Load-balancing in Large Machines Applied to Weather Forecast: a Preliminary Study

Eduardo Rocha Rodrigues, Philippe O. A. Navaux
UFRGS - Federal University of the Rio Grande do Sul
Institute of Informatics
{errodrigues, navaux}@inf.ufrgs.br

Jairo Panetta
INPE - Brazilian Institute for Space Research
CPTEC - Center for Weather Forecast and Climatic Studies
panetta@cptec.inpe.br

Abstract

The increase in performance have become synonym of increase in the number of processors. Computers with thousands of processors are becoming common in many computing environments. This fact imposes plenty of challenges for the development, debugging and management of the programs executed on these computers. One of these challenges is load balancing. Several works have been done on load-balancing, but few of them deal with large machines and highly dynamic applications like weather forecasts. This article presents a preliminary study of load imbalances in weather forecast models, that can prevent this kind of application from taking full advantage of large machines. The model chosen for the tests was the BRAMS model, an open source mesoscale model commonly used in Brazil. This work focuses on the source of load imbalance related to the nesting grid technique.

1. Introduction

The increasing performance of the digital computers has made possible the scientific and technical development of several areas of the knowledge and human activity. The more powerful the computers are, more information they can process and consequently finer models can be implemented on them.

Teraflop computers have become commonplace in an increasing number of computing environments. One example of this class of computers is the BlueGene/L in the Lawrence Livermore National Laboratory (LNLL), that has 131.072 processors. In the article [22] it is presented a molecular dynamic simulation program that reached a performance of 103 Teraflops over a 7 hour run on this computer. Another example is the Earth Simulator [9], that consists of 640 nodes with 8 vector processors each. This system reached 26.58 Teraflops of peak performance executing a global atmospheric circulation model [21].

These new high performance computers are employing increasingly more processors, as it can be seen from the TOP500 list. This imposes multiple challenges for the development, debugging and management of the programs executed on these computers. The task of keeping every processors busy with useful work is one of these challenges, known as load-balancing. Several works have been done on load-balancing [2] [7] [14], but few of them deal with large machines and highly dynamic applications such as weather forecasts.

This article presents a preliminary study of load imbalances in weather forecast models that prevents this kind of application from taking full advantage of parallel machines. The model chosen for the tests was the BRAMS model that is an open source mesoscale model commonly used in Brazil. Few processors was employed to confirm the existence of one source of load imbalance in PC cluster, but it is underway tests with more processors.

The remainder of the article is organized as follows. The Section 2 presents the load-balancing problem. Section 3 shows related works. Some general sources of load imbalance found in meteorological model are presented in the Section 4. The Section 5 focuses on the load imbalance due to the technique of nesting grids, that represents a trade-off between computational speed and high-resolution. The last section concludes this article with some remarks.

2. Load-balancing

The main objective of the increase in the number of processors used to perform a computation is the reduction of the execution time. Two metrics are commonly used in this context: speedup (S) and efficiency (E). They are defined as follows:

$$S = \frac{T_s}{T_p} \qquad E = \frac{S}{p}$$

where T_s is the sequential execution time and T_p is the parallel execution time on p processors. Ideally, the efficiency is equal to 1, but usually it tends to saturate as the number of processors increase [8]. There are mainly three reasons for this fact [6].

The first reason is that some parts of the computation cannot be parallelized, i.e. these parts do not take advantage of multiple processors. Thus, the minimum time needed to finish the computation is, at best, the time needed to execute these sequential parts. This fact is known as Amdahl Law [1]. The second reason for the decrease in efficiency is related to the proportion between communication and computation. If the communication grows faster than the computation with the increase of the number of processors, then it is expected a limit from which the increase in the number of processors leads to a speedup reduction.

Load imbalance is the third reason for the parallel efficiency decrease. Each processor in a parallel machine should receive an equal workload in order to achieve the best performance. If some of the processors receive more work to do than others, then the less loaded processors should wait for the more loaded ones, thus the efficiency drops. Load balancing techniques are used to avoid this unequal distribution of work. These techniques can be categorized as static or dynamic. In the first one, the load distribution occurs before starting the computation and does not change during the execution. The static load balancing is also known as scheduling problem [23] and it is suitable for problems whose behavior is known *a priori*.

The dynamic load balancing technique makes few assumptions about the application behavior. The informations about the load are obtained at runtime in order to drive the load balancing mechanism. This approach may incur in a higher overhead due to the spent time to collect data and to distribute or redistribute the load. However, this approach is more suitable for inherently dynamic application.

The book [23] classifies dynamic load balancing as centralized or distributed. In the centralized approach, only one processor takes the decisions about the load balancing, whereas all processors participate on this process in the fully-distributed one. Hybrid approaches are also possible as presented in [25]. The article [20] points that the centralized approach incurs a lower overhead during the load estimation, but this implies in slower responses on machines with many processors. The distributed approach, in its turn, has a better performance in machines with many processors, however it implies in a bigger overhead for load estimation.

The dynamic load balancing can be implemented directly in the application code. This alternative causes a lesser overhead in the execution, because it works directly on the data structure of the application. However, the direct implementation requires a deep knowledge of the application code, therefore it is difficult to use it on legacy systems. Moreover, as it is said in [5], the resulting implementation cannot be used in other applications since it is highly tied to the original one.

An alternative to the direct implementation are the load balancing toolkits. Typically, these toolkits are implemented as libraries that are linked to the application and offers services to it. This approach is more convenient for legacy systems, since its use does not require or requires less change in the application. However, the toolkits usually cause more overhead.

3. Related work

Several proposals and implementations of load balancing strategies can be found in the literature, that shows a great interest in the subject. The winner article of the Gordon Bell prize of 2005 [22] is an example of dynamic load balancing directly implemented in the application. This article presents molecular dynamics simulation, that is a computational technique for simulations of particles as atoms and molecules.

The general structure of the molecular dynamics (MD) simulations is presented in [18]. Usually this structure is simple, therefore the implementation of the load balancing on MD codes also tends to be simple. On the other hand, other applications have too many forms and they are too complex that load balancing becomes a complex task. Weather and climate forecasts are examples of this kind of application.

MM90 is an example of meteorological model that implements dynamic load balancing. It is a parallel implementation of the Penn State/NCAR Mesoscale Model (MM5) in Fortran 90. The article [14] describes this model, the dynamic load balancing method and the performance results. However, the article does not describe the used criteria to drive the load balancing, that is essential to determine the scalability, as [20] points out.

The article [7] presents an algorithm to load balancing the subgrid orography scheme applied to the CAM3 model, that is a global atmospheric circulation code designed to execute on several platforms. The subgrid orography scheme treat the influence of the orography, i.e. the average height of land, on the forecast accuracy. As the orography does not

change, a static load balancing strategy is used. The algorithm tries to minimize the communication, while it finds the best load distribution. However, these results are specific to the used scheme.

Other articles on load balancing are [19] [24] [15] [17] and [3].

3.1. Load balancing toolkits

All works presented above are examples of direct implementation. This section presents some load balancing approaches based on toolkits.

Zoltan [4] is an open source library developed by Sandia National Laboratories that provides load balancing services. This library is not restricted to a specific application nor imposes restrictions in the data structures of the application. However, the use of the Zoltan must be taken in account during the development of the application. Therefore, in legacy systems, it would be needed to change the original code in order to use this toolkit.

Mosix is a management system for clusters of PCs, that implements load balancing through process migration [2]. In this system, the processes migrate from slower to faster nodes, or lightly loaded nodes, in order to distribute workload. Even after it migrates, the process maintains its system context in the node of origin, to solve system calls. However, this fact causes performance degradation in applications with heavy interprocess communications, as [12] points out.

Developed in the University of Illinois, the Charm++ is an object-oriented language based on C++, and a parallel execution environment [10]. A program written in CHARM++ consists of a collection of distributed objects, whose methods can be called asynchronously. The execution environment controls the attribution and re-attribution of objects to the available processors and mediates the communication between the objects. This approach allows optimizations such as load balancing and automatic overlapping of computation and communication.

4. Sources of load imbalance in weather forecast models

Numerical weather forecast is an important application. Since the beginning of the development of electronic computing machines, with John von Neumann using the Eniac to predict weather [16], until now, with the increasing interest in climate change, this kind of application has demanded improvements in the computer science. Although several works had been done in load-balancing weather models, the recent increase in the number of processors or cores per machine to hundreds or thousands imposes a big challenge. In this section, it is presented some sources of load imbalancing that can prevent the models to use the full power of the recent machines.

Typically, numerical weather forecasts represent the state of the atmosphere as a set of floating point values, representing physical quantities (e.g. temperature, wind components, etc), over a discretized domain. Then, it is applied atmospheric motion governing equations to the initial atmospheric state, producing the state of the atmosphere at the next time-step. The iterative application of this procedure allows forecasting for the desired time period. Usually, the domain is discretized as a regular grid and the numerical computations is uniformly applied on it. However, localized atmospheric processes can lead to load imbalances. Examples of such processes are microphysics and dynamically localized emission sources on environmental models.

Microphysics mean physical processes active on the scale of individual clouds. As these processes are usually triggered by some threshold values in the grid representing the atmosphere, the computational load due to these processes can dynamically change through the domain. Thus, if the domain decomposition employed is a spacial decomposition, as typically it is, some processors would execute the microphysics while others would not. The article [11] presents a study about load imbalance of the Gesima model caused by some criteria that triggers the microphysics of this model.

Load imbalance caused by microphysics was also found in the Rams model, as pointed out by [13]. More than 20% of load imbalance was attributed to the presence of clouds in sub-domains of the overloaded processors.

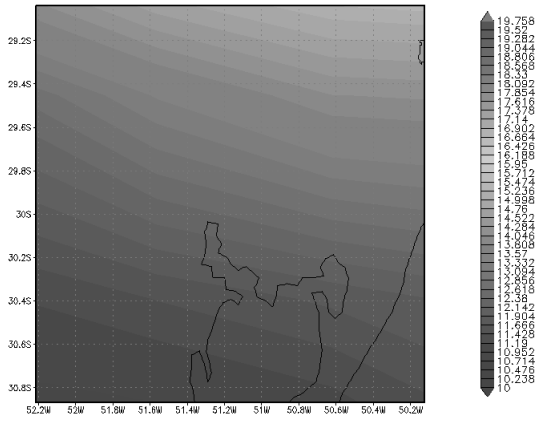
Dynamically localized emission sources on environmental models can also cause load imbalances. One example of such a model is the CATT-BRAMS, that is a system to monitoring atmospheric transport of biomass burning and anthropogenic emissions (emissions derived from human activities).

Another source of load imbalance is related to the way that the domain is discretized. Some techniques have been developed in order to improve resolution in limited areas. One of these techniques is known as nesting grids, where higher resolution grids are embedded in, and can exchange information with a lower resolution grid. The next Section discuss more about this source of load imbalance through the analysis of a specific meteorological model, the BRAMS.

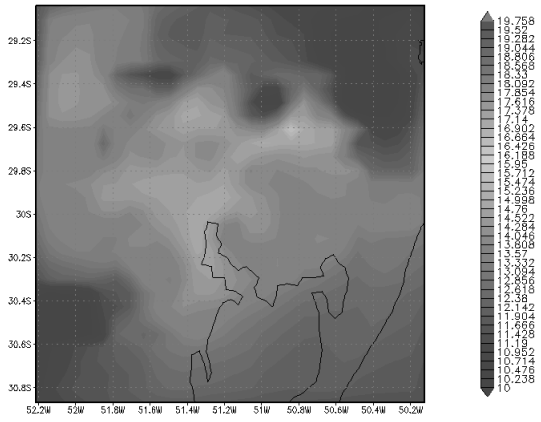
5. Experiments

This Section presents some experiments that show the nested grids in the BRAMS meteorological model can lead to load imbalance in cluster of PCs. BRAMS means Brazil-

ian Regional Meteorological System, it is a model derived from Rams and it includes tropical parameterizations. BRAMS receives as input a lower resolution data from another model (Figure 1(a)) and produces a higher resolution output for a specific region (Figure 1(b)).



(a) lower resolution forecast from a global model



(b) 7-km resolution forecast from BRAMS

Figure 1. Temperature forecast over the area of Porto Alegre.

When executing with more than one grid, the BRAMS begins advancing the coarser grid as if it did not contain nested grids. After that time-step, the coarser grid sends the boundary condition to the nested grid. The sent data is interpolated to the boundary points in the nested grid. The interior points of the nested grid are integrated under the influence of the boundary condition. Usually the time-step of the nested grid is a fraction of the coarser one to ensure numeric stability, thus it is needed to integrate the nested grid until its time reaches the same time that the coarser grid. After that, the whole nested grid is used to update the coarser grid where the finer one exists.

Both coarser and nested grid are decomposed among the processors in an equitable manner. However, the interpolation and the coarser grid update are done only on the processors with the sub-domain of the coarser grid where the nested grid exists. This is done to guarantee binary reproducibility: code results are processor count invariant, but this cause load imbalance.

A previous performance test in a SX-6 supercomputer had confirmed the existence of this source of load imbalance. In the present work, it was performed a similar test to find if the same behavior occurs in a PC cluster. The performance test consists of a 24-hour simulation over the area showed in the Figure 2. The output data are two grids with 32x32 points each one on the horizontal and 32 vertical levels. This test was executed on one and six slave processors and one master processor [1]. The number of used processors were the same as in the previous test on the SX-6 supercomputer.

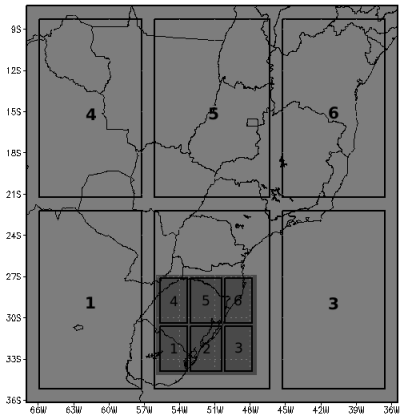


Figure 2. Domain decomposition.

During the test, it was collected the execution times of the most important sections of the simulation time-step. They are:

- Init: Initialization, that occurs before the time-step loop;
- outerBeforeInner: Data receiving from the master;
- TS: Calling of the main time-step routine;

1 Pentium III 1133MHz connected by a Gigabit Ethernet network.

Section	Sequential	Parallel					
		processor 1	processor 2	processor 3	processor 4	processor 5	processor 6
Init	5.0744	5.2841	5.2839	5.2840	5.2841	5.2839	5.2822
outerBeforeInner	20.7467	93.6903	22.6495	79.5858	66.6019	52.4408	38.8769
TS	1,619.0143	353.5216	328.7484	346.9930	340.8613	335.0797	326.6488
CoarseToFine	10.8653	9.2350	34.3023	15.9393	22.1928	28.1787	36.6182
FineToCoarse	33.0197	21.4158	93.0999	35.5287	48.5392	62.5785	76.3586
CLF	2.5818	0.5469	0.4221	0.5302	0.5306	0.5954	0.5315
DomainsToMaster	6.1528	7.8200	7.0224	7.6625	7.5140	7.3544	7.2041
TOTAL	1,697.5511	491.5949	491.5960	491.5963	491.5978	491.5844	491.5908

Table 1. Execution time in seconds.

- CoarseToFine: Sending of the boundary condition to the nested grid;
- FineToCoarse: Coarser grid update;
- CLF: Computation of the CFL-criterium;
- DomainsToMaster: Sending of data to the master.

The Figure 2 shows the domain decomposition and the Table 1 shows the performance results. The sections corresponding to the nesting grid processing (*CoarseToFine* and *FineToCoarse*) on the processor 2 exhibits the biggest time, as it is supposed to be since it is the only processor in the charge of the finer grid. However it can also be seen that the *outerBeforeInner* section of the processor 2 is lower than the other processors. That is because the other processors have to wait for the master processor, that, in its turn, waits for the overloaded processor 2. Another important fact is that the spent time in the sections *CoarseToFine* and *FineToCoarse* of the lightly loaded processors increases through the processor 1 to 6. The reason is that those processors have to send their portions of the finer grid to the processor 2, which receives the data sequentially.

These results show that the processor 2 is overloaded and this cause the other processors to become idle. If it is only considered the *TS* section, that is the main time-step routine, the efficiency in this test is 76%, however the load imbalance cause the total efficiency to drop to 57%. This test was also done with 12 and 18 slave processors. It was observed that the total efficiency drops faster than *TS* efficiency as the number of processors increase.

In order to perform this test with more processors, it is underway a study of the Charm/BigSim environment, that can be used to simulate large machines and predict the performance on those machines.

6. Final remarks

Load balancing is not a new theme, however the large number of recent articles shows that this theme remains of central importance. That is mainly because load balancing is very sensitive to the architecture and the application. A specific solution hardly obtain a good efficiency in different machines and different applications. Besides this, the use of the available solutions are very limited in production environments, that poses questions on the suitability of these solutions to deal with real problems.

This work is a preliminary study on load imbalances in meteorological models, that is undoubtedly an important application. Here it was presented some sources of load imbalance in meteorological models and analyzed the imbalance caused by nested grids. It was found that this load imbalance source limits the efficiency of the BRAMS model as the number of processors increase.

Next steps are underway and include to adapt BRAMS to execute it on Charm/BigSim environment in order to estimate its performance on large machines and to study the behavior of the nesting grid technique on these systems. On this environment it will be also possible to study the other sources of load imbalances such as microphysics and plumerise. The final goal of these studies is the development of a model to deal with load imbalances on large machines applied to meteorological models.

References

[1] G. Amdahl. The validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Conf. Proc.*, volume 30, pages 483–485, 1967.

[2] A. Barak, O. La'adan, and A. Shiloh. Scalable cluster computing with mosix for linux, 1999.

[3] J. Behrens. Adaptive atmospheric modeling: Scientific computing at its best. *Computing in Science and Engg.*, 7(4):76–83, 2005.

[4] K. Devine, E. Boman, R. Heapby, B. Hendrickson, and C. Vaughan. Zoltan data management service for parallel dynamic applications. *Computing in Science and Engineering*, 04(2):90–97, 2002.

[5] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio. New challenges in dynamic load balancing. *Appl. Numer. Math.*, 52(2–3):133–152, 2005.

[6] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, editors. *Sourcebook of parallel computing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[7] S. Ghan and T. Shippert. Load balancing and scalability of a subgrid orography scheme in a global climate model. *Int. J. High Perform. Comput. Appl.*, 19(3):237–245, 2005.

[8] A. Y. Grama, A. Gupta, and V. Kumar. Isoefficiency: Measuring the scalability of parallel algorithms and architectures. *IEEE Parallel and Distributed Technology*, 01(3):12–21, 1993.

[9] S. Habata, M. Yokokawa, and S. Kitawaki. The earth simulator system. 2003.

[10] L. Kalé and S. Krishnan. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In A. Paepcke, editor, *Proceedings of OOPSLA'93*, pages 91–108. ACM Press, September 1993.

[11] C. Koziar. Load imbalance aspects in atmosphere simulations. In *ICPPW '01: Proceedings of the 2001 International Conference on Parallel Processing Workshops*, page 134, Washington, DC, USA, 2001. IEEE Computer Society.

[12] R. Lottiaux, P. Gallard, G. Vallee, C. Morin, and B. Boissinot. Openmosix, openssi and kerrighed: a comparative study. In *CCGRID '05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 1016–1023, Washington, DC, USA, 2005. IEEE Computer Society.

[13] C. L. Mendes and J. Panetta. Selecting directions for parallel rams performance optimization. In *Selecting Directions for Parallel RAMS Performance Optimization*, 1999.

[14] J. Michalakes. MM90: A scalable parallel implementation of the Penn State/NCAR Mesoscale Model (MM5). *Parallel Computing*, 23(14):2173–2186, 1997.

[15] S. P. Muszala and J. J. Hack. The promise of load-balancing the parameterization of moist convection using a model data load index. *Journal of Atmospheric and Oceanic Technology*, 23(4):525–537, 2006.

[16] G. W. Platzman. The eniac computations of 1950—gateway to numerical weather prediction. *Bulletin of the American Meteorological Society*, pages 302–312, 1979.

[17] B. Radi and J.-F. Estrade. Adaptive parallelization techniques in global weather models. *Parallel Comput.*, 24(9):1167–1175, 1998.

[18] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, second edition, 2004.

[19] R. Reilein and G. Rünger. Combining thread programming with message passing for atmosphere simulation. In *PDPTA*, 2000.

[20] H. Rotithor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. In *Computers and Digital Techniques*, volume 141, pages 1–10, 1994.

[21] S. Shingu, H. Takahara, H. Fuchigami, M. Yamada, Y. Tsuda, W. Ohfuchi, Y. Sasaki, K. Kobayashi, T. Hagiwara, S. ichi Habata, M. Yokokawa, H. Itoh, and K. Otsuka. A 26.58 tflops global atmospheric simulation with the spectral transform method on the earth simulator. *sc*, 00:52, 2002.

[22] F. H. Streitz, J. N. Glosli, M. V. Patel, B. Chan, R. K. Yates, B. R. de Supinski, J. Sexton, and J. A. Gunnels. 100+ tflop solidification simulations on bluegene/l. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005.

[23] B. Wilkinson and M. Allen. *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[24] P. H. Worley and J. B. Drake. Performance portability in the physical parameterizations of the community atmospheric model. *Int. J. High Perform. Comput. Appl.*, 19(3):187–201, 2005.

[25] G. Zheng. *Achieving High Performance on Extremely Large Parallel Machines: Performance Prediction and Load Balancing*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.

Space weather is a discipline that lives between academia and industry, given the relevant physical effects on satellites and power grids in a variety of applications, and the field therefore stands to benefit from the advances made in industrial applications.Â The clearest opportunity lies in creating space weather forecasting models that can respond in real time and that are built on both physics predictions and on observed data. 1 Artificial Intelligence: Is This Time for Real?Â Even though many such machines exist today, especially in large HPC labs worldwide, we would think that the typical HPC user has not been persuaded to fully embrace GPU computing (at least in space physics), possibly because of the steep learning curve required to proficiently write GPU codes. Among the many reasons for load imbalance in weather forecasting models, the dynamic imbalance caused by localized variations on the state of the atmosphere is the hardest one to handle. As an example, active thunderstorms may substantially increase load at a certain time step with respect to previous time steps in an unpredictable manner - after all, tracking storms is one of the reasons for running a weather forecasting model.Â hard to implement load balancing in legacy codes. ConseÂ quality on extremely large machines or tend to take much. longer time to yield good solutions in a rapidly changing. environment. With the recent growth in the size of parÂ Our study case is a forecast of a moving thunderstorm. in the Southeast region of Brazil. We conï¬gured BRAMS. This hierarchical load balancing strategy reduces the load balancing overhead in large distributed computing systems with communication-optimized hierarchy. In the new strategy, the computation rate of node and time-varying characteristics of communication delay are considered, and a delay prediction model based on generalized neural network (GNN) theory is constructed.Â The GNN constructed by neurons can be applied to predict communication delays of large distributed computing systems having high practicability. The intelligent neuron has information storage ability and adjusts its transfer function in a set of functions by some training algorithms.