Thesis overview:

Parallel processing. Dynamic Load Balance in Sorting Algorithms

Marcelo R. Naiouf
Universidad Nacional de La Plata, Facultad de Ciencias Exactas
September 2004

The growing importance and interest in parallel processing within Computer Sciences are undeniable, and thus make it one of the more influencing areas of this discipline. Several reasons support this fact, such as the increase in the computer power, the existence of problems in which the sequential solution times are unacceptable, the possibility of mapping the problem's implicit concurrency into parallel processes in order to minimize response times, etc.

A *parallel system* is the combination of a parallel algorithm and the machine over which this is run, and both factors count with several variables. As regards parallel algorithms, they can be specified using a wide range of models and paradigms. As regards supporting architectures, even though they all count with more than one processor, they can be different in several dimensions, such as in the control mechanism, the address space organization, the processors granularity, and the interconnection network.

Among the parallelism objectives we can find those of reducing the running time and making an efficient use of computing resources. The unbalanced use of the processing elements may cause a poor efficiency or a parallel execution time greater than the sequential. *Load balancing* consists in - given a set of tasks encompassing a complex algorithm and a computers set over which they can be run - finding the mapping of tasks into computers which entails an approximately balanced work for each computer. A mapping which balances the processors' load enhances the global efficiency and reduces the running times.

The mapping problem is *NP*-complete for a general system with *n* processors, and thus the task of finding a minimum cost assignation cannot be dealt with computationally, except for really small systems. For this reason, alternative approaches can be used, such as relaxation, development of solutions for particular cases, enumerative optimization or approximate optimization (use of heuristics, which provides sub-optimal, though acceptable, solutions).

In some cases, the computing time associated to a task can be determined *a priori*. In such a circumstance, mapping can be used before the computation (*static* load balancing). For an important and increasing class of applications, the workload for a particular task can change during a computation and cannot be estimated beforehand; in these cases, the mapping should change during the computation (*dynamic* load balancing), performing balancing stages during the application execution.

Generally, the static load balancing has lesser complexity than the dynamic, but it is also less versatile and scalable. Conceptually, dynamic methods require some way of keeping a global view of the system and some negotiation mechanism for process and/or data migration. Even though they can potentially improve the application's global performance by redistributing the load among the processing elements, this activity is carried out at the expense of useful computation, produces a communication overhead, and requires extra space in order to keep information. An effective and efficient method cannot be established for all the cases. The selection always depends on the application and the supporting platform and, in several cases, it is necessary to adapt or combine existing methods in order to achieve a good performance.

Sorting is one of the most common operations carried out in a computer. The sorting task is defined as the ordering of a disordered set of elements in increasing (or decreasing) order. Several applications require the data to be ordered so as to be more efficiently accessed. Sorting is particularly important within parallel computations due to its close relation to data routing among processors (essential part of some algorithms). Many *routing* problems can be solved by ordering packages in their target addresses, while several sorting algorithms are based on routing schemes for their efficient implementation. In addition, the ordering operation is frequently used in Databases' processing, for instance, in operations with clauses *Distinct*, *Order By*, and *Group By* in SQL. Sorting algorithms can be categorized as based on comparison and not based on comparison. The former orders by repeatedly comparing pairs of elements and swapping them when necessary. For n items, the sequential

sorting based on comparison has a lower bound in time of $\Theta(n \log n)$. In particular, *mergesort* is $O(n \log n)$, which does not allow substantial improvements in the sequential algorithms. The methods which are not based on comparisons use certain known properties of the elements, and the inferior level is $\Theta(n)$.

There exist several ordering algorithms both sequential and parallel. The parallel sorting includes both the distributed versions of classical, sequential algorithm and the directly parallel methods. The process of parallelizing a sequential sorting algorithm entails the distribution of the elements in the available processors, which implies dealing with topics such as where input and output sequences are stored, or how comparisons are made. The sorting problems with large data volumes per processor are the most interesting and over which the current parallel machines work better, due to their computing power and memory capacities.

Many of the load balancing methods refer to problems which count with some way of knowing which the load is, for example, expressing it as a relation of the quantity of input points of a grid to be processed. It is interesting to treat the load balance problem in applications where the work does not depend on the input data sizes but on certain features of them. Many of the sorting algorithms can be found within this type.

Some sorting techniques try to balance the load by means of an initial sampling of the data to be ordered and their distribution according to *pivots*. Others redistribute partially ordered lists so that each processor stores an approximately equal number of keys and all take part of the merge process during the running. This Thesis presents a new method which dynamically balances the load based on a different approach, trying to carry out a work distribution making use of an *estimator* that allows *predicting* the pending workload.

The propose method is a variant of the Parallel *Sorting by Merging*, i.e., a technique based on comparison. Block orderings are carried out by the Bubble method or *Bubble Sort* with sentinel. In this case, the work to be done (in terms or comparisons and swappings) is affected by the *disordering degree* of data. The evolution of the work quantity in each algorithm's iteration has been studied for different types of input sequences (n data with values of 1 to n without repetition, random data with normal distribution), noting that the work decreases in each iteration. This was used in order to obtain an estimate of the remaining work waited for from a given iteration, and to correct the load distribution.

With this idea in mind, the method does not initially distribute all the data to be ordered among the tasks; instead, a percentage of this data is "reserved". After a certain quantity of "rounds" (in particular, with the 5% of the iterations), it estimates the remaining work of each task basing on what has already been done, and dynamically distributes the reserved data inversely proportional to the estimated remaining work for each task.

The presented scheme uses the master/slave paradigm, and it was implemented in a parallel architecture with bus communication (homogeneous PCs clusters), even though it can be also run in shared memory machines. The main characteristics of this method are its simplicity, efficiency, limited communication, and application possibilities in different problems. The results showing the goodness of the estimations and the possibility of suitably balancing the load in a high percentage of the cases are presented.

Marcelo R. Naiouf
mnaiouf@lidi.info.unlp.edu.ar