop the AVU-GSR module that aims to produce a Global Sphere Reconstruction, with the aid of some parallelization techniques and of an ad-hoc compression algorithm of the Total Matrix. During the mission, the code will be used for a range of 10 million stars up to a maximum of 30-50 millions. The estimated memory requirements are between 5 GB up to 2 TB of RAM. The parallel code, using MPI and OpenMP (where available), is characterized by an extremely low communication level between the processes, so that preliminary tests on computing speed show a behavior close to the theoretical one. In the initial phase we will use 256 nodes, instead during the mission operations access to 1,500 nodes of the IBM BGQ system at CINECA is anticipated. This project is supported by the CINECA (Italian Supercomputing center) and it has been recently adopted by the Italian NGI (IGI) for the test and running phase of the application. It will be used both for the developing of the middleware (gLite/EMI) for applications that can run on the two different infrastructures (HPC and Grid-MPI) and in the first period of the mission run. We are currently working to improve the application, by increasing the degree of parallelism and by tackling the problem of critical sections.

VII. ACKNOWLEDGEMENT

This work has been partially funded by ASI under contract to INAF I/058/10/0 (Gaia Mission - The Italian Participation to DPAC).

REFERENCES