Introduction

The 3D-CRS Stack is a time processing step based on a hyperbolic traveltime approximation. It is based on a fully data driven approach that requires a large computational effort to extract the eight kinematic wavefield attributes, that fully describe the CRS operator, from the seismic data. Although it is possible to simplify the task by reducing, e.g., the number of attributes, the Eni experience (Marchetti et al. [2002], Borrini et al.[2005], Cristini et al.[2003]) has clearly demonstrated that the computational effort for an accurate determination is well repaid by the quality of the final result.
Eni's previous implementation was based on an MPI approach with a master that had a double role: the I/O control on the filesystem and the process distribution among the slaves. This approach has fully satisfied the production expectation for several years even with the typical limitations related to MPI, a message passing library that, in our experience, still represents a good solution in case of a stable Linux cluster. These hypotheses, however, becomes weak in case of clusters with a very large number of CPUs (or cores) since the probability of a failure increases linearly with this number. On the other hand the size of the prestack volumes is always increasing, so that there is a real necessity of using *thousands* of CPUs, but with an unacceptable probability of crash of the MPI application. This situation has an important negative impact on the planning of the production activities.

To address these limitations, a new non-MPI, parallel computational model for the 3D-CRS Stack application has been implemented and denoted as *grid version*.

Another limitation of the MPI version of our code is the impossibility to modify dynamically the number of computing nodes. This means that while the application is in production we cannot aggregate the additional resources freed by other users. With the new Grid version these extra-resources can be easily used in two different ways. As a matter of fact, it is possible to reduce the total time or to run the application with a larger and then more expensive set of parameters obtaining, altogether a better result.

For a programmer team and for the users the approach to Grid computing presents some initial difficulties. The idea that there is no longer a "slave driver" that coordinates the job for everyone and checks everything is quite strange. In the Grid approach there are much more resources and everyone gives its best contribution. But the most interesting thing is that the "show" goes on even if someone is not very well.

The new Eni grid version allows the user to run 3D-CRS applications on some clusters, up to *thousands* of calculating cores without the risk of coming back on Monday morning discovering that he has to explain his boss that the application crashed five minutes after he leaved his office on Friday afternoon. Monday morning is already rather hard by itself.

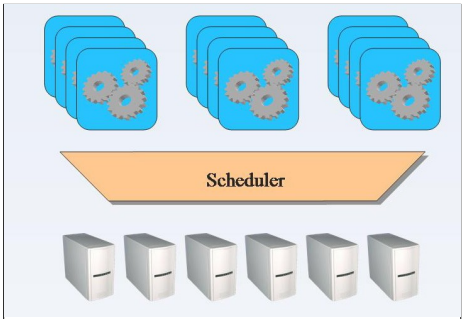
Analysis of the problem

Requirements

Since we had the need of reducing the total time of the 3D-CRS data stacking, we asked ourselves how to increase the number of CPUs preserving all the improvements introduced for several years on our 3D-CRS application. Some options were possible but several expectations were also to be fulfilled: a) the original MPI version was five years old and in this period many improvements were implemented, giving rise to a very good, stable code base. We had to retain all the acquired know-how thus avoiding to start from the scratch. b) the application had to be portable on clusters with different operating systems and since it was not so clear that we were going to have one large cluster, it should have been possible to process the same dataset on several smaller clusters. c) the candidate cluster did not have any special configuration needs. It is obvious that it had to be a good platform of which we could not make any assumption regarding the filesystem, the network, the number of core per node and the amount of RAM memory. d) It would have been better to have the possibility to change dynamically the number of CPUs. This would have allowed to maximize the computing resources adapting the load to the presence of other users.

The strategy of parallelization of the 3D-CRS problem is quite simple since it is possible to find *concurrently* the parameters of the operator for each single stacked trace. The amount of memory required for every process is less then a gigabyte and, as matter of fact, many processes share one datum.

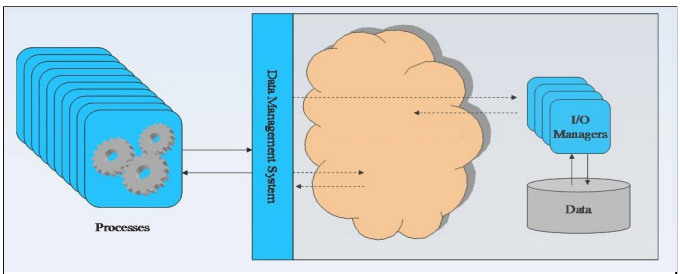
Using the MPI version the bottleneck was the data traffic to and from the master. Depending on the characteristics of the acquisition and on the seismic traces (sampling interval and trace length) the master reached saturation when controlling between the 600 and 800 slaves. A faster processor would have reduced in theory the total computing time, but without allowing the scaling of the application to a larger number of CPUs.



The job scheduler takes care of managing the execution of tens (or hundreds) thousands of jobs, taking into consideration the resources currently available and the policies

Solution strategy

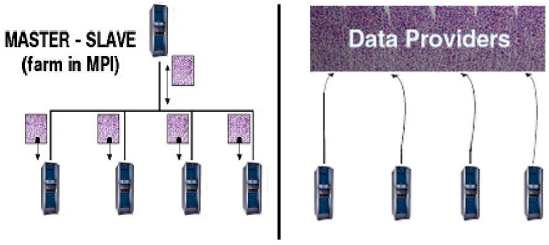
The new Eni code does not have a master. There are some demons working on dedicated nodes that manage the I/O. All the calculating jobs process a small part of the whole dataset corresponding to a small range of inlines and crosslines. These jobs are independently submitted, at the moment, to only one cluster. Every job will ask to the I/O nodes the required data to accomplish their task and then the result is sent back to the same nodes. The original computational FORTRAN routines have been almost completely reused.



The data management system is like a service, completely independent of calculate jobs. The interface between calculate process and data management system are the clean API.

In the approach with MPI, the master is in charge of assigning the workload to the slaves and

of dispatching the data. In the new code, thousands of independent jobs are coordinated by a scheduler and by the data-management infrastructure. There are different advantages in the new application: each job runs independently so that failures do not affect the entire elaboration; the data management is independent of the computation, thus allowing the implementation of different I/O strategies. Each process obtains its data with a *pull strategy*.



The MPI version uses a master-slave paradigm, where the master is in charge of pushing data to the slaves. In the Grid version of application, every process pulls the data it needs for the elaboration by issuing requests to the data management system.

Testing

One mandatory requirement was that the *grid version* of the 3D-CRS was to avoid a strongly coupling of the application to a particular job scheduler. The present code works properly also without any scheduler but via ssh protocol. However, this is not a friendly use of the application in a conventional industrial environment. The intensive testing, and, now, the production deployment are done on a Linux AMD cluster. Every node has two dual-core CPUs with 2 GB of RAM and the total number of node is 1250, corresponding to 5000 cores. The cluster is not dedicated to 3D-CRS. The communication network used by the application is the Gigabit Ethernet; the filesystem is a GPFS and the job scheduler Platform LSF.

In our application there are two different kinds of steps. The first one is the determination driven by the data of the CRS parameters; the second one consists in stacking the midpoint-offset data along the trajectories defined by the estimated parameters. Only the first kind is extremely computationally intensive. The second one is dominated by a combination of performances depending on the speed of the filesystem access and of the network. Altogether, the total amount of idle time spent is only a small part in the whole process.

The application testing was done on a large variety of datasets. It was necessary to find a good trade-off between the I/O and CPU time. This could be easily found empirically by submitting as many as possible small jobs pulling data of different sizes to estimate the proper compromise between the number of queued jobs and the processing time, bounded for a single job between 15 and 30 minutes.

The relevant quantities that can affect the functionality of our approach are, from the part of the data, the size of the prestack dataset, the number of traces per gather and the number of samples per trace. From the point of view of the code, there are parameters to be set from the users that strongly influence the processing time of a single job. Moreover, the time spent in some internal routines depends on the quality of the input data since they have exit conditions based on the stability of the solution. During the tests the size of the prestack volumes were in the range between 50 and 640 GB with a fold between 15 and 240 and the number of samples per trace between 400 and 2000. These values represent quite well the conventional seismic volumes.

Administration

The easier way to monitor and control a large number of jobs is the use of the scheduler tools. The possibility to set arrays and groups of jobs gives the user the

opportunity to administrate only two “entities”, the array of I/O daemons and the group or calculating jobs. Using the features of the queues it is possible to automatically expand the cluster occupation with the running application when other users free their CPUs; conversely, the scheduler releases these resources when they are requested by other applications, thus obtaining the maximum utilization of the cluster(s). For other purposes some utilities were developed.

Next steps.

All the tests were done on a large single cluster. In the next months, the available resources will change; from one cluster to several clusters that will have each 1000 or 2000 cores. We plan to move in the direction of the multi-cluster version of our application.

Conclusions

Our solution gives a result that fulfils the expectation of many operators. The seismic processing manager can have results in a shorter time, the system administrator is fully satisfied to see an intensive production on his computing resources, and the seismic analyst can run a large application trusting in the system reliability without worrying too much about the processing and has then the possibility to spend his time thinking to the geophysical meaning of his work. A quite important aspect is that this result was obtained working on a mature source code, already validated and very well known by the users, avoiding any the long initial training of a product with many small bugs.

Acknowledgments

The authors would like to thank Eni E&P Division for the permission to publish these results, the Eni E&P Scientific Calculation Dept. , Nicola Bienati (Eni) for the useful discussions, CINECA for the support during the testing phase and Ernesto Bonomi (CRS4) for the advises coming from his long term vision and the proofreading of this paper.

Bibliography

Marchetti P., Cristini A., Cardone G. , [2002], *3D Zero Offset-Common Reflection Surface Stack for Land Data - Real Data*, 64th EAGE Annual Conference & Exhibition Florence

Borrini D., Cristini A., Follino P., MArchetti P., Ojo C., Zamboni E. [2005], *3D CRS processing: a new approach to enhance S/N ratio of Western Africa low-fold data.*, 67th EAGE Conference & Exhibition, Expanded, Madrid, Spain.

Cristini A., Cardone G., Marchetti P., Zambonini R.,[2003], *3D ZO CRS Stack: issues related to complex structures and real data.* , 73rd Ann. Internat. Mtg., Soc. Expl. Geophys., Dallas, Texas

Bergler, S. [2004]. *On the determination and use of kinematic wavefield attributes for 3D seismic imaging*. Ph,D thesis , University of Karlsruhe, Germany

Müller, A. [2003], *The 3D Common-Reflection-Surface Stack - Theory and Application*, diploma thesis, Karlsruhe University.

Nabrzyski J. , Schopf, J. , Weglarz J., [2004] *Grid resource management: state of the art and future trends* , Kluwer Academic Publishers

LSF 7.0 User Guide