ON OPTIMAL PROCESSOR ALLOCATION TO SUPPORT PIPELINED HASH JOINS

Ming-Ling Lo*, Ming-Syan Chen†, C. V. Ravishankar* and Philip S. Yu†

EECS Department*
University of Michigan at Ann Arbor
Ann Arbor, MI 48109

IBM Thomas J. Watson Research Center†
P.O.Box 704
Yorktown Heights, NY 10598

Abstract

In this paper, we develop algorithms to achieve optimal processor allocation for pipelined hash joins in a multiprocessor-based database system. A pipeline of hash joins is composed of several stages, each of which is associated with one join operation. The whole pipeline is executed in two phases: (1) the table-building phase, and (2) the tuple-probing phase. We focus on the problem of allocating processors to the stages of a pipeline to minimize the query execution time. We formulate the processor allocation problem as a two-phase mini-max optimization problem, and develop three optimal allocation schemes under three different constraints. The effectiveness of our problem formulation and solution is verified through a detailed tuple-by-tuple simulation of pipelined hash joins. Our solution scheme is general and applicable to any optimal resource allocation problem formulated as a two-phase mini-max problem.

1 Introduction

In recent years, multiprocessor-based parallel database machines have attracted considerable attention from both the academic and industrial communities because they can efficiently execute complex database operations [1] [5] [10] [17]. In relational database systems, joins are the most expensive operations to execute, especially with the increases in database size and query complexity [3] [14] [19]. Many applications usually need to specify the desired results in terms of multi-join queries, some of which may take hours or even days to complete. As a result, parallelism has been recognized as the only solution for the efficient execution of multi-join queries for future database management [6] [7] [8] [12].

*Authors in this work were supported in part by the Consortium for International Earth Science Information Networking.

Among various join methods, the hash join has been the focus of much research effort and reported to have performance superior to that of others, particularly because it presents an opportunity for pipelining [4] [13] [15] [18]. Using hash joins, multiple joins can be pipelined so that the early resulting tuples from a join, before the whole join is completed, can be sent to the next join for processing. A pipeline of hash joins is composed of several stages, each of which is associated with one join operation that can be executed, in parallel, by several processors. Though pipelining has been shown to be very effective in reducing the query execution time, prior studies on pipelined hash joins have focused mainly on heuristic methods for query plan generation. Most of the prior work on query plan generation, such as static right-deep scheduling, dynamic bottom-up scheduling [16], and segmented right-deep trees [2][1], resorted to simple heuristics to allocate processors to pipeline stages. Also, these methods dealt with only memory as a constraint for the execution of pipelined hash joins. Little effort was made to take processing power into consideration and optimize processor allocation. Notice that two opportunities for parallelism exist for pipelined hash joins: Not only can each hash join be implemented by several processors, but also several joins can be pipelined. As a result, processor allocation arises as an important unexplored issue in the performance of pipelined hash joins. In view of this fact and the increasing demand for better performance of database operations, the objective of this paper is to study and improve processor allocation for pipelined hash joins.

In this paper we derive optimal processor allocation algorithms that take both memory constraint and processing power into account. We assume that a pipeline of hash joins is given a priori, which can be generated based on the approaches in [2] [16]. A pipeline of hash joins is sequentially executed in two phases:

[1]Segmented right-deep trees are bushy trees with right-deep subtrees [2].

(1) the table-building phase and (2) the tuple-probing phase. In the table-building phase, hash tables are constructed from inner relations using hash functions on join attributes. In the tuple-probing phase, tuples of the outer relation are used to probe the hash tables for matches. Note that the processing time of each phase is determined by the maximal execution time among all stages, and that the same allocation of processors to a stage is retained across the table-building and tuple-probing phases. The execution time of a pipeline is thus the sum of two correlated maxima. The characteristics of pipelined hash joins allow the processor allocation problem to be formulated as a two-phase mini-max optimization problem. Specifically, for a pipeline with k stages, the execution time of the pipeline, TS, can be expressed as

$$TS = \max_{0 \leq i \leq k-1} \frac{WB_i}{n_i} + \max_{0 \leq i \leq k-1} \frac{WP_i}{n_i}, \qquad (1)$$

where WB_i and WP_i are, respectively, the workloads for the table-building and tuple-probing phases in stage i. Consequently, the processor allocation problem for pipelined hash joins can be stated as follows: "Given WB_i and WP_i, $0 \leq i \leq k-1$, determine the processor allocation $\langle n_i \rangle = (n_0, n_1, \ldots, n_{k-1})$, so as to minimize TS in Eq.(1), where n_i is the number of processors allocated to stage i."

For example, consider the workloads shown in Table 1 for a pipeline of five stages. First, it is observed that the workloads of stage 2 are less than those of stage 3, suggesting that stage 3 should be assigned more processors than stage 2. However, stage 3 has a heavier load in the table-building phase than stage 4, while the latter has a heavier load in the tuple-probing phase. In such a configuration, there is no obvious way to allocate processors to minimize the pipeline execution time specified in Eq.(1). For an illustrative purpose, suppose the total number of processors available to execute the pipeline in Table 1 is 20. It can be seen that the allocation $\langle n_i \rangle$= (4, 4, 4, 4, 4) leads to $\max_{\forall i} \frac{WB_i}{n_i}$= 1.5, $\max_{\forall i} \frac{WP_i}{n_i}$= 1.2, and TS = 2.7, whereas the one $\langle n_i \rangle$= (6, 3, 3, 6, 2), which is based on the workloads of the table-building phase, leads to $\max_{\forall i} \frac{WB_i}{n_i}$= 1.0, $\max_{\forall i} \frac{WP_i}{n_i}$= 2.5, and TS = 3.5. Clearly, to develop an optimal processor allocation to minimize TS in Eq.(1) is in general a very difficult and important problem. Since the table-building and tuple-probing phases are executed one after the other, we minimize the sum of two correlated maxima in Eq.(1). In view of this, the optimal processor allocation problem in Eq.(1) is hence termed the two-phase mini-max optimization problem. This feature distinguishes our allocation problem from other conventional resource allocation problems [9], which may be considered one-

stage	0	1	2	3	4
WB_i	6	3	3	6	2
WP_i	4	4	2	3	5

Table 1: The workloads in two phases of each stage.

phase mini-max optimization problems.

To develop the optimal processor allocation scheme for the two-phase mini-max optimization, we consider the following three constraints: (1) the total number of processors available for allocation is fixed, (2) a lower bound is imposed on the number of processors required for each stage to meet the corresponding memory requirement, and (3) processors are available only in discrete units. We develop solution schemes by incrementally adding constraints to our optimization problem. Specifically, three optimal processor allocation schemes are devised, i.e., one under Constraint (1), another under Constraints (1) and (2), and the third under all the three constraints. The three allocation schemes will be shown to be optimal under their corresponding constraints. Also, it can be verified that, for a system with N processors and k pipeline stages, the time complexity of the first two allocation schemes is $O(k \cdot \log k)$, and that of the third one is $O(Nk \cdot \log(Nk))$.

Generally speaking, a complex real world problem such as pipelining usually needs to be simplified and abstracted before it can be mapped into a mathematical model to further analyze and optimize. The contributions of this study are twofold. We not only formulate the optimal processor allocation problem for pipelined hash joins as a two-phase min-max problem, but also derive solutions to this new class of optimization problem. We verify the effectiveness of our problem formulation and solution procedure through a detailed simulation of pipelined hash joins. Simulation results show that the proposed allocation schemes lead to much shorter query execution times than conventional approaches. It is worth mentioning that our solution scheme is general and applicable to any optimal resource allocation problem formulated as a two-phase mini-max problem. Although the one-phase mini-max optimization problem has been studied extensively in the literature, this study, to the best of our knowledge, provides the first solution to the two-phase mini-max optimization problem.

This paper is organized as follows. Section 2 provides the background and the problem description. Section 3 develops three algorithms for deriving three optimal processor allocations. Proofs of lemmas and theorems can be found in [11]. In Section 4, we demonstrate the performance improvement achieved by our proposed scheme via simulation. The paper concludes with Section 5.

2 Preliminaries

2.1 Notation, Assumption and Definition

As in most prior work, we assume that execution time is the primary cost measure for estimating the efficiency of database operations. The architecture assumed is a multiprocessor system with distributed memory and shared disks. Each processing node (or processor) in the system has its own memory and address space, and communicates with others through message passing. Each node is assumed to have the same amount of memory, and the amount of memory available to execute a join is in proportion to the number of processors involved. The disks in the system are accessible by all nodes through shared I/O buses.

A pipeline of hash joins is composed of several stages, each of which is associated with one join operation. The relation in a hash join that is loaded into memory to build the hash table is called the *inner relation*, while the other relation, whose tuples are used to probe the hash table, is called the *outer relation*. The inner relations of a pipeline are the inner relations of its stages. The outer relation of a pipeline is defined to be the outer relation of its first stage. In the table-building phase, the hash tables of the inner relations are built using hash functions on join attributes. In the tuple-probing phase, tuples of the outer relation are first probed, one by one, against the entries in the hash table of the first stage using the corresponding hash function. If there are matches, the resulting tuples are generated, and then sent to the next stage for similar processing. The table-building and tuple-probing times of a pipeline are the time spans of the building and probing phases respectively. The execution of one pipeline is given in Figure 1 for illustration.

2.2 Problem Formulation

As pointed out earlier, the query processing time can be approximated as $T_{ins} + \max_{\forall i} \frac{WB_i}{n_i} + \max_{\forall i} \frac{WP_i}{n_i}$, where n_i is the number of nodes allocated to stage i and T_{ins} is the sum of those costs independent of processor allocation. It is also derived in [2] that WB_i is proportional to $|R_i|$, where $|R_i|$ is the size of the inner relation R_i at stage i of the pipeline[2]. Since T_{ins} is constant over all processor allocations, the objective function that we shall minimize is $TS = \max_{\forall i} \frac{WB_i}{n_i} + \max_{\forall i} \frac{WP_i}{n_i}$, the sum of costs dependent on processor allocation. In what follows, we shall concentrate on deriving the optimal processor allocation, $\lambda =$

[2]This relationship between WB_i and $|R_i|$ is useful in our derivation for optimal allocation in Section 3.2 later.

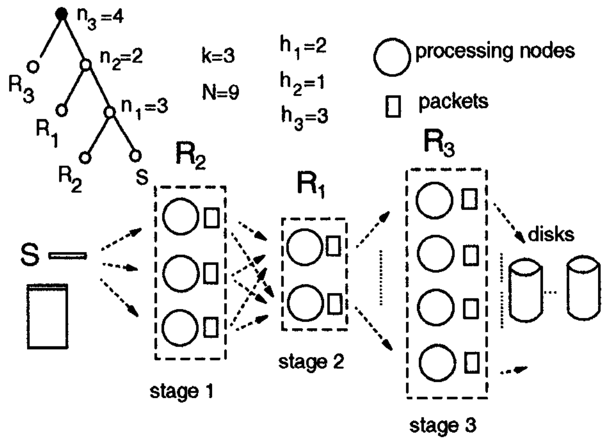

Figure 1: Execution of one pipeline.

$(n_0, n_1, \ldots, n_{k-1})$, to minimize TS, where k is the number of stages in the pipeline.

Note that the formula for TS does not depend on the order of stages in the pipeline. The workloads of the various stages in the pipeline determine the allocation. As mentioned earlier, we approach this problem by expressing it as an allocation problem with three constraints. We start with assuming only one constraint holds, and incrementally consider more constraints.

Constraint I (no idling): The total number of processors assigned to all stages is equal to the total number of processors in the system, i.e., $\sum_{i=0}^{k-1} n_i = N$.

Constraint II (sufficient memory): The amount of memory allocated to each stage must be large enough to accommodate the hash table of that stage, i.e., $n_i \geq \frac{|R_i|}{M}$, for all i, where M is the memory size of each processor.

Constraint III (discrete allocation): Processors must be allocated to stages in discrete units.

3 Optimal Processor Allocation

We shall develop three optimal processor allocation schemes in this section. The scheme in Section 3.1 corresponds to Constraint I (denoted by λ_I), that in Section 3.2 is subject to Constraints I and II (denoted by λ_{II}), and the one in Section 3.3 satisfies all the three constraints (denoted by λ_{III}).

3.1 Allocation under Constraint I

Under Constraint I, n_i, the number of processors allocated to stage i, could be any positive real number. We obtain the following necessary condition for λ_I, stating that each stage under λ_I must be a bottleneck in either the table-building or the tuple-probing phase. This is referred to as the *two phase allocation* condition.

Lemma 1: Let TB_i and TP_i be, respectively, the table-building and tuple-probing times for a stage i under λ_I. Then, either $TB_i = TB$, or $TP_i = TP$, where TB and TP are the pipeline building and probing times, respectively.

Since this lemma only defines the two phase allocation condition as a necessary condition, there may be some non-optimal allocations which also satisfy this condition. For every allocation, we identify an ordered pair of sets $\tau = (A, B)$ such that,

$$A = \{\text{stage } i \mid \frac{WB_i}{n_i} = TB\}, \quad (2)$$

$$B = \{\text{stage } j \mid \frac{WP_j}{n_j} = TP\}, \quad (3)$$

where $TB = \max_{\forall i} \frac{WB_i}{n_i}$ and $TP = \max_{\forall i} \frac{WP_i}{n_i}$. Specifically, A (respectively, B) is the set of stages that are bottlenecks in the table-building (respectively, tuple-probing) phase. Define

$$B' = B - (A \cap B) = \{\text{stage } j \mid \frac{WP_j}{n_j} = TP, j \notin A\}. \quad (4)$$

It follows from Lemma 1 that $A \cup B'$ comprises all stages in the pipeline. The problem of finding the optimal allocation can now be reduced to that of finding all (A, B), or (A, B'), satisfying the two phase allocation requirement in Eqs. (2) - (4), and then determining the one with the shortest processing time.

To avoid the exponential complexity of enumerating all possible two phase allocations, we shall first introduce the concept of *workload ratios (B/P ratios)* for the pipeline stages, and then in light of this concept, prove that the number of allocations needed to consider is no more than the number of pipeline stages. The B/P ratio for stage i is defined as $r_i = \frac{WB_i}{WP_i}$. We order the stages in descending order of their B/P ratios, and denote the sequence as $\vec{a} = (a_0, a_1, \ldots, a_{k-1})$. For the example profile in Table 1, we obtain Table 2 where stages are sorted according to their B/P ratios.

Let $WB_I = \sum_{i \in I} WB_i$, $WP_I = \sum_{i \in I} WP_i$, and $n_I = \sum_{i \in I} n_i$, where I denotes A, B or B'. An example of B' can be found in Figure 2. Then, under the two phase allocation condition, we have

$$TB = \frac{WB_i}{n_i} = \frac{WB_A}{n_A}, \forall i \in A, \quad (5)$$

$$TP = \frac{WP_j}{n_j} = \frac{WP_B}{n_B} = \frac{WP_{B'}}{n_{B'}}, \forall j \in B. \quad (6)$$

	$a_0 = 3$	$a_1 = 0$	$a_2 = 2$	$a_3 = 1$	$a_4 = 4$
WB_i	6	6	3	3	2
WP_i	3	4	2	4	5
r_i	2	1.5	1.5	0.75	0.4

Table 2: The workloads in two phases of each stage after sorting.

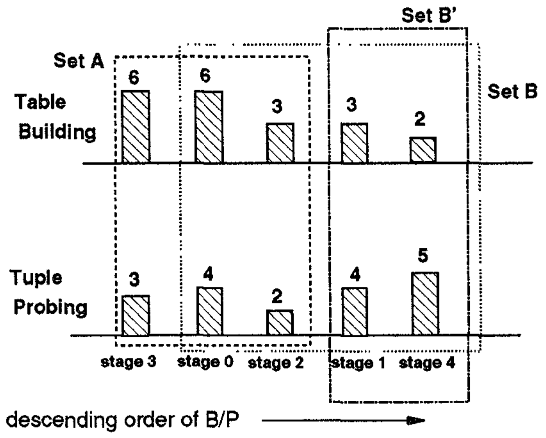

Figure 2: Example of a candidate ordered pair $\tau = (A, B)$ for a pipeline of 5 stages.

The pipeline processing time can be expressed as:

$$TS = \max_{\forall i \in A} \frac{WB_i}{n_i} + \max_{\forall i \in B} \frac{WP_i}{n_i}.$$

Then, we get,

$$TS = \frac{WB_A}{n_A} + \frac{WP_B}{n_B} = \frac{WB_A}{n_A} + \frac{WP_{B'}}{n_{B'}}, \quad (7)$$

where the total number of processors $N = n_A + n_{B'}$. If we select an A, thus fixing WB_A and $WP_{B'}$, TS can be expressed as a function of n_A.

Note that there is an upper bound and a lower bound on the amount of processors that one can assign as n_A (and $n_{B'}$) for a given pair of A and B. The reason is that Eq. (7) is valid only when A and B satisfy the conditions that they are the bottleneck sets in the table-building and the tuple-probing phases, respectively. If for a particular pair of sets A and B (and the associated WB_A and $WP_{B'}$), we allocate so many processors to stages in set A (i.e., assign so large a number to n_A) that the table-building times of stages in set A become shorter than those of some stages in set B, A would not be the bottleneck set of the table-building phase any more, and Eq. (7) would no longer give the correct pipeline processing time TS. Under such an allocation of processors, the processing time TS will be determined by different sets of A and B (and hence different WB_A and $WP_{B'}$). Therefore, there is an upper bound on the amount of processors one can meaningfully assign to stages in set A, and so is there a lower bound on the amount of processor assignable to stages in set A since $N = n_A + n_{B'}$. This phenomenon is formally stated below.

Let a_g be the last stage (the stage with the smallest B/P ratio) in A, i.e., $g = |A|-1$, and a_{g+1} be the first stage in B'.

Lemma 2: Under the conditions that all stages in set A (respectively, B) have the same table-building time (respectively, tuple-probing time), if the stages in set A and those in set B are indeed the bottlenecks in the table-building phase and tuple-probing phase, respectively, i.e. $\frac{WB_A}{n_A} > \frac{WB_i}{n_i}$, $\forall i \in B'$ and $\frac{WP_{B'}}{n_{B'}} \geq \frac{WP_i}{n_i}$, $\forall i \in A$, then n_A must satisfy the following constraints

$$N \cdot \frac{1}{1 + \frac{WB_{a_g}}{WP_{a_g}} \frac{WP_{B'}}{WB_A}} \leq n_A < N \cdot \frac{1}{1 + \frac{WB_{a_{g+1}}}{WP_{a_{g+1}}} \frac{WP_{B'}}{WB_A}} \qquad (8)$$

and vice versa.

From the above lemma, we get the following lemma to determine the optimal allocation for a given set A.

Lemma 3: The optimal value of n_A minimizes $\frac{WB_A}{n_A} + \frac{WP_{B'}}{n_{B'}}$ under the constraints in Eq. (8).

Next we consider the n_A that minimizes $f(n_A) = \frac{WB_A}{n_A} + \frac{WP_{B'}}{n-n_A}$ without any constraint. It can be derived by taking $\frac{d(f(n_A))}{dn_A} = 0$. The optimal allocation, n_A^{onc}, where the superscript *onc* indicates optimality under no constraints, is

$$n_A^{onc} = \frac{N \cdot \sqrt{WB_A}}{\sqrt{WB_A} + \sqrt{WP_{B'}}}. \qquad (9)$$

The corresponding optimal total processing time without the constraints in Eq. (8) is given by

$$TS^{onc} = \frac{1}{N} \cdot (WB_A + WP_{B'} + 2\sqrt{WB_A \cdot WP_{B'}}). \qquad (10)$$

We next define $\inf(n_A)$ and $\sup(n_A)$ as follows:

$$\inf(n_A) = N \cdot \frac{1}{1 + \frac{WB_{a_g}}{WP_{a_g}} \frac{WP_{B'}}{WB_A}}, \qquad (11)$$

$$\sup(n_A) = \left(N \cdot \frac{1}{1 + \frac{WB_{a_{g+1}}}{WP_{a_{g+1}}} \frac{WP_{B'}}{WB_A}} \right)^{-}. \qquad (12)$$

Note that x^- represents $x - \delta$ where δ is an infinitesimal quantity. Let n_A^{oc} be the n_A that minimizes TS under the constraints in Eq. (8).

Lemma 4: n_A^{oc} falls into the set of $\{\inf(n_A), \sup(n_A), n_A^{onc}\}$.

Obviously, if n_A^{onc} lies between $\inf(n_A)$ and $\sup(n_A)$, i.e. n_A^{onc} satisfies the constraints in Eq. (8), n_A^{oc} would be equal to n_A^{onc}. Otherwise, if n_A^{onc} is smaller than $\inf(n_A)$, n_A^{oc} is equal to $\inf(n_A)$. On the other hand, if n_A^{onc} is greater than $\sup(n_A)$, n_A^{oc} is $\sup(n_A)$. The procedure PA_I to derive λ_I can now be summarized as follows.

Procedure for allocation λ_I (PA_I)

1. For each stage i, $0 \leq i \leq k-1$, calculate its B/P ratio $r_i = \frac{WB_i}{WP_i}$.

2. Sort the stages in descending order of r_i to get the sequence $\vec{a}$.

3. For each j, $1 \leq j \leq k$, let $A = \{a_0, ..., a_{j-1}\}$, and find the n_A that minimizes $TS = \frac{WB_A}{n_A} + \frac{WP_{B'}}{n_{B'}}$ under the constraint

   $$\frac{N}{1 + \frac{WB_{a_g}}{WP_{a_g}} \frac{WP_{B'}}{WB_A}} \leq n_A < \frac{N}{1 + \frac{WB_{a_{g+1}}}{WP_{a_{g+1}}} \frac{WP_{B'}}{WB_A}}$$

   (based on Lemma 4). Choose the A and the associated n_A that achieves the shortest processing time.

4. From this optimal set A and n_A, obtain the number of processors n_i for each stage

   $$n_i = \begin{cases} n_A \cdot \frac{WB_i}{WB_A} & \text{if } i \in A, \\ (N - n_A) \cdot \frac{WP_i}{WP_{B'}} & \text{if } i \in B'. \end{cases} \qquad (13)$$

   This is allocation λ_I.

For example, from the profile in Table 2, we have the following 4 possible configurations: $A = \{a_0\}$, $\{a_0, a_1, a_2\}$, $\{a_0, a_1, a_2, a_3\}$, or $\{a_0, a_1, a_2, a_3, a_4\}$. For $A = \{a_0\}$ (and $B' = \{a_1, a_2, a_3, a_4\}$), we have WB_A=6 and $WP_{B'}$=15 from the rows WB_i and WP_i in Table 2, leading to TS= 2.375 by Eq. (7). Similarly, each configuration of A determines a value of TS by Eq. (7). It can then be verified that from the 4 possible configurations of A, the minimum of TS, denoted by TS_{λ_I}, is 2.362 that is achieved for $A = \{a_0, a_1, a_2\}$. From Eq. (9), we next have n_A=4.51 and $n_{B'}$= 15.49, which leads to λ_I= (4.51, 3.88, 2.25, 4.51, 4.85) from Eq. (13).

In PA_I, the combined complexity of Steps 1 and 2 is $O(k \cdot \log k)$ because of the need of sorting. The complexity of Step 3 is $O(k)$ since there are only k different ways of determining set A, and for each A, there are only 3 potential processor allocations that can be optimal according to Lemma 4. The complexity of PA_I is thus $O(k \cdot \log k)$. Since PA_I considers all the allocations satisfying the necessary conditions stated in Lemmas 1 and 4, the allocation obtained λ_I is the optimal.

Theorem 1: PA_I determines the optimal processor allocation to the pipeline stages under Constraint I.

3.2 Allocation under Constraints I and II

Next, we take into account Constraint II, which imposes a lower bound on the number of processors required for each stage. Let m_i be the number of nodes required to hold R_i, i.e., $m_i = \frac{|R_i|}{M}$. As mentioned earlier, $WB_i = C_2 \cdot |R_i|$ where C_2 is a constant. Hence, Constraint II can be expressed as $n_i \geq m_i = \frac{|R_i|}{M} = \frac{WB_i}{C_2 \cdot M}$ for every stage i.

We will show that taking Constraint II into consideration for processor allocation amounts to adding the condition $n_A \geq \frac{WB_A}{C_2 \cdot M}$ into the inequality Eq. (8) derived in Section 3.1. The procedure PA_{II} to derive λ_{II} can be summarized as follows.

Procedure for allocation λ_{II} (PA_{II})

1. For each stage i, $0 \leq i \leq k-1$, calculate its B/P ratio $r_i = \frac{WB_i}{WP_i}$.

2. Sort the stages in descending order of r_i to get the sequence $\vec{a}$.

3. For each j, $1 \leq j \leq k$, let $A = \{a_0, ..., a_{j-1}\}$, and find the n_A that minimizes $TS = \frac{WB_A}{n_A} + \frac{WP_{B'}}{n_{B'}}$ under the constraint

$$\max\left(\frac{WB_A}{C_2 \cdot M}, \frac{N}{1 + \frac{WB_{a_g}}{WP_{a_g}} \frac{WP_{B'}}{WB_A}}\right) \leq n_A < \frac{N}{1 + \frac{WB_{a_g+1}}{WP_{a_g+1}} \frac{WP_{B'}}{WB_A}}. \quad (14)$$

   Choose the A and the associated n_A that achieves the shortest processing time.

4. From this optimal set A and n_A, calculate the number of processors n_i for each stage as in Eq. (13). This is allocation λ_{II}.

Also Lemma 4 holds if $\inf(n_A)$ is defined as

$$\inf(n_A) = \max\left(\frac{WB_A}{C_2 \cdot M}, \frac{N}{1 + \frac{WB_{a_g}}{WP_{a_g}} \frac{WP_{B'}}{WB_A}}\right). \quad (15)$$

To show that the allocation λ_{II} satisfies constraint II, we need to prove the following two lemmas.

Lemma 5: Under allocation λ_{II}, every stage in A satisfies Constraint II.

The lemma follows directly from Eq. (13) as $n_A \geq \frac{WB_A}{C_2 M}$ implies $n_i = n_A \frac{WB_i}{WB_A} \geq \frac{WB_i}{C_2 M}$.

Lemma 6: Under allocation λ_{II}, every stage in B satisfies Constraint II.

For example, suppose $\langle m_i \rangle = \langle \frac{|R_i|}{M} \rangle$= (5.0, 2.5, 2.5, 5.0, 1.7) for the profile in Table 1, with $N = 20$. We then have 2 possible configurations of A to meet Constraints I and II, i.e., $A = \{a_0, a_1, a_2\}$, or $\{a_0, a_1, a_2, a_3\}$. From Eq. (7), it can be verified that these 2 configurations of A lead to TS=2.40 and TS=2.538, respectively. We hence get λ_{II}= (5.00, 3.33, 2.50, 5.00, 4.17) and $TS_{\lambda_{II}}$= 2.40.

Like PA_I, PA_{II} is also of complexity $O(k \cdot \log k)$. Following the same reasoning as for PA_I, we have the theorem below, stating the optimality of PA_{II}.

Theorem 2: PA_{II} determines the optimal processor allocation to the pipeline stages under Constraints I and II.

3.3 Allocation under three constraints

Given allocation λ_{II}, one straightforward approach to meet Constraint III is to round up the number of processors allocated to some stages, and truncate the number of processors for others. However, this simple approach, though applicable to some cases, does not in general yield the optimal allocation. It is noted that for certain configurations if the number of processors allocated to stage i by λ_{II} is n_i, the allocation to stage i by λ_{III} can be very different from either $\lfloor n_i \rfloor$ or $\lceil n_i \rceil$. It is thus necessary to develop the procedure for the optimal allocation λ_{III}.

From Constraints II and III, it follows that stage i must be allocated with at least $\lceil m_i \rceil$ processors, where $m_i = \frac{|R_i|}{M}$. If we allocate $\lceil m_i \rceil$ processors to each stage i, there will be $L = N - \sum_{\forall j} \lceil m_j \rceil$ remaining processors. To achieve λ_{III}, we must allocate these L processors to appropriate stages. Clearly, the number of additional processors allocatable to a given stage, say stage i, is between 0 and L, and the total number of processors allocated to that stage can thus range from $\lceil m_i \rceil$ to $\lceil m_i \rceil + L$.

After each stage has got $\lceil m_i \rceil$ processors to satisfy its memory constraint, we consider the issue on how to allocate the remaining L processors. The subtlety of the allocation comes from the fact that the optimal allocation of p processors cannot be obtained directly from a given optimal allocation of $p-1$ processors by greedily allocating the additional processor to the bottleneck stage. For example, consider a 3 stage pipeline with workload $\langle WB_i \rangle = (1,1,2)$ and $\langle WP_i \rangle = (10,10,2)$. Assume that each stage needs one processor to hold its hash table. Clearly, if there are 4 processors, the optimal allocation is (1, 1, 2), since allocating the extra processor to either stage 1 or stage 2 will not change the pipeline building and probing times. However, if there are 5 processors, the optimal allocation is (2, 2, 1), meaning that the optimal allocation of 5 processors cannot simply be obtained based on the optimal allocation of 4 processors.

We therefore need to devise an efficient mechanism to determine the optimal allocation. Let TB_i^x and TP_i^x be, respectively, the table-building and tuple-probing times of stage i, where x is the total number of processors allocated to this stage. For each allocation of x processors to stage i, we then have a corresponding *allocation descriptor*, (i, x, TB_i^x, TP_i^x). We shall maintain certain data structures on the allocation descriptors to facilitate the allocation of the L additional processors.

First, we order the possible allocation descriptors into two *allocation queues*, QB and QP, each of which consists of all $k(L+1)$ descriptors. Elements in QB and QP are sorted in decreasing order of TB_i^x and TP_i^x, respectively. Note that each allocation descriptor will appear in both QB and QP, but the positions in which it appears in QB and QP may be different. Marking an allocation descriptor can be thought of as allocating one additional processor to the stage specified in that descriptor. Allocating L additional processors to the pipeline stages then corresponds to *marking* L distinct allocation descriptors. Our method of marking allocation descriptors is to follow the descriptor orders in either QB or QP. Thus a legitimate marking of L descriptors would be a marking on the first j descriptors of QB and the first k descriptors of QP, if there are exactly L distinct descriptors among these $(j+k)$ descriptors. (Note one descriptor may fall into both the first j elements in QB and the first k elements in QP.) In each stage, the marking can only occur in increasing order of x values for that stage. The allocation descriptors for a stage i start with $(i, \lceil m_i \rceil, TB_i^{\lceil m_i \rceil}, TP_i^{\lceil m_i \rceil})$. When (i, x, TB_i^x, TP_i^x) is marked, it means that at least $x+1-\lceil m_i \rceil$ descriptors of stage i must have been marked, and at least $x+1$ processors must have been allocated to stage i. If the last allocation descriptor marked for stage i is labeled with x processors, the total number processors assigned to that stage is $x+1$, and the corresponding stage building and probing times will be TB_i^{x+1} and TP_i^{x+1}, respectively. Consequently, we have the following lemma.

Lemma 7: The table-building time of the first unmarked element in QB is the table-building time of the pipeline, and the tuple-probing time of the first unmarked element in QP is the tuple-probing time of the pipeline.

Denote the i-th element in QB as b_i, and the associated building time as $TB_{<b_i>}$. p_j in QP and $TP_{<p_j>}$ are defined similarly. The procedure PA_{III} for λ_{III} can be summarized as follows.

Procedure for allocation λ_{III} (PA_{III})

1. For each stage i, $0 \le i \le k-1$, first allocate $\lceil m_i \rceil$ processors to that stage.
2. For each stage i, $0 \le i \le k-1$, determine the $L+1$ allocation descriptors, (i, x, TB_i^x, TP_i^x), $\lceil m_j \rceil \le x \le \lceil m_j \rceil + L$, for the possible allocations of additional processors, where $L = N - \sum_{\forall j} \lceil m_j \rceil$.
3. Construct QB and QP (each with $k(L+1)$ descriptors), where the allocation descriptors are sorted in descending order of TB_i^x and TP_i^x, respectively.
4. For each j, $0 \le j \le L$, determine $k(j) (\ge (L-j))$ such that the number of distinct elements in the set $D_j = \{b_1, b_2, \ldots, b_j\} \cup \{p_1, p_2, \ldots, p_{k(j)}\}$ is L. (This can be achieved by marking the first j elements in QB first and then the first $(L-j)$ unmarked elements in QP. Thus D_j is the set of marked allocation descriptors and corresponds to an allocation of L processors.) For each j, calculate the time $TB_{<b_{j+1}>} + TP_{<p_{k(j)+1}>}$, which is the processing time of the pipeline under the corresponding allocation.
5. Choose the j, $0 \le j \le L$, with the shortest processing time. The corresponding allocation is λ_{III}.

Consider the example in Table 2. Since $\langle m_i \rangle = \langle \frac{|R_i|}{M} \rangle$= (5.0, 2.5, 2.5, 5.0, 1,7) from λ_{II} in Section 3.2, we get $\langle \lceil m_i \rceil \rangle$ = (5, 3, 3, 5, 2), meaning that there are $L = 2$ processors left that will be added to the appropriate stages by λ_{III}. QB and QP can then be

determined, each of which consists of $(2+1)*5 = 15$ elements in the form of (i, x, TB_i^x, TP_i^x). We then have, $b_1 = (0,5,1.2,0.8)$, $b_2 = (3,5,1.2,0.6)$, $b_3 = (4,2,1.0,2.5)$, $b_4 = (1,3,1.0,1.33)$, etc., for QB, and $p_1 = (4,2,1.0,2.5)$, $p_2 = (4,3,0.66,1.66)$, $p_3 = (1,3,1.0,1.33)$, $p_4 = (4,4,0.5,1.25)$, etc., for QP.
From Step 4 in PA_{III}, we obtain
for $(j,k) = (0,2), TS = TB_{<b_1>} + TP_{<p_3>} = 1.2 + 1.33 = 2.53$,
for $(j,k) = (2,0), TS = TB_{<b_3>} + TP_{<p_1>} = 1.0+2.5 = 3.5$,
for $(j,k) = (1,1), TS = TB_{<b_2>} + TP_{<p_2>} = 1.2 + 1.66 = 2.86$, etc.
It follows that the 2 additional processors should be added to stage 4 to achieve an optimal allocation $\lambda_{III} = (5,3,3,5,4)$ with $TS_{\lambda_{III}} = 2.53$.

It can be verified that the complexity of PA_{III} is $O(Nk \cdot \log(Nk))$ because of the sorting in QB and QP. Also, from the way QB and QP are constructed, it follows that the allocation achieved by λ_{III} is the one with shortest processing time among all possible allocations.

Theorem 3: Considering all three constraints, PA_{III} gives the optimal processor allocation to the stages in a pipeline.

4 Simulation

We have formulated the processor allocation as a deterministic optimization problem and developed optimal solution procedures. In the simulation, each individual tuple actually runs through the stages in a pipeline of hash joins, so that the burst effects and the actions for pipeline fill-up and depletion are captured. The simulation verifies that our formulation of the two-phase mini-max optimization problem well approximates the original pipelined hash join problem and provides it with an effective solution procedure.

4.1 The Simulation Model

To focus on the effect of processor allocation on pipelined hash joins, we simulate pipeline segments of hash joins, with and without using the optimal processor allocation scheme. The simulator takes hardware parameters, a pipeline segment of hash joins and a processor allocation for the pipeline as inputs. It outputs the query processing time of the pipeline. The number of stages in the pipeline is predetermined in the simulation. The cardinalities of the relations in pipeline stages are randomly chosen from fixed ranges. In order to simulate the behavior of each tuple accurately, we scaled down the average relation size and reduced the memory size of each processor accordingly, so that the ratio of average relation size to memory size is nonetheless realistic.

parameters	*mu*-sec
t_{read}	14
$t_{receive}$	20
t_{hash}	12
t_{comp}	12
t_{build}	8
t_{part}	12
t_{send}	20
t_{insert}	2
t_{write}	14

Table 3: Architectural parameters employed in simulation.

The hash table for each pipeline stage is built in the table-building phase by partitioning its inner relation into subtables, one for each processor allocated to the stage. In the probing phase, each incoming tuple is routed to one of these subtables at random. A random number generator, coded based on the join selectivities, is then used to generate the resulting tuples. The time spent on various actions such as partitioning, hashing, matching, or building resulting tuples, are highly dependent on the architecture, OS (or equivalent system software) and the actual implementation. However, to better reflect reality, we have determined the relative CPU times for the various actions by actually taking measurements for sample code on a SUN sparc station. These parameters are reported in Table 3.

4.2 Simulation Results

Two processor allocation schemes are used for each query in the simulation: the optimal processor allocation algorithm PA_{III} and a heuristic HT. Heuristic HT allocates processors to the stages in proportion to their hash table sizes. The numbers of processors so allocated are real numbers, which are then converted to integers subject to the constraint that the number of processors allocated to each stage is large enough to hold its hash table.

The pipelines employed in the simulation are chosen to be of five stages (i.e. six relations). Results from four experiments are presented here, although more experiments on sensitivity analysis have been conducted and indicated similar trends. In each experiment, three hardware configurations, with 16, 32 and 64 processors, respectively, are used. Sixty pipelines are simulated for each hardware configuration. Each combination of pipeline and hardware configuration is simulated three times to capture the randomness on the tuple generation

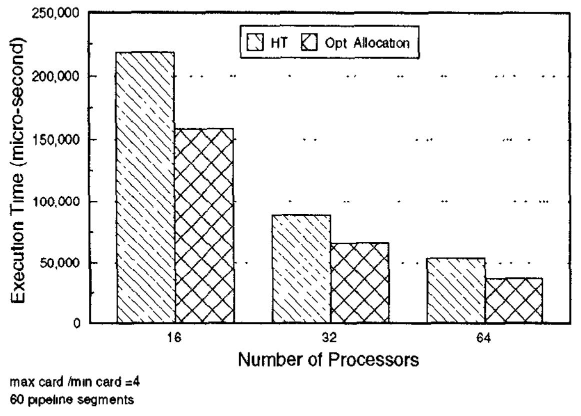

Figure 3: Comparison of the optimal scheme and heuristic HT when max card./min card.=4.

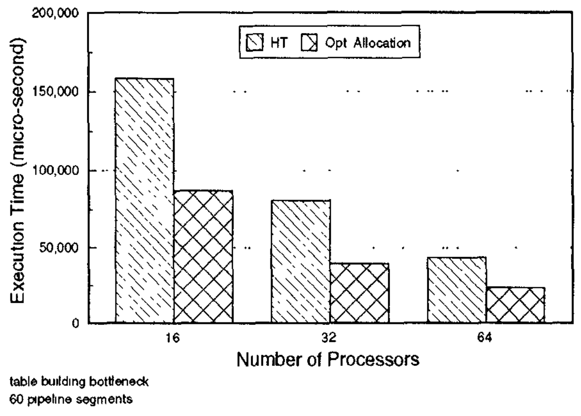

Figure 5: Comparison of the optimal scheme and heuristic HT for table-building phase bottleneck.

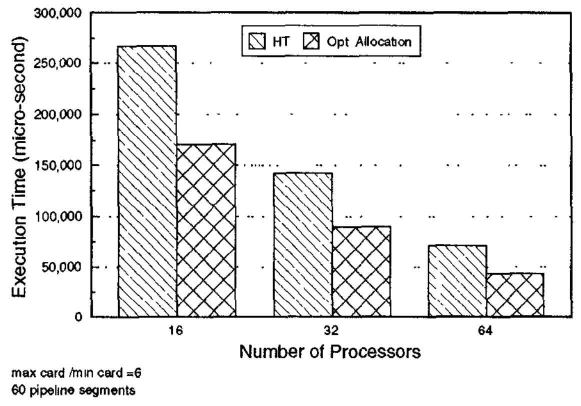

Figure 4: Comparison of the optimal scheme and heuristic HT when max card./min card.=6.

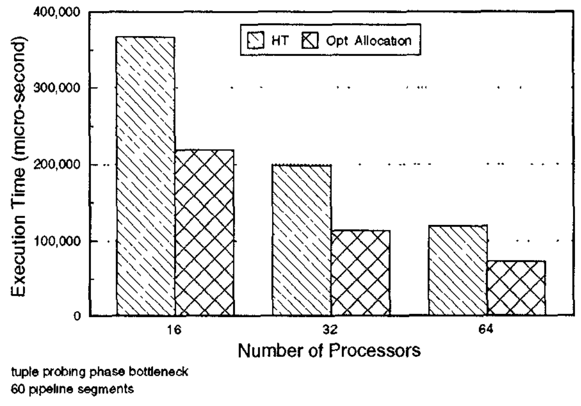

Figure 6: Comparison of the optimal scheme and heuristic HT for tuple-probing phase bottleneck.

and matching in a join during the simulation.

In the first experiment, the cardinalities of all inner relations and intermediate relations are randomly selected from a range between 800 and 3200, with the ratio of maximal to minimal cardinality equal to 4. The second experiment deals with relations with a larger variance, and has relation cardinalities generated from 600 to 3600, with the ratio of maximal to minimal cardinality equal to 6. In the third experiment, we randomly set a stage to be the bottleneck of the table-building phase by assigning that stage with an inner relation ten times as the average size of other inner relations. The ratio of maximal to minimal cardinality of the non-bottleneck stages is set to 4. In the fourth experiment, we set one stage to be the bottleneck of the tuple-probing phase by adjusting its tuple-probing phase workload to be ten times as the average of other stages. (This is achieved by making the intermediate relation from the previous stage ten times as large as the average.) Without loss of generality, we set the bottleneck stage to be the last stage in a pipeline. The ratio of maximal to minimal cardinality of the non-bottleneck stages is also set to 4.

The performance data for the first two experiments are shown in Figures 3 and 4. In can be seen that in the first experiment, the optimal processor allocation scheme shows 28% to 31% improvement over the simple heuristic. When the ratio of the maximal to minimal cardinality increase to 6 in the second experiment, the performance improvement of the proposed scheme increases to the range between 36% and 38%, meaning that when the cardinalities of relations and join attributes become more widely varied, PA_{III} offers even better performance improvement. As shown in Figures 5 and 6, it is observed that in the case of a tuple-probing phase bottleneck, the optimal processor allocation provides 39% to 43% improvement in execution time. In the case of a table-building phase bottleneck, the optimal processor allocation shows 45% to 51% improvement, an even better result. It can be seen that the more skewed the input is, the more improvement can be achieved by a better processor allocation scheme. As a matter of fact, skewed inputs are usually those creating performance bottleneck, and the very ones we would like to tackle for better overall performance. In general, it is observed from simulation results that the proposed scheme performs very consistently, and can lead to significant performance improvement over simple heuristics.

5 Conclusions

This paper presents a method for achieving optimal processor allocation for pipelined hash joins. We formulated the processor allocation problem as a two-phase mini-max optimization problem and developed both exact and approximate solution procedures. We also demonstrated through simulation that our problem formulation, though not explicitly incorporating all dynamic effects of pipelining, can lead to solutions with substantial performance improvement over previous approaches. The results are useful for dealing with both right-deep trees and segmented right-deep trees.

References

[1] H. Boral, W. Alexander, et al. Prototyping Bubba, A Highly Parallel Database System. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):4–24, March 1990.

[2] M.-S. Chen, M.-L. Lo, P. S. Yu, and Y. C. Young. Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins. *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 15–26, August 1992.

[3] M.-S. Chen, P. S. Yu, and K.-L. Wu. Scheduling and Processor Allocation for Parallel Execution of Multi-Join Queries. *Proceedings of the 8th International Conference on Data Engineering*, pages 58–67, February 1992.

[4] D. J. DeWitt and R. Gerber. Multiprocessor Hash-Based Join Algorithms. *Proceedings of the 11th International Conference on Very Large Data Bases*, pages 151–162, August 1985.

[5] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.I. Hsiao, and R. Rasmussen. The Gamma Database Machine Project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, March 1990.

[6] D. J. DeWitt and J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. *Comm. of ACM*, 35(6):85–98, June 1992.

[7] W. Hong. Exploiting Inter-Operator Parallelism in XPRS. *Proceedings of ACM SIGMOD*, pages 19–28, June 1992.

[8] K. Hua, Y.-L. Lo, and H. C. Young. Including the Load Balancing Issue in the Optimization of Multi-Way Join Queries for Shared-Nothing Database Computers. *Proceedings of the 2nd Conference on Parallel and Distributed Information Systems*, pages 74–83, January 1993.

[9] T. Ibaraki and N. Katoh. *Resource Allocation Problems: Algorithmic Approaches.* The MIT Press, Cambridge, Massachusetts, 1988.

[10] M. Kitsuregawa, H. Tanaka, and T. Moto-Oka. Architecture and Performance of Relational Algebra Machine GRACE. *Proceedings of the International Conference on Parallel Processing*, pages 241–250, August 1984.

[11] M.-L. Lo, M.-S. Chen, C. V. Ravishankar, and P. S. Yu. Optimal Processor Allocation for Pipelined Hash Joins. *IBM Research Report, RC 18303*, September 1992.

[12] R. A. Lorie, J.-J. Daudenarde, J. W. Stamos, and H. C. Young. Exploiting Database Parallelism In A Message-Passing Multiprocessor. *IBM Journal of Research and Development*, 35(5/6):681–695, September/November 1991.

[13] H. Lu, K. L. Tan, and M.-C. Shan. Hash-Based Join Algorithms for Multiprocessor Computers with Shared Memory. *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 198–209, August 1990.

[14] P. Mishra and M. H. Eich. Join Processing in Relational Databases. *ACM Computing Surveys*, 24(1):63–113, March 1992.

[15] N. Roussopoulos and H. Kang. A Pipeline N-Way Join Algorithm Based on the 2-Way Semijoin Program. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):461–473, December 1991.

[16] D. Schneider and D. J. DeWitt. Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines. *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 469–480, August 1990.

[17] M. Stonebraker, R. Katz, D. Patterson, and J. Ousterhout. The Design of XPRS. *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 318–330, 1988.

[18] A. Wilschut and P. Apers. Dataflow Query Execution in Parallel Main-Memory Environment. *Proceedings of the 1st Conference on Parallel and Distributed Information Systems*, pages 68–77, December 1991.

[19] P. S. Yu, M.-S. Chen, H. Heiss, and S. H. Lee. On Workload Characterization of Relational Database Environments. *IEEE Transactions on Software Engineering*, 18(4):347–355, April 1992.