distribute the subrelations to the extra sites. The algorithm determines for each relation R_i the partitioning level, denoted by l_i, that should be used. The partitioning level indicates the number of fragments into which R_i should be divided. Each iteration of the algorithm considers the addition of one extra site if the savings obtained in local processing costs outweigh the additional data partitioning costs. The complete partitioning algorithm is given below.

Partitioning Algorithm(PART):

```
for $i = 1$ to r do
    $l_i = 1$; /* $l_i$ is the partitioning level of $R_i$ */
for $k = 1$ to $(n - r)$ do begin
    for $i = 2$ to $r$ do
        compute Gain($R_i^{l_i}$);
    Max_Gain = max$_i$(Gain($R_i^{l_i}$));
    if (Max_Gain > 0) then
        $l_j = l_j + 1$; /* $R_j$ is the relation for which Max_Gain is achieved */
end;
for $i = 1$ to r do
    partition $R_i$ into $l_i$ subrelations of equal size and distribute them to the extra sites;
where   Gain($R_i^{l_i}$) $= (S_{i,f}^{l_i} + S_{i,b}^{l_i}) - (S_{i,f}^{l_i+1} + S_{i,b}^{l_i+1}) + (PT_i^{l_i} - PT_i^{l_i+1})$.
        $S_{i,f}^{l_i}$ and $S_{i,b}^{l_i}$ are the local processing costs at site $S_i$ in forward and backward
        reduction given that $R_i$ is at partitioning level $l_i$; $PT_i^{l_i}$ stands for the partitioning
        cost.
```

Given n sites and r relations (with $n > r$), the CPU time complexity of the algorithm is $O((n - r)^* r)$.

Example 6. Let us consider the query $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$, and let us assume that there are 6 sites in the network. Initially all relations are at partition level 1. The gains to be computed are $(S_{i,f}^1 + S_{i,b}^1) - (S_{i,f}^2 + S_{i,b}^2) - PT_i^2$ for i = 2,3,4. Note that PT_i^2 denotes the cost to obtain a partition containing half the tuples of R_i at a new site. Let us assume that Max-gain is the gain obtained by partitioning R_3 into two equal sized partitions, one at its original site S_3 and the second at the extra site S_5. Since we have one additional site available, the actual partitioning does not occur yet, but we update the partitioning level of R_3 to 2. In the next iteration the gains obtained by partitioning R_2 and R_4 are unchanged, but Gain(R_3^2) becomes: $(S_{3,f}^2 + S_{3,b}^2) - (S_{3,f}^3 + S_{3,b}^3) + (PT_3^2 - PT_3^3)$. Now PT_3^3 denotes the cost of obtaining a partition of R_3 containing one third of the tuples of R_3. It is important to observe here that we should only consider subdividing a relation into fragments of equal size. A partitioning into unequal fragments will imply that the largest fragment becomes the bottleneck with regard to local processing costs. Let us assume that Gain(R_3^2) is largest among the gains considered. Thus the algorithm terminates with R_3 at partitioning level 3 and all other relations at partitioning level 1. This means that the additional sites S_5 and S_6 will each receive a fragment of R_3 whose cardinality is 1/3 of the original cardinality of R_3.

15

The modified pipeline algorithm, which we shall denote by ***partitioned pipeline***, is almost identical to the original, except that it makes use of broadcasting in both reduction phases. Thus, for example, if R_{i+1} has to be reduced by R_i and R_{i+1} is fragmented among a number of sites, then R_i will send its tuples identifiers to all these sites. The cost model for the response time of the modified pipeline algorithm is given in Appendix B.

The ***adaptive*** algorithm for response optimization chooses the algorithm best suited for a particular system and data configuration as summarized below.

```
Algorithm adaptive response time:
Case 1: no extra sites available
    if ($Response_{PIPE} > Response_{PAR}$)
    then use Parallel Algorithm;
    else use Pipeline Algorithm;
Case 2: extra sites available
    if ($Response_{PIPE} < Response_{PAR}$)
    then use Partitioned Pipeline Algorithm;
    else if ($Response_{PART} < Response_{PAR}$)
        then use Partitioned Pipeline Algorithm;
        else use Parallel Algorithm;
```

5. Experimental results

We have implemented our experiments on a SUN SPARC-IPX workstation having 16 MB main memory. The test relations were created from the Wisconsin benchmark database [4], with some modifications as explained in Section 5.1 below. Our simulation experiments are based on actual runs which compute the I/O time and CPU time explicitly by making use of the UNIX 'time' facility. In the absence of an actual distributed database system, the communication time was calculated by measuring the amount of data being transmitted and dividing it by the actual bandwidth. In our experiments we assumed a local area network with a bandwidth of 10 Mb/sec. The cost model developed in Appendix B is used only in Section 5.2.3 in order to determine the partitioning levels of the relations in PART. Hence, it is important to distinguish between the estimated values of the output parameters which are used in the cost model and the output parameters given in the simulation experiments which are based on actual runs. The estimated values of the output parameters are used by the cost model for response time optimization in order to decide which version of the algorithm to use.

Our workstation was used when the load on the server was low. In order to discount the I/O delay caused by competing users, we replicated each experiment five times and took the best timing value as the result since the best timing value is the closest to the actual time taken when the experiment is run stand-alone. Each run was treated as a separate process; hence all the buffers were flushed before the start of the next run.

The experiments were performed with chain queries. They were further subdivided into two major categories: one, for which we assumed that the bipartite graphs fit in main memory and a second, geared toward disk-based systems. Since the Wisconsin benchmark database contains relations where the attribute values are uniformly distributed, we used this set-up in most of our experiments. We performed also experiments with skewed data in order to study the robustness of our PART algorithm.

5.1. *The workload*

Each of the synthetic databases used consisted of relations having 3 attributes of size = 4 bytes. We deliberately chose small tuple sizes in order to avoid a priori a bias in favor of our method which uses only bipartite graphs versus the other approaches which use temporary relations with a variable number of columns. We report here on experiments with 4-way joins.

For the first set of experiments reported here we modified slightly the Wisconsin benchmark [4] in order to be able to account for all the cost factors in our model. Thus, in order to be able to generate backward messages in the pipeline algorithm when we perform a join $R_{i-1} \bowtie R_i$, it is necessary to transmit some forward messages (id_{i-1}, a_{i-1}) for which there are no matching id_i's at S_i. Similarly, to generate *Right_messages* during right reduction with the PAR algorithm, it is necessary to have some tuples (id_i, a_i) for which there are no matching id_{i-1}'s at S_{i-1}. By using the original benchmark, it is not possible to generate relations which satisfy these two conditions at the same time.

Our experiments with the modified Wisconsin benchmark were performed with a database consisting of 4 relations, each having the schema (*unique*1, *unique*2, *join_attr*). The range of *join_attr* in each relation was defined so as to achieve backward messages. Thus *join_attr* has a range of [0–4999] in R_1, [50–5049] in R_2, [75–5074] in R_3, and [85–5084] in R_4 with uniform distribution. Using these relations and the query class $Q_1 = select_S(R_1).join\_attr = R_2.join\_attr$ and $R_2.join\_attr = R_3.join\_attr$ and $R_3.join\_attr = R_4.join\_attr$, where S stands for a selection criterion, we defined three test sets, at shown in Table 2. The database dependent parameters were varied by changing the cardinalities of the relations and the 2-way join selectivities between adjacent relations R_i and R_{i+1} in the optimal join sequence, denoted by g_i in Table 2. The query dependent parameters were varied by choosing different selection criteria S on R_1 prior to performing the join.

The final join selectivity for an N-way join is computed as follows:

$$\text{final join selectivity} = \frac{\text{Cardinality of the final join result}}{\prod_{k=1}^{N} |R_k|}$$

Therefore, small variations in the final join selectivity imply quite significant variations in the cardinality of the final join result for relations of reasonable size.

A second set of experiments was performed with a testbed conforming to the original Wisconsin benchmark. This testbed was chosen in order to experiment with a workload for which our approach generated the smallest communication costs; hence no backward messages were desired. All 4 relations had identical schemas, i.e., R_i(*unique*1, *unique*2, *sel_attr*), but *sel_attr* has been used as a dynamic attribute which was substituted for one of the attributes *two*, *four*, *ten*, *twenty*, and *hundred* in different experiments. This enabled us to obtain different values for the final join selectivity using the same selection

Table 2. Workload for test sets 1–3.

	Test set 1	Test set 2	Test set 3
Database dependent parameters			
Card(R_i)	(5000,20000,20000,40000)	(5000,20000,40000,30000)	(30000,40000,30000,30000)
$g_i/(e-4)$	(1.86,1.99,1.996)	(1.86,1.99,1.996)	(1.98,1.99,1.99)
Query dependent parameters			
S = (*join_attr* <)	(100, 200, 300, 400)		(100, 120, 140, 160)
Output parameters			
Card(join)	(1920,14720,27520,40320)	(2880,22080,41280, 60480)	(25920,60480,95040,129600)
final sel./(e-13)	(0.24,1.84,3.44,5.04)	(0.24,1.84,3.44,5.04)	(0.24,0.56,0.88,1.20)

Table 3. Workload for test sets 4 and 5.

	Test set 4	Test set 5
Database dependent parameters		
sel_attr	{*two*, *four*, *ten*, *twenty*, *hundred*}	
Card(R_i)	(20000,20000,20000,20000)	(10000,10000,10000,10000)
g_i/(e-4)	(0.5,0.5,0.5)	(1.0,1.0,1.0)
Query dependent parameters		
S	*sel_attr* = 0	
Output parameters		
Card(join)	(200,1000,2000,5000,10000)	(100,500,1000,2500,5000)
Final sel./(e-14)	(0.125,0.625,1.25,3.125,6.25)	(0.125,0.625,1.25,3.125,6.25)

criterion $sel\_attr = 0$ in the query class. The query class for this set of experiments was $Q_2 = (Select_{sel\_attr=0}\ (R_1)).unique1 = R_2.unique2$ and $R_2.unique1 = R_3.unique2$ and $R_3.unique1 = R_4.unique2$. Using these relations and the query class Q_2, we generated two additional test sets, 4 and 5, whose parameters are summarized in Table 3.

The parameters used by the cost model are given in Table 4. We used the following values for the hardware related parameters:

$$C_{\text{compare}} = 3\ \mu\text{s},\ C_{\text{sort}} = 4\ \text{s},\ P = 1\ \text{Kbytes},$$
$$m = 1\ \text{Kpages},\ C_{i/o} = 25\ \text{ms},\ TR = 10\ \text{Mb/s}$$

The values of the remaining parameters are determined from the Wisconsin benchmark.

5.2. *Main-memory systems*

We report here on experiments falling in three subcategories. In the first subcategory of experiments we compared the performance of our PIPE_CHQ algorithm with the approach

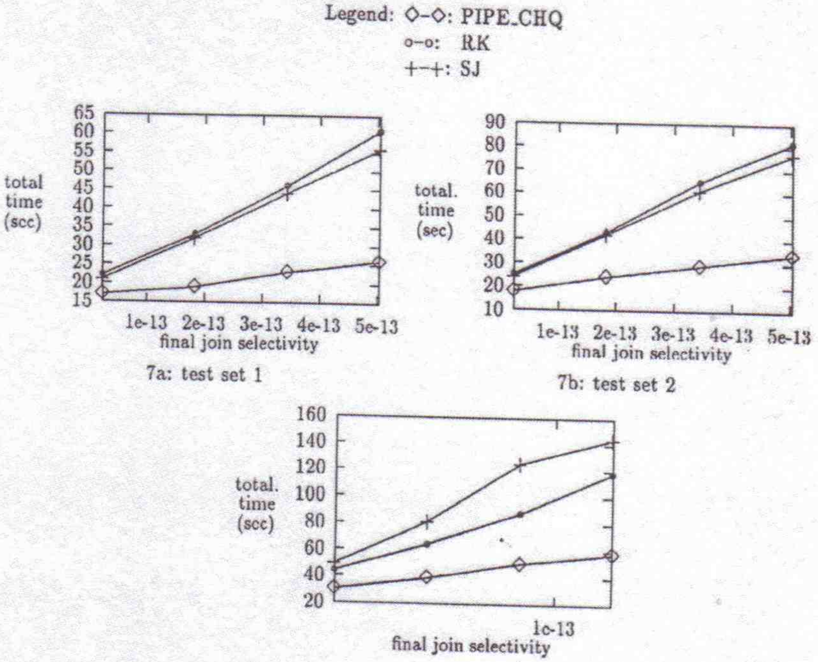
18

Table 4. Parameters used for cost estimation.

n	number of sites
r	number of relations
S_i	site i
R_i	relation i
ID (or ID_i)	stands for the tuple identifier (or key) of a relation
$R(ID)$	denotes the set of distinct values of ID in relation R
$R(attributes)$	denotes the projection of R on *attributes*
n_i	cardinality of R_i
$\|R_i(X)\|$	size of attribute X of R_i in bytes
$\|R_i\|$	size of a tuple in R_i in bytes
TR	data transmission speed in bytes/sec
$C_{i/o}$	I/O time per page
m	maximum number of pages that can be sorted at a time in main memory
C_{sort}	CPU time to sort m pages
C_{compare}	CPU time for one comparison
P_i	size of R_i in pages
P	page size in bytes
PT_i	time to partition R_i
f_{ij}	semijoin selectivity of R_j by R_i
g_i	join selectivity between R_i and R_{i+1}

of Roussopoulos and Kang [21] (to be abbreviated as RK) and the semijoin approach of Bernstein and Chiu [2] (to be abbreviated by SJ). In the second subcategory of experiments we compared the performance of the PART algorithm with the partition-based scheme developed by Shasha and Wang [25] (to be abbreviated as SW).

5.2.1. PIPE_CHQ versus RK and SJ. The SJ method performs forward and backward reduction using semijoins. Forward reduction is being performed from site 1 until site n, while backward reduction is being performed from site n until site 1. The reduced relations are then sent to the final site where the join is being computed. We use merge-sort to perform the reductions and the final join. The RK method requires constructing for each relation tuple connectors that consist of the projection of the relation on all its joining attributes and a tuple identifier. These tuple connectors are being constructed during the forward phase and then are reduced during a backward phase. In addition to tuple connectors, each site also constructs during the backward phase an incremental pipeline planner, with the final pipeline planner constructed at site 1 corresponding to our implicit join of the bipartite graphs. The reduced tuple connector at site i is constructed by performing a semijoin between the original tuple connector constructed in the forward phase and the incremental pipeline planner of site $i+1$. In order to perform this semijoin in the backward phase site $i+1$ needs to send to site i its incremental pipeline planner. It is important to note here that



7a: test set 1

7b: test set 2

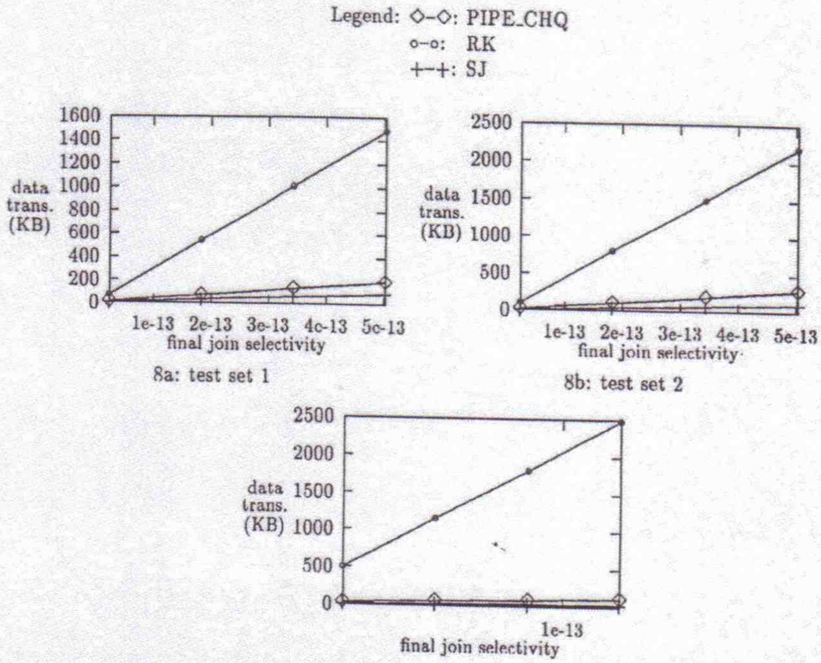
7c: test set 3

Figure 7. Total time of PIPE_CHQ algorithm compared to other methods.

the incremental pipeline planners are increasing in size as we proceed towards site 1 and, hence, are much larger than our bipartite graphs. Finally, the pipeline planner constructed at site 1 is sent to the query site where the actual join is computed.

We first compared the performance of PIPE_CHQ with RK and SJ on the modified Wisconsin benchmark, i.e., using test sets 1–3. Figure 7 shows the total processing times of the three algorithms using the workloads of test sets 1–3, while the communication costs are compared in figure 8.

The PIPE_CHQ method produced substantial savings in the total processing time: 23–59% as compared to the RK method and 19–60% as compared to the SJ. As expected, the savings become more substantial for higher values of final join selectivity. In this set of experiments all the attributes of the relations appear in the query target and SJ has the smallest communication costs, while RK is 4 to 20 times worse than PIPE_CHQ. The high communication costs of RK are due to the fact that it needs to transmit the incremental pipeline planners during the backward phase and the pipeline planner during the final step. We observe here that their final pipeline planner containing n-tuples of *ID*s exceeds by far the space needed by the $(n - 1)$ bipartite graphs shipped in our method. A second set of experiments were conducted with the original Wisconsin benchmark, using test sets 4 and 5. As explained above, this workload generated fewer communication costs for PIPE_CHQ. In addition, we



8a: test set 1

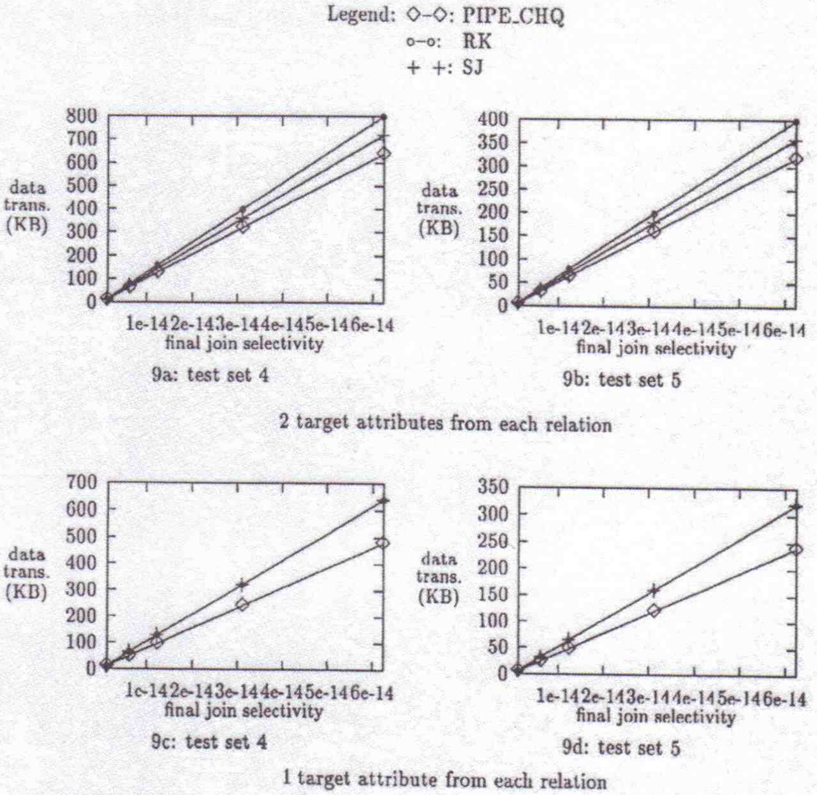
8b: test set 2

8c: test set 3

Figure 8. Comm. costs of PIPE_CHQ algorithm compared to other methods.

now varied the number of attributes from each relation to be included in the query target. Since our method does not require that the join attributes be sent to the query site in the final step, as is required in SJ, it achieves lower communication costs when some or all of the join attributes are not part of the query target. Figure 9 shows the communication costs of the three algorithms using the workload of test sets 4 and 5 and a variable number of target attributes. As expected, the smaller the number of target attributes from each relation is, the larger are the gains in communication costs of PIPE_CHQ. Thus, on this testbed for two target attributes from each relation, the communication costs of SJ are about 13% higher than those of PIPE_CHQ, while RK is 25% worse than PIPE_CHQ. In the case of one target attribute from each relation, the communication costs of SJ and RK are 33% higher than those of PIPE_CHQ. In [24] we also report on experiments with 10-way joins. These results show that as the cardinality of the final join increases, the predominant cost of the total processing time is the I/O cost necessary to materialize the implicit join.

5.2.2. PART versus SW. In order to compare the performance of our partitioned pipeline algorithm (PART) with other leading methods based on partitioning, we used the test sets 1–3 for which the response time of PIPE_CHQ is smaller than the response time of PAR_CHQ. If



9a: test set 4

9b: test set 5

2 target attributes from each relation

9c: test set 4

9d: test set 5

1 target attribute from each relation

Figure 9. Comparison of communication costs for different numbers of target attributes.

extra sites are available, our adaptive procedure will select PART as the algorithm of choice. As a competitor to PART, we chose the SW partitioning method proposed in [25] which partitions the relations across all sites using a hashing method. A dynamic programming algorithm was developed to determine the optimal join sequence assuming that all relations are partitioned. At each step a new intermediate relation of minimum cost is generated and distributed across the sites. For our test queries we computed the optimal join sequence using the SW method and used it for the evaluation of both methods. We assumed that none of the relations were originally partitioned and then generated the appropriate partitioning scheme as required by SW. In our simulation experiments we calculated the response time for SW by including the time to partition the relations, i.e., transmission time and I/O time. In order to decide on the partitioning level of the relations in PART we employed the cost model developed in Appendix B. Note again the distinction betweeen the estimated values of the output parameters which are used in the deciding the partitioning levels in PART and the actual simulated output parameters given by the experiments.