Database Program Mapping onto a Shared-Nothing Multiprocessor Architecture: Minimizing Communication Costs

Sophie Bonneau, Abdelkader Hameurlain

Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier
118, route de Narbonne, 31062 Toulouse cedex, France
E-mail: {bonneau, hameur}@irit.fr

Abstract. This paper deals with the minimization of inter-task and inter-processor communication costs in the context of one SQL query mapping onto a shared-nothing architecture, as far as the parallel decisional query processing is concerned. After setting both the models of the application handled and the target multiprocessor architecture, a study aimed at minimizing both inter-processor transfer times depending on the interconnection network topology, and the impact of pipeline starting/closing (start-up) times, is mainly presented.

1 Introduction

In the context of the compilation of a database program on a parallel architecture, the present paper focusses on the particular problem of mapping tasks making up a SQL query onto shared-nothing parallel architecture processors.

The mapping problem, generally as well as for many special cases known to be NP-complete, has been intensively studied within the database community, further to parallel DBMS development, for these last few years [7, 8, 10]: typically, the double optimization criterion of the proposed resolution methods (such is also our case) is the response time (or makespan as well) minimization and the system throughput maximization, and the allocation constraint, processor load partitioning or balancing.

Nevertheless, these heuristics are all distinguished by not taking into account communication costs, indeed estimated on the basis of inter-processor and inter-task communication times, inherent in the target parallel architecture specificity and the application type features, while these contribute to the increase in the response time [2]: first, a shared-nothing parallel system may be characterized by a sizeable inter-processor communication cost overhead, and second, the considerable communication (data and control) volume exchanges of the decisional SQL query bound tasks generate significant inter-task communication cost. On the one hand, inter-task communication times can be improved by reducing communication volume: at the mapping level, it has been achieved by introducing allocation constraints which facilitate propagating the number of processors and partitioning attributes [3] to avoid data redistributing whose cost may significantly exceed a task execution cost, and force mapping of the tasks in charge of reading data from disk, at the very place where data they use are stored. However, on the other hand, inter-processor communication times depend on both the target multiprocessor architecture interconnection network topology, and the available different parallelism (partitioned and pipeline) types taken into account in the application mapping process.

This is the reason why the problem of inter-processor communication cost minimization is here particularly focussed on, within the context of one SQL query static mapping onto a shared-nothing parallel architecture [5]. In Section 2, a model for the application to be mapped and the target parallel system, are described. In Section 3, our study of inter-processor communication cost optimization is summed up, as well as its consequences on the mapping process to be designed. Finally, Section 4 concludes with a synthesis of the work achieved and an overview of future related prospects.

2 Problem Modelization

The application to be mapped is a *decisional SQL query dependence graph* [9]: this is an oriented graph whose nodes represent tasks (i.e. Scani, Buildi and Probei[1]), and edges, information (data and control messages) communication and/or time-related dependence links. The communication link between two tasks I and J is either of the "precedence" or of the "pipeline" type: in the former case, J will be executed and will receive information from I only when I has completed its execution, whereas, in the latter case, I and J exchange data flow in a producer-consumer manner. Only tasks Buildi and Probei are constrained by a precedence type communication. In addition, this graph is valuated by each task local response time, each task couple communication cost (see Section 3), and the number of processors required to execute each task, which is estimated by the cost evaluator [4]. The communication mode of each bound task couple is also known (i.e. broadcast, distribution or propagation).

The target *shared-nothing parallel architecture* is characterized by the number of its processors, and its interconnection network topology (i.e. point-to-point, multi-stage or fully connected). It is to be noted that communications, depending on the network topology and the required data partitioning on each memory space, may generate a sizeable parallelism overhead.

3 Inter-processor Communication Cost Minimization

3.1 Inter-processor Communication Time Definition and Minimization Approach

Nowadays, trying to minimize inter-processor communication times on shared-nothing parallel architecture, still seems to be relevant. Multi-stage or fully connected type networks have, indeed, uniform communication times unlike point-to-point networks, and new technologies (i.e. optic fibre, co-axial cable support) cannot account for a minimal Communication Time/CPU Time ratio yet, for the number of MIPS has not improved in the same proportion as commmunication latency and rate have decreased.

Generally, the inter-processor communication time between two processors refers to

1. i.e. tasks respectively reading tuples from disk, building and probing the hash table of a simple-hash join node.

$T_{comm} = T_{transfer} + T_{start\text{-}up}$, where $T_{transfer}$ is the information (data and control) transfer time between two processors, and $T_{start\text{-}up}$, the information receiving/sending (start-up) CPU time overhead. On the one hand, in order to minimize the former component while mapping, when a point-to-point topology is at stake, "bringing nearer" emitter and receiver processors involved in communication can be attempted by using a distance matrix. On the other hand, as the latter component can prevail in target parallel system enabling pipeline parallelism to be exploited (such is our case), and because our application data are transferred in flow, minimizing of pipeline start-up time impact is essential, and based on the following observation: as a rule, in order to maximize pipeline potential chains, it is attempted to allocate two different processor sets to each pipeline communicating task couple. Well, the communication cost (estimated from data communication volume) between two of these tasks can sometimes exceed their execution cost (based on local response time) because some tasks such as Scan, Join, Grouping, don't always handle all the data received [2]. In this case, if only one processor is supposed to be allocated to each task, it is to be noted that the pipelined task couple response time mapped onto two different processors, is less than the pipelined task couple response time mapped onto the same processor, only if the communication cost is relatively less than the task local response times. As a consequence, taking into account partitioned parallelism (i.e. assigning several processors to each task) sets the question of whether it is always relevant to systematically allocate two different processor sets to pipelined tasks. Our resolution method consists, for each pipelined task couple, in analyzing associated costs to determine whether it should be better, in terms of mapped task couple response time, to allocate onto the same processor set or onto two different processor sets.

3.2 Resolution Method Mainspring

Let two tasks I and J, with respective local response times $t_i>0$ and $t_j>0$, and pipeline communication cost $C_{i,j}>0$, which respectively require $n_i>1$ and $n_j>1$ processors. Depending on the inter-task communication mode and the number of processors assigned to each of them, the following cases can be distinguished:

1st case (see Figure 1):

* $n_i = n_j$ (for instance, I and J are both assigned 2 processors.)

* communication mode: propagation

2nd case (see Figure 2):

* $n_i \neq n_j$ (for instance, I and J are respectively assigned 2 and 3 processors.)

* communication mode: distribution or broadcast

Our method consists in attempting to *formally evaluate* this *task couple response time* for each of the mapping configurations [1], under the following assumptions: i/ The couple tasks to be mapped will be scheduled in a balanced and symmetric way [6], ii/ all processors are identical.

3.3 Formal Study Consequences and Interpretation

The following conclusions can be drawn from the response time study of a task couple

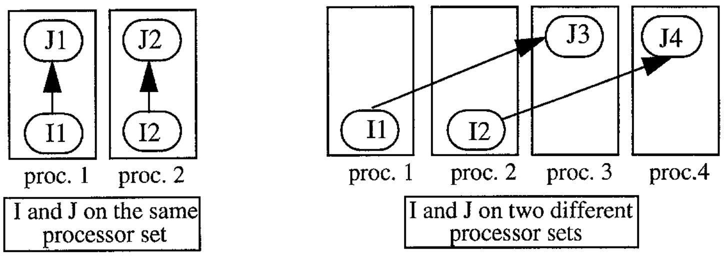
J1
I1
J2
I2
proc. 1
proc. 2
I and J on the same processor set
I1
I2
J3
J4
proc. 1
proc. 2
proc. 3
proc.4
I and J on two different processor sets

Fig. 1. Possible mapping configurations of pipelined tasks propagating

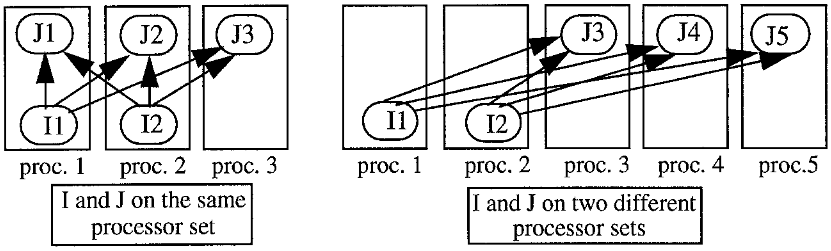

Fig. 2. Possible mapping configurations of pipelined tasks distributing or broadcasting

in each mapping configuration, depending on communication mode, towards the application dependence graph:

i/ tasks distributing or broadcasting (e.g. Scani→Buildj, Scani→Probej and Probei→Buildj if the number of processors and partitioning attributes were not propagated), should always be mapped onto two different processor sets, to exploit pipeline parallelism. In this case, it can, indeed, be proved that the response time of the task couple mapped onto two different processor sets will always be less than the response time of the task couple mapped onto the same processor set, whatever the communication cost (start-up) significance with regard to task execution costs [1].

ii/ tasks propagating (e.g. Probei→Buildj if the number of processors and partitioning attributes were propagated), should be mapped onto two different processor sets, to favour pipeline parallelism, when the communication cost is relatively less than the task execution cost. On the contrary, multi-programming execution would better be favoured by mapping tasks onto the same processor set. Decision relativity can be settled by the following fast calculation:

let $\boldsymbol{R} = \boldsymbol{max(t_i, t_j) - (t_i + t_j)}$ (negative by definition for t_i, $t_j > 0$)

1st case. $C_{i,j} > |R|$: the communication cost will compensate saving due to pipeline parallelism, i.e. the communication cost will inhibit saving due to pipeline execution. Therefore, task multi-programming execution would better be favoured by mapping onto the same processor set.

2nd case. $C_{i,j} \leq |R|$: saving due to multi-programming execution is compensated by the communication cost. Pipeline execution would better be favoured by mapping tasks onto two different processor sets.

It is to be noted that the pipeline communication format Probei→Probej is not deliberately considered in this work: the data localization allocation constraint [5] binds

Probei (resp. Probej) mapping to be Buildi (resp. Buildj) one. So, the mapping clue for such a task couple will a priori be physical allocation onto two different processor sets.

4 Conclusion

This paper has focussed on the problem of minimization of inter-task and, more particularly, inter-processor communication costs, which contributes to the optimization criterion minimization (i.e. response time) of one SQL query mapping onto shared-nothing parallel architecture. After describing a problem modelization, this study and an adequate resolution method have been presented.

So it has been emphazised that the mapping process could mainly act on the inter-processor communication cost by attempting to:

i/ minimize inter-processor transfer time for point-to-point networks, by choosing to allocate sets of processors which are nearer each other to communicating tasks, thanks to a distance matrix,

ii/ minimize pipeline communication start-up time impact, by working particularly on dependence graph pipeline communications. For each pipelined task couple, this handling determines whether it would be better, in terms of mapped task couple response time and depending on the communication mode, to allocate onto the same processor set or onto two different processor sets.

Finally, the next goal is to integrate these results into the PSA (Parallel Scheduling Algorithm) scheduling and simultaneous mapping algorithm [5].

References

1. BONNEAU, S. et al., “Placement d’un programme bases de données sur une architecture multiprocesseur à mémoire distribuée: minimisation des coûts de communication”, Rapport de Recherche, No. IRIT/97-08-R, Lab. IRIT, Janvier 1997, 23 pages.
2. ENGLERT, S., et al., “Parallelism and its price: a case study of NonStop SQL/MP”, Sigmod Record, vol. 24, n°4, Dec. 1995, pp. 61-71.
3. HAMEURLAIN, A., et al., " An Optimization Method of Data Communication and Control for Parallel Execution of SQL Queries ", Intl. Conf. on Database and Expert Systems Applications, LNCS, n°720, Prague, Sept. 6-9, 1993, pp. 301-312.
4. HAMEURLAIN, A., et al.," A Cost Evaluator for Parallel Database Systems ", 6th Intl. Conf. on Database and Expert Systems Applications, DEXA'95, London, 4-8 Sept. 1995, LNCS, n°978, pp. 146-156.
5. HAMEURLAIN, A., et al., "Scheduling and Mapping for Parallel Execution of Extended SQL Queries", 4th Intl. Conf. on Information and Knowledge Management, ACM Press, Baltimore, Maryland, 28 Nov. - 2 Dec. 1995, pp. 197-204.
6. HASAN, W., “Optimization of SQL queries for parallel machines”, dissertation fot the degree of Doctor of Philosophy, Stanford University, Dec. 1995.
7. LO, Y.L., et al., “Scheduling queries for parallel execution on multicomputer DataBase Management System”, 7th Intl. Conf. on Database and Expert Systems Applications, DEXA'96, Zurich, Sept. 1996, LNCS, n°1134, pp. 698-707.
8. MAHIOUT, A., et al., “Modéliser les dépendances entre les tâches data-parallèles pour le placement et l’ordonnancement automatique”, 6ièmes Rencontres Francophones du parallélisme, Lyon, Juin 1994, pp. 37-40.
9. SCHNEIDER, D., et al.,“Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines”, Proc. of the 16th VLDB Conf., Brisbane, Australia 1990, pp. 469-480.
10. WOLF, J. L., et al., “A Hierarchical Approach to Parallel Multiquery Scheduling”, IEEE Transactions on Parallel and Distributed Systems, 1995, vol. 6, n°6, pp. 578-589.