CENTER FOR
PARALLEL OPTIMIZATION

OPTIMAL PROCESSOR ASSIGNMENT FOR PARALLEL DATABASE DESIGN

by

Shahram Ghandeharizadeh
Robert R. Meyer
Gary L. Schultz
Jonathan Yackel

Computer Sciences Technical Report #1022

May 1991

OPTIMAL PROCESSOR ASSIGNMENT FOR PARALLEL DATABASE DESIGN*

SHAHRAM GHANDEHARIZADEH†, ROBERT R. MEYER‡, GARY L. SCHULTZ‡ AND JONATHAN YACKEL‡

Abstract. The computing time benefits of parallelism in database systems (achieved by using multiple processors to execute a query) must be weighed against communication, startup, and termination overhead costs that increase as a function of the number of processors used. We consider problems of minimizing overhead subject to allocating data among the processors according to specified loads. We present lower bounds for these combinatorial problems and demonstrate how processors may be optimally assigned for some problem classes.

1. Introduction. In highly-parallel database machines (e.g., Gamma [2], Bubba [1], Non-Stop SQL [12], XPRS [11] and Volcano [6]) relations are partitioned across multiple processors. (Livny *et al* [9] and Ries and Epstein [10] introduced the related concept of "horizontal" partitioning.) This allows each processor to execute a portion of a query in parallel with the other processors, resulting in a lower response time for the query. However, there is communication overhead associated with initiating and terminating a query on multiple processors, and this overhead increases as a function of the number of processors used to execute a query [1]. In order to minimize overhead while balancing the workload among the processors, Multi-Attribute GrId deClustering (MAGIC) introduced by Ghandeharizadeh [3] partitions a relation by assigning ranges of several attribute values to each processor in the system. To illustrate MAGIC consider the partitioning of the Employee relation EMP in figure 1.

* This research was partially supported by the Air Force Office of Scientific Research under grant 89-0410, the Defense Advanced Research Projects Agency under contract N00039-86-C-0578, and by the National Science Foundation under grants CCR-8907671 and DCR-8512862.

† Computer Science Department, University of Southern California, Los Angeles, California 90089-0782.

‡ Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706.

[1] This overhead is primarily in the form of additional messages to control the execution of the query on additional processors and, in the Gamma database machine [2], increases linearly with the number of employed processors.

		Salary in $K		
		0-20	20-50	> 50
Age in Years	0-25	1	1	2
	26-50	1	3	3
	> 50	2	3	2

FIG. 1. *Processor assignment for the* EMP *relation*

For parallel computation, MAGIC partitions the EMP relation by establishing ranges of Salary and Age attribute values. Each cell of this grid corresponds to a fragment of the relation and must be assigned to some processor. For example, the cell which contains records with Salary attribute values that range from 0 to 20 and Age attribute values that range from 26 to 50 is assigned to processor 1. Given a query on either the Age or Salary attribute, the predicate of the query maps to either a row or a column (termed a "slice") of the grid and the corresponding processors are used to execute it. Note that, for the assignment depicted by figure 1, every processor is assigned three cells, and every query requires two processors.

Although we concentrate on the limiting case in which overhead is minimized, the optimal processor assignments that we obtain below have properties that suggest that they may also be good approximations to assignments that would minimize other response time functions. For example, suppose the response time r for an average query as a function of the number of processors ν used by the query is modeled by $r(\nu) = \mathcal{O}\nu + \mathcal{Q}/\nu$, where $\mathcal{O}$ is the overhead per processor and $\mathcal{Q}$ is the processing time for a query on a single processor, *i.e.* the overhead increases proportionally with the number of processors, while the processing time is inversely proportional to the number of processors[2]. In the absence of any constraints, r is minimized when the number of processors per query is $\nu^* = \sqrt{\mathcal{Q}/\mathcal{O}}$. We shall see in (§3) that for our version of the problem, which in some sense minimizes the number of processors used per query (subject to a load balancing constraint), optimal assignments nevertheless have $\approx \sqrt{N}$ processors assigned to each query, where N is the number of processors in the system. If $\sqrt{\mathcal{Q}/\mathcal{O}} \leq \sqrt{N}$, which is the case if the communication overhead for using *all* N processors dominates the processing time for a single query, then our optimal solution comes as close as possible (among assignments that balance the workload among processors) to the *unconstrained* minimum of the alternative objective r.

2. Basic Mathematical Problem Statement. Suppose that we wish to assign the cells of a D-dimensional grid to N processors, and that the size of the grid is $M_1 \times M_2 \times \ldots \times M_D$ (*i.e.*, the dth attribute is partitioned into M_d ranges). Let $V := \prod_d M_d$ denote the number of cells (volume) of the grid. A "slice" is a $(D-1$-dimensional) subgrid containing all the cells with a common value for a given coordinate (this corresponds to a query). For example, in an $M_1 \times M_2$ grid the slices

[2] The linear speedup results presented in [2] justify this assertion.

are the M_1 rows and the M_2 columns, and in an $M_1 \times M_2 \times M_3$ grid the slices are the $M_1 + M_2 + M_3$ two-dimensional subgrids. Let $\mathcal{S}$ denote the collection of slices.

Given an assignment of cells to processors and an arbitrary slice s of the grid , let ν_s denote the number of *distinct* processors in the slice s. Given a processor p, let load_p denote the number of cells assigned to p. The objective function for the optimization problem that we develop measures total or worst case overhead. $\theta_{\text{total}} := \sum_{s \in \mathcal{S}} \nu_s$ and $\theta_{\text{max}} := \max_{s \in \mathcal{S}} \nu_s$. Note that if each slice has the same frequency of access and we are interested in minimizing the *average* query overhead, then we should minimize θ_{total}. If, on the other hand, we are interested in minimizing the *worst case* overhead incurred by *any query*, then θ_{max} should be minimized. The balancing constraints are defined by specifying a load for each processor. (In typical applications, we require that the loads are equal or differ by at most 1.) The problem may be formally stated as follows:

Let the following data be given: a dimension D, a number of processors N, the cardinality vector M of the partitions in each dimension, and a load for each processor. Find an assignment that

$$\begin{aligned} &\textit{minimizes } \theta \quad (\textit{where } \theta \textit{ is chosen as } \theta_{total} \textit{ or } \theta_{max}) \\ &\textit{s.t.} \quad \textit{processor } p \textit{ is assigned } \text{load}_p \textit{ cells } (p = 1, 2, \ldots, N). \end{aligned}$$

The number of assignments satisfying the balancing constraint is $\frac{V!}{\prod_{p=1}^{N} (\text{load}_p!)}$. Complete enumeration of these assignments is not feasible even for relatively small problems. For example, given a 5×5 grid, 5 processors, and a load of 5 for each processor, there are $623,360,743,125,120$ assignments that satisfy the balancing constraint.

A similar class of data aggregation problems was studied by Helman [8]. Suppose that we replace our notion of "slice" by "arbitrary subset", *i.e.*, the problem data consist of the grid plus a collection of subsets of cells. Then the problem of minimizing the total or maximum number of distinct processors in the subsets (with equal loads for all processors) corresponds to one of Helman's *K-size aggregation problems*, (with $K = V/N$) which he shows to be NP-complete.

3. Lower Bounds. In this section we will develop lower bounds on the measures θ_{total} and θ_{max}. Throughout this section, the following notation is used. χ_s^p indicates whether processor p appears in slice s: $\chi_s^p = 1$ if p appears in slice s and $\chi_s^p = 0$ otherwise. (Notice that ν_s, the number of distinct processors in slice s, can be represented in terms of χ_s^p: $\nu_s = \sum_p \chi_s^p$.) σ_p denotes the number of slices containing processor p: $\sigma_p = \sum_s \chi_s^p$, and $\sigma_{p,d}$ is the number of slices in dimension d containing processor p.

A key relationship in the development of the lower bounds is

$$\sum_{\text{slices in dim } d} \nu_s = \sum_{p=1}^{N} \sigma_{p,d} \tag{1}$$

i.e. the sum over the slices in a particular dimension of the numbers of distinct processors in the slices is the same as the sum over all processors of the numbers of slices in that dimension in which the processors appear.

3.1. Lower bounds on θ_{total}. It is useful to consider the overhead measure θ_{total} from two different points of view: $\theta_{\text{total}} = \sum_s \nu_s = \sum_p \sigma_p$. We first derive a lower bound on σ_p.

LEMMA 3.1. *For any processor p (assigned to load_p cells),* $\left\lceil D \ \text{load}_p^{\frac{1}{D}} \right\rceil \leq \sigma_p$.

Proof. In each dimension, permute the slices in that dimension so that the slices containing p come first. Notice that the first $\sigma_{p,d}$ slices in dimension d will contain p. Notice also that rearranging the order of the slices does not alter any of the $\sigma_{p,d}$. Then the $\sigma_{p,1} \times \sigma_{p,2} \times \ldots \times \sigma_{p,D}$ box in the upper corner of the grid contains all of the cells assigned to p. Therefore the "volume" $\prod_{d=1}^{D} \sigma_{p,d}$ of this box is at least load_p, i.e.,

$$\text{load}_p \leq \prod_{d=1}^{D} \sigma_{p,d}. \tag{2}$$

Taking the Dth root of both sides of (2) and applying the arithmetic mean/geometric mean inequality (see Hardy *et al* [7]) we obtain $(\text{load}_p)^{\frac{1}{D}} \leq \left(\prod_{d=1}^{D} \sigma_{p,d}\right)^{\frac{1}{D}} \leq \frac{1}{D} \sum_{d=1}^{D} \sigma_{p,d} = \sigma_p/D$, whence $D\,(\text{load}_p)^{\frac{1}{D}} \leq \sigma_p$. Since the RHS of the last inequality is integral, we may take the ceiling of the LHS. □

THEOREM 3.2. $\sum_p \left\lceil D \ \text{load}_p^{\frac{1}{D}} \right\rceil \leq \theta_{total}$.

Proof. Use lemma 3.1 and the fact that $\theta_{\text{total}} = \sum_p \sigma_p$. □

In §4 we give cases in which the lower bound of theorem 3.2 is tight.

3.2. Lower Bounds on θ_{max}. At this point we prove a lemma that is used in deriving lower bounds on θ_{max}. Let $\tilde{\nu}_d$ be the average of the ν's for the slices in dimension d *i.e.* $\tilde{\nu}_d := \sum_{\text{slices in dim } d} \nu_s/M_d$.

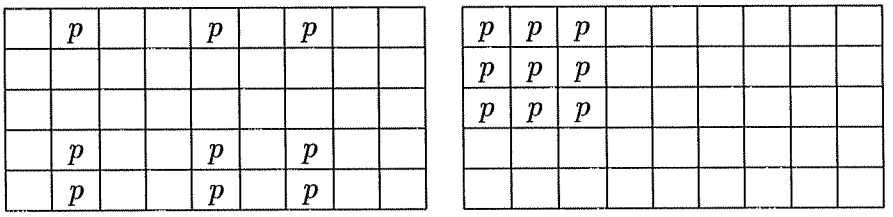
LEMMA 3.3. $\prod_d \tilde{\nu}_d \geq \frac{N^D \prod_p \text{load}_p^{\frac{1}{N}}}{V}$.

Proof. Using inequality (2) and the arithmetic mean-geometric mean inequality, $\prod_p \text{load}_p^{\frac{1}{N}} \leq \prod_d \left(\prod_p \sigma_{p,d}\right)^{\frac{1}{N}} \leq \prod_d \frac{\sum_p \sigma_{p,d}}{N}$. By equation (1), $\frac{N^D \prod_p \text{load}_p^{\frac{1}{N}}}{V} \leq \prod_d \tilde{\nu}_d$. □

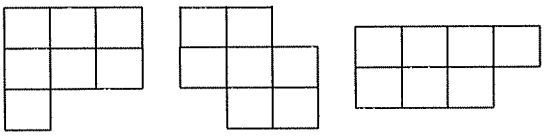
THEOREM 3.4. $\left\lceil N \left(\prod_p \text{load}_p^{\frac{1}{N}}/V\right)^{\frac{1}{D}} \right\rceil \leq \theta_{max}$.

Proof. Lemma 3.3 gives a bound on a product of averages of ν's, so one ν must be at least as big as the ceiling of the Dth root of the bound. □

4. Achieving the Bounds. In order to achieve the lower bound for θ_{total}, each processor must contribute exactly $\left\lceil D \ \text{load}_p^{\frac{1}{D}} \right\rceil$ to the objective function. For example, this will occur if processor p occupies a hypercubical block of cells with sides of length $= \text{load}_p^{\frac{1}{D}}$. Note that since interchanging slices in the grid does not affect the objective function, the slices of a processor's block do not have to be contiguous. For example both placements of processor p with a load of 9 below are equivalent and optimal for processor p.



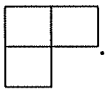
However, the block occupied by a processor may be irregular and still be optimal. Consider a 2-dimensional grid where a processor's load is 7. Then the lower bound is $\lceil 2\sqrt{7} \rceil = 6$. The following non-square blocks (and obvious variants) have height + width = 6, and are therefore optimal:



If optimal blocks for each processor can be interleaved so as to cover the grid then the lower bound on θ_{total} can be met. One class of problems that have easily obtainable optimal solutions are instances in which hyper-rectangular blocks are optimal for each processor and the grid can be covered by these blocks. Below we demonstrate an optimal assignment for such an instance: a 6×15 grid with 6 processors, each of which has a load of 15. (See Ghandeharizadeh *et al* [5] for a collection of classes for which optimal solutions are developed.)

1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
4	4	4	4	4	5	5	5	5	5	6	6	6	6	6
4	4	4	4	4	5	5	5	5	5	6	6	6	6	6
4	4	4	4	4	5	5	5	5	5	6	6	6	6	6

Another class of problems for which it is possible to construct optimal assignments for *both* θ_{total} and θ_{max} is the class of $N \times N$ grids with N processors, where each processor has a load of N. We have developed an algorithm that constructs optimal assignments for such problem instances [5]. Figure 1 is an example of such an assignment. Note that for any processor the slices in figure 1 may be permuted so

that the set of cells occupied has the following optimal shape:  .

5. Conclusions and Future Work. We have formalized the problem of partitioning data on a parallel database machine in order to minimize overhead. Lower bounds on the objective functions have been developed and we have demonstrated how the bounds can be attained in many cases. We would like to explore other approaches to the data partitioning problem. A branch and bound type of approach in a suitably restricted search space seems promising. We also have a nonconvex nonlinear programming formulation of the problem that suggests other solution techniques. Extending the work already done, the square grid assignment algorithm may generalize to more than 2 dimensions. In addition, we would like to deal with more general objective functions and load balancing constraints. Finally, it would be interesting to consider other applications that fit into the task assignment/parallel computing framework developed here (see, *e.g.*, Ghandeharizadeh *et al* [4]).

REFERENCES

[1] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez, *Prototyping Bubba, a highly parallel database system*, IEEE Transactions on Knowledge and Data Engineering, 2 (1990).

[2] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen, *The Gamma database machine project*, IEEE Transactions on Knowledge and Data Engineering, 2 (1990).

[3] S. Ghandeharizadeh, *Physical Database Design in Multiprocessor Systems*, PhD thesis, University of Wisconsin - Madison, 1990.

[4] S. Ghandeharizadeh, L. Ramos, Z. Asad, and W. Qureshi, *Object placement in parallel hypermedia systems.* to appear in the proceedings of the 1991 VLDB Conference.

[5] S. Ghandeharizadeh, G. L. Schultz, R. R. Meyer, and J. Yackel, *Optimal balanced assignments and a parallel database application*, Computer Sciences Technical Report 986, University of Wisconsin - Madison, Madison, WI, December 1990.

[6] G. Graefe, *Volcano: An extensible and parallel dataflow query processing system*, Computer Science Technical Report, Oregon Graduate Center, Beaverton, OR, June 1989.

[7] G. Hardy, J. Littlewood, and G. Polya, *Inequalities*, Cambridge, 1959.

[8] P. Helman, *A family of NP-complete data aggregation problems*, Acta Informatica, 26 (1989), pp. 485–499.

[9] M. Livny, S. Khoshafian, and H. Boral, *Multi-disk management algorithms*, in Proceedings of the 1987 ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems, May 1987.

[10] D. Ries and R. Epstein, *Evaluation of distribution criteria for distributed database systems*, UCB/ERL Technical Report M78/22, UC Berkeley, May 1987.

[11] M. Stonebraker, D. Patterson, and J. Ousterhout, *The design of XPRS*, in Proceedings of the 1988 VLDB Conference, Los Angeles, CA, September 1988.

[12] Tandem Performance Group, *A benchmark non-stop SQL on the debit credit transaction*, in Proceedings of the 1988 SIGMOD Conference, Chicago, IL, June 1988.