Title
PROCESSOR SCHEDULING FOR MULTIPROCESSOR JOINS

Authors
Murphy, M.C.
Rotem, D.

Publication Date
1989-02-01

LBL-25470 c.2
Reprint

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

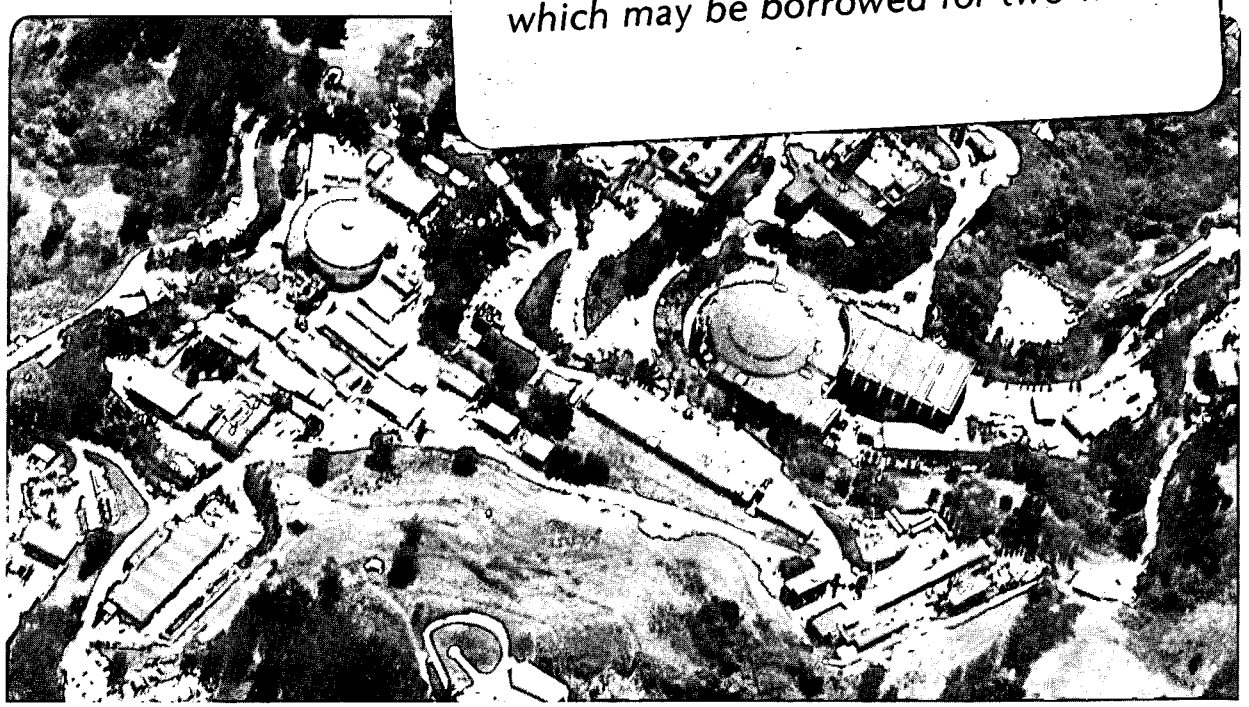
Information and Computing Sciences Division

Presented at the 5th International Conference on Data Engineering,
Los Angeles, CA, February 6–10, 1989

Processor Scheduling for Multiprocessor Joins

M.C. Murphy and D. Rotem

February 1989



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

DISCLAIMER

COMPUTER SOCIETY PRESS REPRINT

PROCESSOR SCHEDULING FOR MULTIPROCESSOR JOINS

Marguerite C. Murphy
Doron Rotem

Reprinted from PROCEEDINGS OF THE FIFTH INTERNATIONAL
CONFERENCE ON DATA ENGINEERING
Los Angeles, CA, February 6-10, 1989

The Computer Society of the IEEE
1730 Massachusetts Avenue NW
Washington, DC 20036-1903

Washington ◦ Los Alamitos ◦ Brussels

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE COMPUTER SOCIETY PRESS

PROCESSOR SCHEDULING FOR MULTIPROCESSOR JOINS

Marguerite C. Murphy† and Doron Rotem

Computer Science Research Department
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

ABSTRACT

This paper presents a family of practical algorithms to schedule join execution in a shared memory multiprocessor environment. The algorithms are based upon page connectivity graphs and determine when to read each data page into memory and how to schedule page joins on the available processors. Our goal is to overlap page reads with parallel join execution in such a way that both the number of processors and total duration of join processing time are minimized. We define optimal time as the minimum time required on an unbounded number of processors and derive upper and lower bounds on the number of processors required to complete join execution in optimal time. We then describe a general strategy for generating read schedules that we conjecture can be processed in minimal time (over all read schedules on any number of processors) and a family of practical algorithms utilizing an arbitrary number of lookahead steps to approximate this general strategy.

1. Introduction

Joins are among the most computationally expensive relational operators to evaluate. Traditional approaches to reducing the execution overhead include access method support for efficient random and key sequential record retrieval as well as query optimization over a set of join processing algorithms. A relatively new approach is to utilize multiprocessor parallelism to (hopefully) reduce the overall duration of query execution, possibly at a cost of increased total computational resource requirements [Bitt83a, Murp85a]. Since the degree of parallelism is potentially unlimited, the system performance rapidly becomes limited by the i/o bandwidth--the rate at which data can be transferred to and from the stable secondary storage devices into main memory for join computation. In this paper we investigate the problem of optimizing parallel join computations within the limits imposed by the i/o subsystem bandwidth.

We consider a shared memory multiprocessor architecture in which all areas of memory are accessible by all processors and any number of processors can read from the same data buffer simultaneously. Note that this does not require simultaneous access to a single data element within that buffer. We assume that the data path from secondary storage to main memory has a concurrency of one and data cannot be accessed until transfer is complete. Extension of our results to an environment with multiple independent data paths is straightforward. The granularity of i/o scheduling is the individual page read and the granularity of processor scheduling the page join. The overall join computation is viewed as a collection of page reads and page joins (with dependencies introduced by the requirement that data pages be read into memory before participating in any page joins). We neglect scheduling overhead and for simplicity initially assume that the duration of a join between two data pages on one processor is equal to the time required to transfer one page of data from secondary storage into main memory. This assumption is quite reasonable in a hardware environment with 20ms disk transfer times and 10 microsecond instruction execution time, where roughly 2000 instructions can be executed during one disk i/o transfer. Although this approximation justifies our assumption, we are currently developing a more detailed stochastic performance model to evaluate the algorithms described below.

As in more traditional approaches, the data on secondary storage is organized into fixed size pages, each of which is individually addressible and can be read into memory independently of the others. We assume an indexing scheme is used which permits generation of a *page connectivity graph*-- a bipartite graph with one node corresponding to each page of each relation and one edge connecting each pair of pages which contain at least one matching join attribute value. This model was introduced by [Merr81a], formalized by [Pram85a] and then later adopted by [Goya88a] within the context of their "Bc tree" access method. A Bc tree is a secondary B+-

† Also with Computer Science Department, San Francisco State University, San Francisco, CA. 94132

This research was supported by the Applied Mathematics Sciences Research Program of the Office of Energy Research, U.S. Department of Energy under contract DE-AC03-76SF00098.

tree index over a join attribute containing entries for tuples in all of the join relations. Given a Bc tree for a join, generating the corresponding page connectivity graph is a straightforward computation involving a scan of the leaf nodes in the Bc tree and elimination of duplicate edges from the final graph. Join graphs can also be generated easily from join indices [Vald87a] as well as from the intermediate results generated during an index-join processing algorithm [Blas76a]

Given a page connectivity graph describing the page joins to be performed, there are several related problems to be addressed: determining the number of processors to allocate to the join computation, scheduling the order in which data pages are read into memory from disk, and scheduling the processing of joins on available processors. Our approach is to first analytically bound the number of processors required to complete execution in optimal time. We then propose a general scheduling strategy (which may not be realizable in a practical algorithm) which will produce a read schedule that we conjecture can be processed in minimal time. Finally we propose a family of practical read scheduling algorithms which come arbitrarily close to generating such a read schedule. Once we have a specific read schedule, scheduling joins on any number of processors in order to minimize total processing time is trivial-- simply assign as many processors as possible to page joins between pages which are memory resident.

We assume that sufficient memory is available to buffer pages until joining pages and processors are available, implying that a page never needs to be read into memory more than once. Note that the goal of our algorithms is to overlap page reads with parallel join processing in such a way that both the number of processors and duration of join processing are minimized. [Pram85a] and [Merr81a] address the quite different problem of scheduling page fetches to minimize the buffer requirements in a uniprocessor environment.

An example of the problem considered in this paper follows. Figure 1 shows a simple page connectivity graph representing two relations, with two and five data pages respectively, and a total of nine page joins to be performed. Note that pages one and seven contain no join attribute values in common and all other page pairs contain at least one common join attribute value.

The following figures illustrate two of the many possible schedules for reading and executing the example join. Pages (nodes) are read into memory one by one in a sequence determined by the read schedule. As soon as both pages participating in a page join are memory resident, that join can be scheduled on a processor for execution. At each time step, processors are scheduled by assigning as many processors as possible to enabled page joins. In the first schedule, no page joins can be processed until the seventh time step, resulting in an overall join execution time of eleven time steps. The second schedule completes processing in only eight time steps, a savings of nearly 25% in total execution time. This is accomplished by scheduling the reads in such a way that page joins can be processed as soon as possible in the execution sequence, resulting in higher overall processor utilization and lower total execution time.

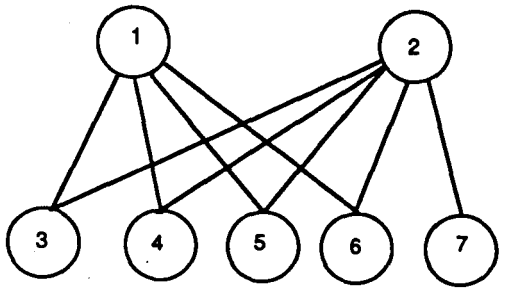


Figure 1. Example Page Connectivity Graph

Time Step	Read Schedule	Processing Schedule	
		P1	P2
1	r7		
2	r3		
3	r4		
4	r5		
5	r6		
6	r1		
7	r2	1,3	1,4
8		1,5	1,6
9		2,3	2,4
10		2,5	2,6
11		2,7	

Figure 2: Non-optimal Read and Processing Schedules

This paper is organized as follows. The following two sections introduce our formal notation and derive bounds on the number of processors required to complete join execution in optimal time. The fourth section describes minimal read strategies by first giving the underlying intuition, then a more formal statement of their derivation. The next two sections introduce a family of practical algorithms which approximate minimal read strategies and examples of their execution. The last section summarizes our conclusions.

2. Preliminaries

In this section we introduce some standard definitions and notations from graph theory which are used in the remainder of the paper. A bipartite graph $B(V,E)$ consists of a vertex set $V=V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$ and edge set $E \subseteq V_1 \times V_2$, i.e., edges are pairs of the form (x,y) where $x \in V_1$ and $y \in V_2$. A complete bipartite graph is a bipartite graph with the maximum number of edges, i.e., $|E| = |V_1| * |V_2|$. For a vertex in V, we denote by $d(v)$ its degree which is equal to the number of edges incident on it. Given a vertex v, its neighborhood is the

Time Step	Read Schedule	Processing Schedule	
		P1	P2
1	r2		
2	r6		
3	r1	2,6	
4	r5	1,6	
5	r4	1,5	2,5
6	r3	1,4	2,4
7	r7	1,3	2,3
8		2,7	

Figure 3. Optimal Read and Processing Schedules

set of all vertices adjacent to it. This is denoted by $\Gamma(v)$. This definition can be extended to a set of vertices R. The neighborhood of R, denoted by $\Gamma(R)$, is the union of the neighborhoods of all vertices in R, i.e, it is the set of vertices adjacent to at least one member of R. A connected component C of B is a subgraph of B such that a path exists between every pair of vertices of C and no path exists between a vertex in $B-C$ and a vertex in C. We can find all the connected components of a graph B in $O(|V|+|E|)$ time using a depth-first search algorithm [Aho74a].

3. Bounds on Processors and Execution Time

In this section we assume that the join graph $B(V,E)$ is a bipartite graph with a single connected component. The results can be easily generalized to graphs with more than one connected component. The bounds we derive here are based on knowledge of some simple parameters of the graph such as the cardinalities of the edge and vertex set or the size of the largest degree in the graph. Of course it is possible to derive tighter bounds if we have more information about the graph. Bounds are important since they give us some ideal measures against which we can compare our algorithms. We consider first bounds on the number of processors required to complete the join in optimal time. We then derive bounds on the time required to complete the join with a given fixed number of processors.

We denote by $T_{opt}(B)$ the optimal time to complete a join represented by the graph B, i.e., the minimum number of time steps with an unbounded number of processors.

Lemma 1 :

$$T_{opt}(B) = |V| + 1.$$

Proof : Let S be an optimal time schedule for the graph B. At each time step we can read exactly one vertex of the graph. Let us denote by v_x the last vertex read by the schedule S. The processing of all edges incident on v_x can be completed only after v_x has been read. Since the graph is connected, there will be at least one edge incident on v_x and therefore at least one additional time step is needed to complete the join after all the vertices have been read. □

In the following theorems we compute an upper and a lower bound on, $P_{opt}(B)$, the number of processors required to complete the join in optimal time $T_{opt}(B)$. We derive a bound which assumes that all we know about the graph is $|V|$ and $|E|$.

Lemma 2 : The maximum number of page joins a schedule can complete during the first $i+1$ steps (with unbounded number of processors) is

$$\begin{cases} \frac{i^2}{4} & \text{for } i \text{ even} \\ \frac{(i-1)(i+1)}{4} & \text{for } i \text{ odd} \end{cases}$$

Proof : Let us denote by $B_i(S)$ the subgraph of B read by a schedule S during its first i steps. This subgraph consists of the set of i vertices read by the schedule and all edges of B with both endpoints in this set. Let V_1^i and V_2^i be the vertices of $B_i(S)$ which belong to V_1 and V_2 respectively. At step $i+1$, the schedule S can perform all joins such that their corresponding edges are in $B_i(S)$. It is easy to see that for any schedule S, the number of edges in $B_i(S)$ is maximized when this graph is a complete bipartite graph with $|V_1^i| = |V_2^i| = \frac{i}{2}$ when i is even and $|V_1^i| = \frac{i-1}{2}$ and $|V_2^i| = \frac{i+1}{2}$ for odd i. The expression in the statement of the lemma represents the number of edges in the graph corresponding to each of these cases. □

Theorem 1 : The number of processors required to complete the join in optimal time on the graph $B(V,E)$ satisfies

$$P_{opt}(B) \geq \frac{|V| - \sqrt{(|V|^2 - 4|E|)}}{2}$$

Proof : The strategy of the proof is to show that there exists a time step in which at least the above number of page joins must be completed in parallel. Let us assume that a schedule S completes all page joins of B in optimal time with a minimum number of processors. Let $MAX(S)$ denote the maximum number of page joins performed in a single step by the schedule S. We denote by $c(i)$ the number of page joins performed by S during its first $i+1$ steps. Then in order for S to complete the join in optimal time, it has to perform the additional $|E| - c(i)$ joins during the remaining $|V| - i$ steps. Therefore there must be at least one step among these $|V| - i$ steps in which the number of page joins performed is at least

$$\left\lceil \frac{|E| - c(i)}{|V| - i} \right\rceil$$

which is the average number of page joins per step during this phase.

By Lemma 2, for $1 \leq i \leq |V|$

$$c(i) \leq \frac{i^2}{4}$$

>From which we conclude that for $1 \leq i \leq |V|$

$$MAX(S) \geq \frac{|E| - \frac{i^2}{4}}{|V| - i}$$

In order to make the bound as tight as possible we will find the value of i for which the right hand side achieves its maximum. We use elementary calculus to find that the right hand side achieves its maximum when the value of i is the closest integer to

$$|V| - \sqrt{(|V|^2 - 4|E|)}$$

By substituting this value of i into the right hand side we obtain

$$MAX(S) \geq \frac{|V| - \sqrt{(|V|^2 - 4|E|)}}{2}$$

as claimed. □

In order to demonstrate the significance of this bound let us consider the family of bipartite graphs in which the number of edges , $|E|$, is $\alpha |V|^2$ for $0 < \alpha \leq 0.25$. In this case, the bound on the number of processors is of the following form

$$\frac{|V|(1-\sqrt{1-4\alpha})}{2}$$

For example, in a graph with $|V| = 18$ and $|E| = 72$ ($\alpha = \frac{2}{9}$) the minimum number of processors required to finish in optimal time is 6.

In the next theorem we exploit more information about the graph to derive an upper bound on $P_{opt}(B)$. We assume that $|E|$, $|V_1|$, $|V_2|$ and the maximum degree of a vertex in V_2 are given. Without loss of generality let us assume that $|V_1| \leq |V_2|$.

Theorem 2 : Let $d_{\max}$ be the maximum degree of a vertex in V_2 . Then $P_{opt}(B) \leq A$ where

$$A = \left\lceil \frac{|E| - d_{\max}}{|V_2| - 1} \right\rceil$$

Proof : We exhibit a simple schedule S' which completes the join in optimal time using no more than A processors. The schedule S' is characterized by the following rules: Let us call a step in which a vertex of V_1 is read a type I step and all other steps are called type II steps.

(a) The vertices of V_2 are sorted in non-increasing order of their degrees and relabelled $v_1, v_2, .. v_l$ ($l = |V_2|$) such that v_1 is the vertex with the maximum degree, $d_{\max}$, and v_l has the minimum degree.
The schedule performs its reads in $l+1$ rounds each consisting of a type II step followed by zero or more type I steps in the following way:
(b) In the first time step of round i ($i < l+1$) v_i is read, this is followed by reading in all the vertices of V_1 connected to it which have not been read yet. Round $l+1$ consists of the final join.
(c) All page joins are scheduled to take place as early as possible, i.e., as soon as the two endpoints of an edge have been read in and there is a free processor to perform the join.

By using just a single processor we can guarantee that after v_2 (the second vertex of V_2) is read , all the edges incident on v_1 have been processed. We are now left with $|E| - d_{\max}$ page joins which must be completed during the remaining $|V_2 - 1|$ rounds. We will now show that A processors are sufficient to complete the join in optimal time. The number of new potential joins introduced at the end of round i is at most equal to, $d(v_i)$, the degree of v_i. For simplicity, from now on we will assume that page joins are performed only on the first step of each round, i.e., on the first step of round $i+1$ we will attempt to perform all page joins involving v_i (this is possible as all endpoints of edges involved have been read). However there are only A processors available and therefore up to $d(v_i) - A$ potential joins may have to be deferred to some future round.

Note that the value of A is the ceiling of the average degree of vertices in the set $V_2 - v_1$. so that there must be a first index j such that $d(v_j) \leq A$. At the beginning of round j, the total number of page joins deferred from all previous rounds is at most

$$\sum_{i=2}^{i=j-1} (d(v_i) - A)$$

On the other hand, by the decreasing order of degrees, there will be a total of at least

$$\sum_{i=j}^{l} (A - d(v_i))$$

available processors to complete these deferred joins during rounds $j+1, j+2, ..., l+1$. Since as we noted before A is equal or larger than the average degree in the set $V_2 - v_1$ we have

$$A \geq \frac{1}{(l-1)} \sum_{i=2}^{i=l} d(v_i)$$

from which it follows that

$$\sum_{i=j}^{l} (A - d(v_i)) \geq \sum_{i=2}^{i=j-1} (d(v_i) - A)$$

and the number of available processors is sufficient to complete all the page joins. □

It is interesting to note that the average processor utilization of the schedule S' is greater than 50% of the maximum possible. We prove this by showing that the number of processors used by the schedule S' is within a factor of two from optimal. Since we are only interested here in an asymptotic result, we will ignore the ceiling functions in the following computation.

Corollary 1 : The number of processors A used by schedule S satisfies

$$A \leq 2 * P_{opt}(B)$$

Proof : By the argument in Theorem 1 and setting i to the value 2, we get $P_{opt}(B) \geq \frac{|E|-1}{|V|-2}$. The ratio of A to the optimal number of processors is

$$\frac{A}{P_{opt}(B)} \leq \frac{A(|V|-2)}{|E|-1}$$

By substituting the value of A and observing that since $|V_2| \geq |V_1|$ we have $\frac{|V|-2}{|V_2|-1} \leq 2$ the result follows. □

We now turn our attention to bounds on the number of time steps with a given number of processors p. Clearly this question is only of interest when $p < A$ otherwise based on our previous results we can finish in optimal time. We will also assume the number of edges in the graph satisfies $|E| \geq p^2$.

Theorem 3 : The time to complete the join with p processors is at least

$$\max\left\{ |V|+1, 2p+1+\left\lceil \frac{|E|-p^2}{p} \right\rceil \right\}$$

Proof : The maximum number of joins during the first $2p+1$ steps is p^2. The remaining $|E|-p^2$ joins can be done at the maximum rate of p per step thus requiring

$$\left\lceil \frac{|E|-p^2}{p} \right\rceil$$

additional steps. The result is proved by taking into account the constraint derived in Lemma 1 which shows that we cannot finish in less than $|V|+1$ steps. □

4. Minimal Read Strategies

In this section we assume that the number of processors is fixed at some arbitrary value which need not be between the bounds described above. We describe a strategy for generating a read schedule which we conjecture can be processed in the minimum amount of time (over all read schedules) on any number of processors. An informal description of the intuition behind this strategy is presented in the next subsection, followed by a more formal statement in the following subsection.

4.1. Intuition

We first introduce our terminology. The *actual work value* of a node at a given time is the number of joins that the node can participate in, that is, the number of unprocessed join edges to other nodes which are memory resident. The *potential work value* of a node is the total number of joins that the node participates in (i.e. the degree of the node). The *system work value,* W, is the sum of the actual work values of memory resident nodes. Under our assumptions this corresponds to a remaining processing time of W/k units on k processors.

The overall goal in read scheduling is to initially increase the system work, W, as quickly as possible and then to maintain it at the highest possible level throughout the remaining processing time, i.e. to maximize the average rate of increase at each step of the read schedule. This will minimize the time processors spend idle at the beginning of join execution which will in turn result in maximum processor utilization and minimum overall execution time. Read schedules which have this property are called minimal.

Read scheduling proceeds in three phases. The first phase partitions the graph into connected components. The second phase determines the order in which pages within each connected component should be read into memory and the third phase ranks the components. Components are ranked by how long they will take to execute, with longer durations ranking more highly. The intuition behind this is that components with longer durations require a relatively high number of join steps after all of their vertices have been read in, and these steps can be used to start reading in the vertices of the following component.

Nodes within each component are scheduled first by actual work values and second by potential work values. That is, at each step, the next node in the minimal read schedule is the one with the highest actual work value. If two or more nodes have equal actual work values, the one with the highest potential work is read next. If two or more nodes have equal potential and actual work values, the node which will maximize the rate at which W increases over the remaining processing time is chosen. Unfortunately, decisions of the latter type require knowledge of the future behavior of the algorithm and are not realizable in practical algorithms.

The next section describes our strategy more formally and the following section presents a family of practial algorithms which come arbitrarily close to selecting a minimal read schedule as defined here. Note that with a minimal read schedule the number of processors required to complete in minimum time is at most equal to the upper bound derived in the previous section, and with this number of processors the processing time is optimal. If the read schedule is not minimal, a greater number of processors may be required to complete in optimal time.

4.2. Formal Statement

The algorithm described here takes a bipartite graph graph $B(V,E)$ as its input and produces a schedule S for p processors from it. For expository reasons we divide the process of producing a schedule into four stages. A schedule consists of a read schedule and a join schedule. The first three stages are directed towards producing an efficient read schedule. In the final step we perform the join schedule.

In stage 1, we decompose the graph into its connected components. In stage 2, for each component C_i we derive independently an ordering of its page reads.

In stage 3, we compute for each component C_i with m edges and n vertices the function $f(C_i)=m/p-n$. We then concatenate the read schedules of all the components in decreasing order of $f(C_i)$. The intuition behind this step is that components with a high value of $f(C_i)$ require a relatively high number of join steps after all their vertices have been read in. We can use these steps to start reading the vertices of the following component. At this point we have a read schedule for the whole graph. It now remains to schedule the page joins at each time step. This is done in stage 4 in a simple manner. We maintain an actual work queue as follows. Initially the queue is empty. Let us assume that vertex v is scheduled to be read at time step i. We remove from the queue the first p edges and schedule their corresponding page joins on processors 1,2,..p respectively. If the queue contains less than p edges then some (or all) of the processors remain idle at this step. We then insert into the end of the queue all the edges incident on v which have their other endpoint read prior to step i. This represents the new actual work introduced by v.

As stages 1,3 and 4 are relatively simple we only give here a more formal description of stage 2 which is presented as Algorithm 1.

Algorithm 1

Input: Component C with vertex set V and edge set E.
Output: A read schedule S for C. We build S by concatenating a new vertex to it each time the loop is processed. The variable $A(v)$ keeps track of the actual work for each node.

Step 1: [Initialize] **For** all $v \in V$, set $A(v):=0$ and $S:=\emptyset$.
Step 2: [First vertex] Choose a vertex v_x such that $d(v_x)=\max\{d(v) \text{ where } v \in V\}$ ties may be broken arbitrarily.
Step 3 : [Add chosen vertex to S] set $last:=v_x$, append $last$ to S.
Step 4: [Update actual]: **For** all $v \in V-S$, where v is adjacent to $last$ set $A(v):=A(v)+1$
While $V-S \neq \emptyset$ **do**
begin
Step 5: [Choose all vertices with maximum $A(v)$] Let MAX be the set of all vertices not in S with maximum $A(v)$.

$$MAX:=\{v \in V-S \mid A(v)=$$

$$=\max\{A(v) \text{ and } v \in V-S\}$$

Step 6: [Maximize potential] set $last:=w$ where $w \in MAX$ with the value of $d(w)$ as large as possible (break ties arbitrarily). Append $last$ to S.
Step 7: [Update actual] For all $v \in V-S$ update actual work as in Step 4.
end

□

5. k-Step Lookahead Algorithms

The intuition behind the k-step lookahead algorithm is to break ties (in Steps 1 and 6) in a more sophisticated way so that among all candidate nodes at a given decision step the one which creates more actual work for future steps will be chosen. We achieve this by looking not only at the potential work of a node w but also at the total potential work of its neighborhood. We can carry this reasoning forward and look at $\Gamma(\Gamma(w))$, the neighborhood of $\Gamma(w)$, etc. The resulting family of algorithms will become increasingly expensive but will provide better schedules. An algorithm which breaks ties by looking at $\Gamma^k(w)$ will be called k-lookahead algorithm. Using this notation, Algorithm 1 above is a 0-lookahead. In the following example we show that a 1-lookahead does better than a 0-lookahead algorithm on a two-processor system.

6. Examples

Figure 4 shows a page connectivity graph for two relations, with seven and ten pages respectively, and a total of seventeen join edges (note that there is no edge between nodes eight and thirteen). The k-step algorithms initially rank all nodes by their potential work values. For this graph, the nodes {2,8,9,10,11,12,14,15,16,17} have potential work values of five, {13,1} have values of four and the remaining nodes have work values of two. The 0-step algorithm will choose randomly from the first set of nodes for the initial page read, then rank the remaining nodes by their actual work values. In this example, node two is selected, which results in actual work values of one for the nodes {3,4,5,6,7}. Node six is then randomly selected, and the algorithm continues as described in Figure 5. The overall execution time is 22 time units, however note that during the period between time steps eight and eleven only one of the two processors is active.

The 1-step lookahead algorithm breaks ties by considering the neighborhood of unread nodes connected by a single edge to the candidate node. Candidate nodes are ranked by the sum of the potentials of their neighborhoods. For the given example, node two has a neighborhood potential value of ten, node eight has a value of of 22, nodes {9,10,11,12} have values of 24 and {14,15,16,17} have values of 25. The 1-step lookahead algorithm will choose randomly from the latter set, then recompute actual work values for all nodes and continue as above. Figure 6 shows a possible strategy generated by this algorithm. Note that although the strategy does not complete in optimal time, all of the processors are utilized throughout the period following the fourth page read. Before this page read, it is not possible to utilize

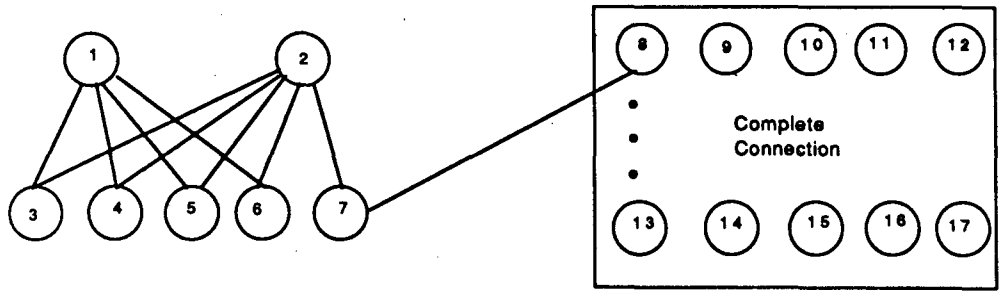


Figure 4: Example Page Connectivity Graph

more processors so this strategy is minimal.

7. Simulations

We coded Algorithm 1 in C and tested it on a large number of randomly generated graphs. A summary of some of these experiments is given in the charts below for the case of a graph with 100 nodes split equally between the two sets (relations). Figures 7 and 8 (at the end of this paper) show the effects of varying the number of edges on the time to complete the schedule and the

Time Step	Read Schedule	Processing Schedule	
		P1	P2
1	r2		
2	r6		
3	r1	2,6	
4	r3	1,6	
5	r4	1,3	2,3
6	r5	1,4	2,4
7	r7	1,5	2,5
8	r8	2,7	
9	r14	8,7	
10	r9	8,14	
11	r15	9,14	
12	r10	8,15	9,15
13	r16	10,14	10,15
14	r11	10,16	9,16
15	r17	8,16	11,14
16	r12	11,15	11,16
17	r13	11,17	8,17
18		9,13	9,17
19		10,13	10,17
20		11,13	12,13
21		12,14	12,15
22		12,16	12,17

Figure 5: 0-Step Lookahead Algorithm Output

processor utilization. Statistics are displayed for two, three and four processor systems. The DENSITY represents the fraction of edges present in the graph out of a maximum of 2500. Figures 9 and 10 show the results of fixing the graph at 750 edges, and varying the number

Time Step	Read Schedule	Processing Schedule	
		P1	P2
1	r14		
2	r9		
3	r15	9,14	
4	r10	9,15	
5	r16	10,14	10,15
6	r11	9,16	10,16
7	r17	11,14	11,15
8	r12	11,16	11,17
9	r8	9,17	10,17
10	r13	12,14	12,15
11	r7	12,16	12,17
12	r2	12,13	9,13
13	r6	10,13	11,13
14	r1	8,17	8,14
15	r5	8,15	8,16
16	r4	7,8	2,7
17	r3	2,6	2,5
18		2,4	2,3
19		1,6	1,5
20		1,3	1,4

Figure 6: 1-Step Lookahead Algorithm Output

of processors from one to eleven. With eleven processors the schedule completes in optimal time and no further improvement is possible.

As expected, adding more processors decreases the time to complete the schedule, but the marginal improvement tends to decrease. Processor utilization increases with the density of the graph and decreases with the number of processors. It is interesting to note that in all these experiments the actual time was within 1% of the lower bound given in Theorem 3.

8. Conclusions

In the previous sections we presented upper and lower bounds on the number of processors required to complete join processing in optimal time, where optimal time is defined as the minimum number of time steps

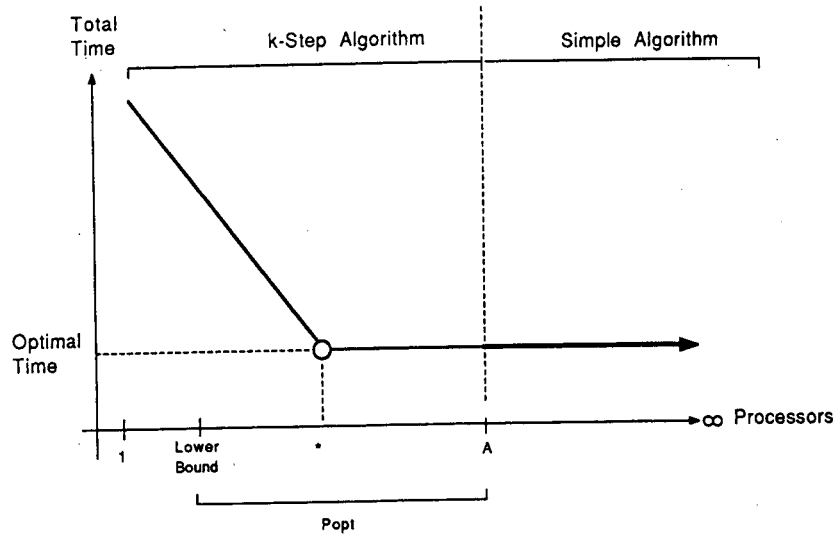


Figure 11. Summary of Results

with an unbounded number of processors. If the number of processors is less than the lower bound, then no read schedule exists which will allow processing to complete in optimal time. If the number of processors is greater than or equal to the upper bound, then an algorithm exists which is guaranteed to produce a read schedule which will finish in optimal time. However, at the upper bound processor utilization may be as low as 50%, as described above. As the number of processors is decreased to fall somewhere in between the upper and lower bounds, processor utilization increases. However in this region the existence of a read schedule which allows join processing to complete in optimal time is no longer guaranteed. The k-step algorithm will come arbitrarily close to generating a minimal read schedule, however this schedule may not complete in optimal time if the number of processors is too small.

Figure 11 summarizes these results. We conjecture that there is some number of processors (* in the figure) which will allow an optimal processing schedule as well as maximize processor utilization.

9. References

[Aho74a] Aho,, Hopcroft,, and Ullman,, *The Design and Analysis of Computer Algorithms,* Addison-Wesley (1974).

[Bitt83a] Bitton, Dina, Boral, Haran, DeWitt, David, and Wilkinson, W. Kevin, "Parallel Algorithms for the Execution of Relational Database Operations," *ACM Transactions on Database Systems* **8**(3) pp. 324-353 (September 1983).

[Blas76a] Blasgen, Michael W. and Eswaran, Kapali P., "On the Evaluation of Queries in a Relational Data Base System," IBM Technical Report #RJ 1745 (#25553) Computer Science (April 8, 1976).

[Goya88a] Goyal, P., Li, H.F., Regener, E., and Sadri, F., "Scheduling of Page Fetches in Join Operations Using Bc-Trees," *Proceedings of 4th International Conference on Data Engineering,* (Feburary 1-5, 1988).

[Merr81a] Merrett, T., Kambayashi, Y., and Yasuura, H., "Scheduling of Page-fetches in Join Operations," *Proceedings 7th International Conference on Very Large Data Bases,* (1981).

[Murp85a] Murphy, Marguerite C., "Architectural Alternatives for Database Machines," Doctoral Dissertation, Memo No. UCB/ERL 85/87, ERL Laboratory, University of California, Berkeley, CA 94720 (5 November 1985).

[Pram85a] Pramanik, S. and Ittner, D., "Use of Graph-Theoretic Models for Optimal Relational Database Accesses to Perform Join," *ACM Transactions on Database Systems* **10**(1) pp. 57-74 (March 1985).

[Vald87a] Valduriez, Patrick, "Join Indices," *ACM Transactions on Database Systems* **12**(2) pp. 218-246 (June 1987).

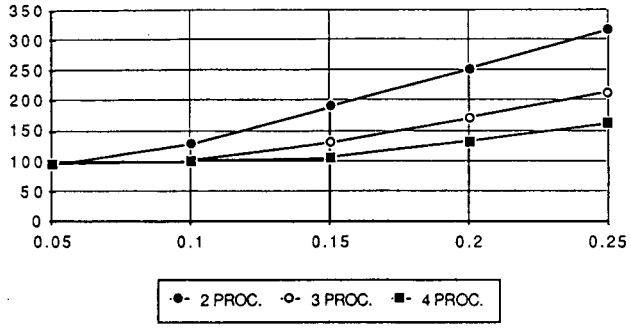


Figure 7. Schedule Execution Time vs. Density

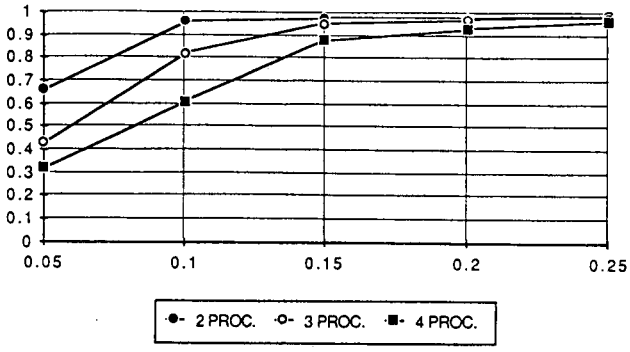


Figure 8. Processor Utilization vs. Density

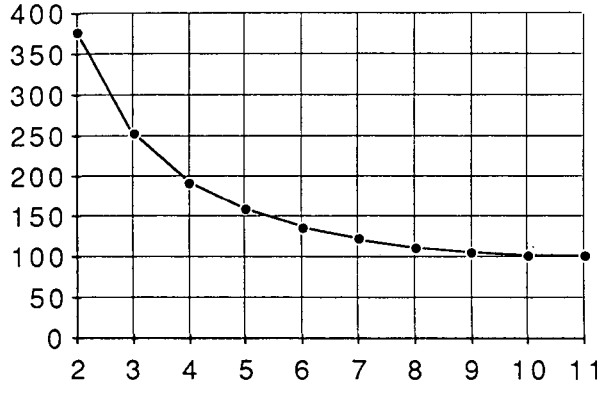


Figure 9. Schedule Execution Time vs Number of Processors

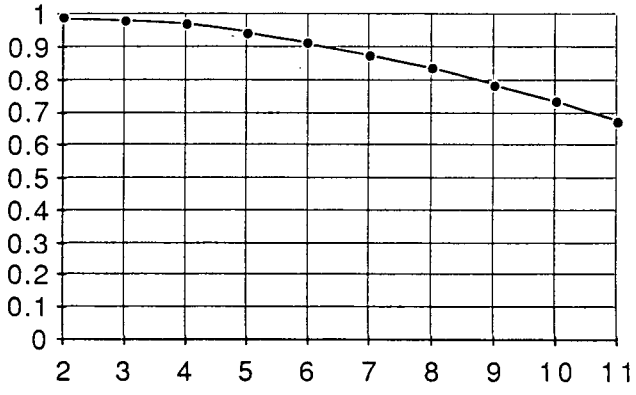


Figure 10. Processor Utilization vs Number of Processors

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
1 CYCLOTRON ROAD
BERKELEY, CALIFORNIA 94720