Integration of a Parallel Algorithm with a Cluster Grid for an Industrial Framework

Massimo Orazio Spata, Salvatore Rinaudo, Angelo Marotta, Francesco Moschella
STMicroelectronics
Stradale Primosole, 50
95121 Catania, Italy

Abstract: - Computer simulations are becoming a fundamental instrument in order to determine the suffer effects of a specific model by varying its input parameters. However, to obtain simulation results, the needed computational power required, by increasing the model parameters, in a reasonable time, is not so indifferent. So the aim of this research is to optimize computational throughput. At ST Microelectronics, a new integrated computational solution to resolve this problem in an efficient way has been implemented and tested. This solution consist of two blocks: a parallel implementation approach, named Exemplar, and a workload balancing system into a Cluster Grid. The first block was designed on a heterogeneous hardware platform which could have different computational performances. It is based on different optimization methods: CRS (Controlled Random Search) method, and Monte Carlo method. The second block consists of a cluster of CPU servers integrated through a workload balance system and a data resource management system (DRMS).

Key-Words: - Cluster Grid, Parallel Computing, Exemplar.

1 Introduction

In many fields of research, simulators have become a fundamental instrument in order to determine the effects on the output of a determined model by varying input parameters. Therefore, the reliability of a simulator depends on the accuracy of the model parameters.

For example, in the event of a simulation of a microelectronics device, the parameters demands in order to model its behavior are connected to physical phenomenon like mobility, recombination and so on. To optimize the characteristics of bipolar transistors, and particularly, when describing an electrical device through an equivalent circuit, the parameters are connected with the electrical members into the circuit.

In industrial applications, it would be opportune to have a global optimizer as Exemplar, integrated to a scheduler based on load balancing in an heterogeneous and distributed environment like a Data Resource Management System (DRMS), with different hardware platforms (32 bit and 64 bit, Intel or AMD) and different Operating Systems (Linux, Unix, Windows).

2 Cluster Grid optimization

To reducing hardware costs and optimizing it, the main target questions of this research were: new faster hardware or keep using what we have? Can we do more with less?

In other words the main target of this research is to optimize Computational Throughput of a Cluster Grid.

Research targets have been achieved and the resulting SW package, called Exemplar, has been constructed choosing the CRS and Monte Carlo methods [4, 5] suitably and modified by us in order to obtain better performance.

Prior art in relation with this paper is reported in [1..3]. In [2], a new algorithm for global optimization has been described. This was implemented in the optimization tool Exemplar and it could run paralleling on multi-vendor CPU and OS. The parallel process was managed by agents through a system of RPC daemons [1..3].

The resulting Exemplar software package allows the extraction of any type of parameters for all kinds of simulators, without the need to know a priori the exact analytical dependence of the output on the input model parameters, and also to automatically calibrate the generic simulator in the working-range of interest. In order to extract the correct parameters from an event the Least Squares Method has been used which minimizes the norm and the differences between the values of the output measured (or requested). Applying CRS and Monte Carlo methods, independence of Exemplar input optimization is guaranteed.
parameters [6], allows the distributed computation on a parallel architecture like a cluster Grid.

2.1 The Monte Carlo simulation method applied
Monte Carlo method is a statistical numerical simulation method that uses sequences of random numbers to perform the simulation. Statistical numerical simulation methods may be contrasted to conventional numerical discretization methods, which typically are applied to ordinary or partial differential equations that describe a complex system. In many applications of Monte Carlo, the system is simulated directly, and there is no need to even write down the differential equations that describe the behavior of the system: the only requirement is that the physical (or mathematical) system be described by probability density functions (pdf's). Assuming that the behavior of a system can be described by pdf's, the Monte Carlo simulation can proceed by random sampling from the pdf's. Therefore many simulations are then performed (multiple "trials" or "histories") and the desired result is taken as an average over the number of observations.

Monte Carlo simulation requires the ability to generate independent pseudo random draws: the linear congruent algorithm supplemented with a shuffle to mitigate correlations [8] was implemented to construct independently distributed random draws, the linear congruent algorithm produces uniformly distributed random draws. Given pseudorandom’ numbers resulting from the linear congruent algorithm, the construction of independent random draws with arbitrary probability distributions becomes a problem of transforming independent, uniformly distributed random draws into random draws distributed according to the desired probability distribution. The implemented Monte Carlo deals with normal distribution, obtaining it through the following transformation [8] of an uniform U(0,1) distribution, (U_1 and U_2 are independent random uniform numbers drawn from U(0,1)):

\[
\frac{1}{\sqrt{2\pi^2}} e^{-\frac{x^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}} \cdot 2\ln(U_1) \cos(2\pi U_2)
\]

Given a known covariance matrix, the correlation among parameters was implemented using the Karhunen-Loeve discrete expansion [8].

Monte Carlo computations have errors which have a probabilistic nature, and the error size is typically expressed through the well-known statistical quantity called the standard error: one typically assumes that the problem variance, \( \sigma^2 \), is constant and independent of the number of observations; then there is a square law relationship [9] between the standard error of a computation, \( \sigma(n) \), and the number of observations \( n \):

\[
\sigma(n) = \frac{\sigma}{\sqrt{n}} \quad (2)
\]

Therefore, to reduce the error \( \sigma(n) \) a large number of simulations are required: due to this fact, a Monte Carlo calculation tends to converge slowly and is often compute bound if it is performed serially on a single compute server. Monte Carlo algorithm is ideally suited to parallel computing since each simulation is independent from each other so each simulation ideally could be performed in a computation server: the parallel computing is often mandatory for Monte Carlo above methods when the system to simulate is complex and each simulation is time and cpu consuming task.

2.2 Experimental computing Environment
The CAD computing environment at ST is based on a layered architecture made up of three basic tiers. In this tiered computing environment, each layer corresponds to 3 basic categories of systems: desktop workstations or pc, login servers, and computing servers. This architecture is reported in fig 2.2.a:

![Fig. 2.2.a The ST architectural model of a Cluster Grid](image)

As showed in fig. 2.2.a the three tier components are:
- **Desktop workstation** is the system in front of the users: this system can be a Sun workstation or a Windows PC running an X11 server (such as Reflection X) and it
basically provides the graphical environment.

- **Login servers** are meant to provide a working environment to the users (Solaris, Linux, Enterprise Linux and so on). Users connect to a login server to access the correct operating environment to run their tools. These servers are also optimized to host a high number of users, but not for computations.

- **Computing nodes** are the servers on which computations take place and are globally referenced as the Compute Farm. It is a heterogeneous system and so in Compute Farm there are several kinds of compute servers of various architectures and operating systems.

The compute farm is not just a bunch of CPUs: these resources are organized and virtualized by a software layer with a Load Balancing System.

There are two fundamental goals in using a compute farm:

- **Computing resources are virtualized**: a user can request for a Linux host with RedHat and more than 4G of memory without having to know which farm node fulfills these requirements.
- **The overall workload is optimized**: when a user submits a job to a queue, the Load Balancing System processes are in charge of distributing the workload in a uniform way among the cluster nodes. This approach is often referred to as high throughput computing (HTC) in contradistinction to high performance computing (HPC). In HTC we want to maximize the number of jobs per hour (throughput), in HPC we want to minimize single job duration.

So, integration of Exemplar with a Load Balancing System permits to attain an HPC/HTC system, maximizing the number of jobs per hour and minimizing single job duration.

### 3 Grid-Exemplar

Research targets have been achieved and the resulting SW package, called **Grid-Exemplar** has been constructed choosing the CRS and Monte Carlo methods [4, 5] suitably and modified by us in order to obtain better performance.

The resulting software package allows the extraction of any type of parameters for all kinds of simulators, without the need to know a priori the exact analytical dependence of the output on the input model parameters, and also to automatically calibrate the generic simulator in the working-range of interest. In order to extract the correct parameters from an event the Least Squares Method has been used which minimizes the norm and the differences between the values of the output measured (or requested). Independence of Exemplar input parameters [6], applying CRS and Monte Carlo methods, allows the distributed computation on a parallel architecture like a cluster Grid.

In order to distribute the computations over a cluster Grid, the described methods were designed by using a RPC multi server network [7]. When it is requested to evaluate a function, the Exemplar algorithm (the client) asynchronously asks to a set of **Exemplar** submit servers (named node1, node2, ..., node_n) to run the simulation. The server requests are sent out with a high-level callrpc(). So, the client catches the replies returned by the servers’ nodes (figure 3.a) through a RPC service daemon:

The new Exemplar target, with the new architecture paradigm of parallel scheduling is reported in fig 3.a:

![Fig.3.a The new architectural model of parallel scheduler for a Cluster Grid](image)

As showing in fig. 3.a a user can login on a Login Server via X11 server (like VNC or Reflection X) and submit a job using data and applications stored on Server storage and shared via NFS file systems. On nodes there is a RPC system of agents that listening jobs request. They were designed to have them executed as independent threads and on distributed server.

#### 3.1 Grid-Exemplar architecture

In order to distribute the computations over a **Cluster Grid**, the described methods were designed by using a RPC multi server network [7]. The Exemplar algorithm asynchronously requests to a set of submit servers (named node1, node2, ..., node_n) to run the simulation. The server requests are sent out with a high-level callrpc(). So, the client catches the replies returned by the servers’ nodes (figure 3.1.a) through a RPC service daemon:
4 Experimental results

Experiment tests have demonstrated that the best total throughput is expected using Exemplar with a Load Balance scheduler. Exemplar tests were performed using Monte Carlo method based on 5000 runs on a Grid Cluster composed by 32 Compute Server Sun with Unix Solaris 2.8 OS and a total of 50 CPUs.

Each run is an electrical simulation of an operational amplifier that measures the gain-bandwidth product at 100 KHz, the phase margin, the positive and negative slew-rate, varying the compensation capacitor of the amplifier model and the load capacitor in agreement with Gaussian distributions; for each run its simulation time was measured. Results with and without the Load Balance scheduler, are reported in fig. 4.a, 4.b and 4.c:

5 Conclusion

Linking the Load Balance scheduler and Exemplar, we have to consider the overhead equal to about 10 seconds per job; by multiplying this time for 5000 simulations, 32 (or 50) per time, we obtain a heavier job dispatching time, and this problem is highlighted by jobs of an average duration of about 254 seconds. Moreover, considering that the Exemplar system without Load Balance runs only a job per server (and not per CPU) whereas the integration runs a job per CPU, in multiple CPU server systems, there is an ulterior loss of performance due to the Front Side Bus bottleneck. On the other side, its equal to the number of submitted nodes, considering a minimum loss of performance during the dispatching phase, the integration reveals a remarkable advantage due to the increase number of parallel jobs per time. So, for example without Load balance integration, on
the Grid cluster we could parallelize only 32 jobs
per time, whereas through Load Balance we have
overcome this limit and used all CPUs running even
per 50 jobs in parallel per time. Another remarkable
advantage consists in the possibility of scheduling
on the basis of the instantaneous Load Balance,
which, in a cluster shared among 1000 users and
130000 jobs per month, like the one at ST, allows to
the users an optimal calculation resources usage.
Finally, as shown in graphs 4.a and 4.b we have
obtained a minimum loss both in terms of average
duration per single job and of the whole duration of
5000 jobs (11.5 hours without Load Balance
systems, 12.0 hours with Load Balance systems).
Anyway, by integrating Exemplar with Load
Balance, we have obtained a greater balance, and a
greater exploitation of the distributed and shared
calculation resources. Furthermore, by increasing
the number of parallel compute CPUs to 50, we
have measured a higher average time per single job
due to the collocation of more jobs per compute
server and a total time lower of about 1.5 hours due
to the greater throughput of 50 jobs per time.

Acknowledgements

Acknowledgements to the EU for granting the
financial support within COMSON for this work.
Acknowledgements to Prof. A.M. Anile.

References:
Muscato “Controlled Random Search Parallel
Algorithm for Global Optimization with
Distributed Processes on Multivendor CPUs”–
“Progress in Industrial Mathematics at ECMI 98”
Implementation in a Industrial Framework of
Statistical Tolerancing Analysis in
Microelectronics” Proceedings of the 5th
International Euro-Par Conference on Parallel
Processing 1999.
[3] Angelo Marcello Anile, Salvatore Spinella,
Salvatore Rinaudo “Stochastic response surface
method and tolerance analysis in
microelectronics”, COMPEL: Int J for
Computation and Maths. in Electrical and
Electronic Eng., Volume 22, Number 2, 2003 ,
pp. 314-327(14).
[4] W.L. Price, Global Optimization by Controlled
Random Search Journal of Optimization Theory
and Application, vol. 40, N. 3, July 1983
a CAD Workstation Journal of Optimization
Theory and Application, vol. 55, N. 1, October
1987
[6] B. Flury, A First Course in Multivariate Statistic,
Springer, 1997
[8] Glenn Eric Johnson: "Constructions of
Particular Random Processes", Proceedings of