Efficient Dynamic Load Balancing Strategies for Parallel Active Set Optimization Methods*

I. Pardines[1] and Francisco F. Rivera[2]

[1] Dept. Arquitectura de Computadores y Automática, Univ. Complutense, 28040 Madrid, Spain
inmapl@dacya.ucm.es
[2] Dept. Electrónica y Computación, Univ. Santiago de Compostela, 15782 Santiago de Compostela, Spain
fran@dec.usc.es

Abstract. In this paper three strategies are described to restore dynamically the load balancing in parallel active set optimization algorithms. The efficiency of our proposals is shown by comparison with other heuristics described in related works, such as the classical Bestfit and Worstfit methods. The computational cost due to the load unbalancing in the parallel code and the communication overheads associated with the most efficient load balancing strategy are analyzed and compared in order to establish whether the distribution is convenient or not. Experimental results on a distributed memory system, the Fujitsu AP3000, highlight the accuracy of our estimations.

1 Introduction

Active set methods are based on a prediction (called working set) of the active constraints at the solution. The working set may change at each iteration, adding or deleting one constraint, which means adding or deleting a column of the Hessian matrix [3]. The addition is performed according to a cyclic distribution of columns among the processors. Therefore in this operation the load balance is maintained. However, deleting columns are a source of imbalance when the problem variables and its computations are distributed in a parallel implementation.

Our parallel algorithm works with a triangular matrix R (being $R^T R$ the Hessian approximation of the system) that is divided into triangular and rectangular blocks [6]. The computations of a rectangular block are executed in parallel whereas the computations of the triangular blocks are sequential. After the execution of each rectangular block a synchronization between all the processors of the system is needed. Initially, the system is completely balanced, but after a number of iterations it is possible that columns, belonging to the same rectangular block, are deleted. When the load unbalancing decreases the efficiency of the parallel code, an effective strategy to redistribute the columns between the processors is essential.

* This work was supported by CICYT under grants TIC 2002/750 and TIC 2001-3694-C02.

H. Kosch, L. Böszörményi, H. Hellwagner (Eds.): Euro-Par 2003, LNCS 2790, pp. 206–211, 2003.

2 Dynamic Load Balancing Strategies

The load redistribution problem described in this paper is a *NP*-hard problem similar to the Multiple Knapsack problems [5]. A set of processors, called donors (D), have extra loads in terms of the number of columns. The value of the unbalancing load w_i of each donor i is stored in an array called weight array (W). In the system there are other processors, called receivers (R), with under the average load, so they have the capacity to receive load. The value of this lack of load c_j per receiver j is saved in an array called capacity array (C). W and C are organized in a decreasing order.

Similar redistribution problems appeared in the literature. These problems try to minimize the total number of messages [4]. However, the objective of our methods is to balance the system minimizing the maximum number of receptions and the maximum number of sent messages per processor. So, we have to deal with a combinatorial optimization problem that can be formalized as follows:

$$\begin{array}{ll} \text{minimize} & \max\{\max \sum_{i\in D} x_{ij} \quad \forall j \in R,\ \max \sum_{j\in R} x_{ij}\ \forall i \in D\} \\ \text{subject to} & \sum_{i\in D} w_i(j)x_{ij} = c_j,\ j \in R; \\ & \sum_{j\in R} w_i(j) = w_i, \quad \forall i \in D; \\ & x_{ij} \in \{0,1\}, \qquad\quad i \in D,\ j \in R, \end{array} \tag{1}$$

where $\sum_{i\in D} x_{ij}$ is the number of messages received by processor j, $\sum_{j\in R} x_{ij}$ is the number of messages sent by processor i, and $w_i(j)$ is the load that processor i sends to processor j. Finally, x_{ij} is 1 if there is a message between donor i and receiver j, and 0 otherwise.

Three strategies are proposed to solve this optimization problem: the Load Donor Strategy (LDS), the Load Receiver Strategy (LRS) and the Load Hybrid Strategy (LHS), which are introduced in the next sections.

2.1 Load Donor Strategy

Two different stages are distinguished in this greedy method. Firstly, equal weights and capacities are searched in W and C, respectively. The perfect situation is based on finding a weight w_i identical to a capacity c_j. So, just one message between the corresponding donor and receiver processors is necessary.

In the second stage, the weight and capacity arrays are swept making matches between processors trying to balance the remaining computational load. In the process a suitable capacity is searched for each element of the weight array. There are two situations:

1. The value of the first element of W is greater than all the elements of C ($w_1 > c_j, \forall j \in R$). In this case, the processor with load w_1 sends $c_j = \max_{k\in R}\{c_k\}$ columns to the processor with the greatest capacity. The remaining load, $w_1 - c_j$ columns, has to be stored again in the proper position of W. The objective is to minimize the maximum number of sends per processor. If a donor redistributes as many columns as possible in a single message, the number of messages needed to achieve the load balancing tends to be minimum.

2. Element w_1 is smaller than c_1. In this case, a capacity c_j that equals w_1 is searched. If this capacity exists, a message is established. On the other hand, if there is not such capacity, the selection will be made among the processors with capacity c_j that verify,

$$c_j - w_1 > \min_{k \in D}\{w_k\} \ . \tag{2}$$

In this way, very small capacities that imply short messages are avoided. So, the possibilities that a donor has to divide its load into a large number of messages is reduced, thereby minimizing the maximum number of sends. The receiver, verifying equation 2, will be that which has already received the minimum number of messages. The aim is to avoid an increase in the maximum number of receptions per processor. In case equation 2 is not satisfied, the receiver processor will be the one with the greatest capacity. This process is repeated until there are no entries to match in W and C.

2.2 Load Receiver Strategy

This method is basically the same as the LDS strategy, but the priority is to minimize the maximum number of receptions per processor. Therefore, the technique is the same as the one described in section 2.1, swapping the roles of the weight and capacity arrays.

2.3 Load Hybrid Strategy

The idea behind this proposal is to combine the previous strategies. The objective is to minimize the maximum number of both sends and receptions per processor. So, it will be the selected heuristic for our parallel algorithm. The first stage is the same in the three methods, searching for equal weights and capacities in W and C. In the second stage, two cases are distinguished:

1. Element w_1 of W is smaller than the greatest capacity (c_1). In this case, the LDS strategy is applied. Therefore, a capacity c_j such as $c_j - w_1 > \min_{k \in D}\{w_k\}$ is searched. The processor with such a capacity that has received the minimum number of messages is selected.
2. Element w_1 is greater than capacity c_1. Then, the LRS strategy is used to minimize the maximum number of receptions. A processor with the minimum number of sent messages and a weight w_i verifying the condition $w_i - c_1 > \min_{k \in R}\{c_k\}$ is searched. This processor will send its extra load to the processor that needs c_1 columns to achieve the load balancing.

3 Estimated Cost of the Load Hybrid Strategy

In order to make the parallel algorithm efficient, the computational cost due to a load unbalancing and the cost due to the application of the LHS strategy have to

be estimated. Then, the parallel algorithm can decide whether the unbalancing effect is strong enough to apply the load balancing method or not.

If the system is unbalanced, the execution time of the parallel algorithm is determined by the processor with the greatest computational load. As the parallel code needs synchronizations after the computation of each rectangular block, the study of the imbalance can be reduced to these blocks. So, the cost due to imbalance per block can be defined as the time that the most overloaded processor needs to compute the excess of columns over the balancing situation (C_{max}). Considering that each column inside a block is composed by αP elements [6], being P the number of processors of the system, and $\alpha \in I\!N$ a parameter. The computational cost per block due to imbalance is,

$$\text{unbalancing(block)} = C_{max} \cdot (a_u + b_u \cdot \alpha P) \ . \tag{3}$$

The cost of the LHS strategy can be established as

$$\text{cost(LHS)} = T1 + T2, \tag{4}$$

where T1 is the execution time of the LHS heuristic, and T2 is the communication time, defined as the time spent by the processor with the maximum number of sends or receptions obtained by the LHS heuristic.

T1 depends on the number of times that W and C are swept. As the number of messages can not be known until the strategy is applied, we have decided to assume the worst case to model it, i.e. when the number of messages is equal to P-1. Therefore, as each iteration means a message, the estimated cost will be,

$$T1 = \text{cost(iter)} \cdot (P - 1) \ . \tag{5}$$

To compute T2, it is necessary to know the maximum number of sends or receptions and the cost of a message. The number of messages per processor is not known a priori. However, a study over a wide set of examples has been made, with the result that for the LHS strategy the more probable value for the maximum number of sent messages or receptions is 3. So, we assume that the number of messages per processor is 3. Therefore, the communication time will be modeled as,

$$T2 = 3 \cdot \text{cost(message)} \ . \tag{6}$$

The cost of a message depends on its size. Different linear fits $a_{men} + b_{men} \cdot n$, being n the message size, have been obtained for the AP3000 [1]. Then, the cost of a message can be estimated when its size is known. Assuming that the excess load is C_{max}, and being 3 the maximum number of messages that a processor sends or receives, each message will have an estimated size of,

$$n = \text{(elements/column)} \cdot \text{(columns)} \cdot \text{(bytes/element)} = \alpha P \cdot \lceil \frac{C_{max}}{3} \rceil \cdot 8 \ . \tag{7}$$

The complexity of the algorithm is $O(P \ log \ P)$.

4 Results

A wide set of 1740 synthetic examples was used to study the efficiency of the proposed strategies. A comparison with the Bestfit (BF) and Worstfit (WF) heuristics, and a greedy method called *Pairs Inside While Loop* (PIWL) [4] can be seen in Table 1. For each method, a column with the total number of times, over the 1740 examples, that the best solution of the optimization problem is obtained is shown. In most of the cases, this solution is achieved for various heuristics at the time. However, in some occasions, only one method obtains the best result. The number of times that this happens is shown in parenthesis. Note that as the number of processors increases, our strategies achieve the best solution most number of the times, outperforming dramatically WF and BF methods. The heuristic that most times achieves the minimum value of the objective function is highlighting in bold.

Table 1. Number of times over the 1740 examples that each heuristic obtain the best solution of the optimization problem

P	LDS	LRS	LHS	PIWL	BF	WF
4	290	290	290	290	290	281
8	230	226(6)	**232**	231	193(10)	181(15)
16	239(5)	**247(14)**	242	238	84(6)	81(9)
32	230(19)	**234(31)**	227	226	27(2)	26(3)
64	202(27)	**214(32)**	211(4)	190(1)	28(4)	16
128	179(40)	179(32)	**202(10)**	173(5)	24(3)	15
TOTAL	1370	1390	**1404**	1348	646	600

On the other hand, the accuracy of the estimation cost of our load balancing strategy has been validated in the AP3000 with 8 processors. The quality of the estimation of the unbalancing cost and the LHS strategy is verified for different sizes of the rectangular blocks. As an example, results for $\alpha = 4$ and $\alpha = 10$ are shown in Fig. 1.

Analyzing the results, we can conclude that for small values of α, the estimated cost of the LHS strategy differs from the heuristic real cost in case of great imbalance. However, as α increases, the LHS strategy predicts more accurately the real heuristic behavior. The reason is that in a balanced situation, each processor has α columns. So, for small α (Fig. 1(a)), great imbalance (high C_{max} values) implies that many processors lose its load completely. Therefore, the greatest loaded processor has to send probably more than 3 messages to restore balance. For large values of α (Fig. 1(b)), it is reasonable to estimate the total number of messages as 3. However, as our estimation is highly accurate for small imbalances, independently of α, the intersection point of the estimated imbalance and LHS strategy cost lines is very close to the real intersection point. The intersection is achieved for small unbalancing values, being even smaller as

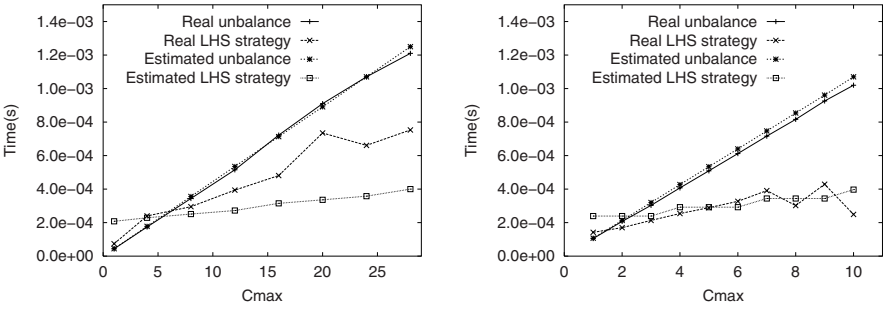

Fig. 1. Comparison between real and estimated cost of the LHS strategy for different block sizes. (a) $\alpha = 4$, (b) $\alpha = 10$

α or P increases, because in this case the computational load is greater and the imbalance will be more significant and expensive. Therefore, it is convenient to execute the load redistribution in almost every case.

5 Conclusions

In this work three load balancing strategies are proposed. Our methods focus on minimizing the maximum number of sends and receptions in order to reduce the communication cost. Improvements with respect to WF, BF and PIWL heuristics have been achieved. An estimation of the cost due to imbalances and of the cost of our LHS strategy has been established to know when it is better to redistribute the computational load. The comparison between them and the estimated results shows the high accuracy of the prediction. Moreover, it is proved that the load redistribution will be realized in practically all the cases.

References

1. Blanco, V. et al.: Performance of Parallel Iterative Solvers: a Library, a Prediction Model and a Visualization Tool. Journal of Information Science and Engineering, Vol. 8 **5** (2002) 763–785.
2. Gill, Philip E., Murray, Walter, Wright, Margaret H.: Practical Optimization. Academic Press, London (1981).
3. Gill, Philip E., Murray, Walter, Wright, Margaret H.: Numerical Linear Algebra and Optimization. Volume 1. Addison-Wesley Publishing Company, California (1991).
4. Haglin, David J., Ford, Rupert W.: The Message-Minimizing Load Redistribution Problem. Journal of Universal Computer Science, Vol. 7 **4** (2001) 291–306.
5. Martello, Silvano, Toth, Paolo: Knapsack Problems: Algorithms and Computer Implementations. Wiley Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York (1990).
6. Pardines, I., Rivera, F.F.: Parallel Quasi-Newton Optimization on Distributed Memory Multiprocessors. Parallel Computing: Advances and Current Issues. Proceedings of ParCo2001. Imperial College Press, London (2002) 338–345.