Offloading Bloom Filter Operations to Network Processor for Parallel Query Processing in Cluster of Workstations

V. Santhosh Kumar[1], M. J. Thazhuthaveetil[12], and R. Govindarajan[12]

[1] Supercomputer Education and Research Centre
[2] Department of Computer Science and Automation,
Indian Institute of Science,
Bangalore 560 012, India
gvsk@hpc.serc.iisc.ernet.in
{mjt,govind}@{csa,serc}.iisc.ernet.in

Abstract. Workstation clusters have high performance interconnects with programmable network processors, which facilitate interesting opportunities to offload certain application specific computation on them and hence enhance the performance of the parallel application. Our earlier work in this direction achieves enhanced performance and balanced utilization of resources by exploiting the programmable features of the network interface in parallel database query execution. In this paper, we extend our earlier work for studying parallel query execution with Bloom filters. We propose and evaluate a scheme to offload the Bloom filter operations to the network processor. Further we explore offloading certain tuple processing activities on to the network processor by adopting a network interface attached disk scheme. The above schemes yield a speedup of up to 1.13 over the base scheme with Bloom filter where all processing is done by the host processor and achieve balanced utilization of resources. In the presence of a disk buffer cache, which reduces both the disk and I/O traffic, offloading schemes improve the speedup to 1.24.

1 Introduction

Cluster computer systems, assembled from commodity off-the-shelf components, have emerged as a viable alternative to high-end custom parallel computer systems for applications demanding high performance [1]. An important component in such cluster computer systems is their high performance interconnect with programmable network interfaces such as Myrinet, Quadrics, and Infiniband [6]. The network processors available in such programmable interfaces often have the capability to perform application related processing, thus facilitating interesting opportunities to enhance application performance in cluster systems.

Our earlier work [10] demonstrates offloading the tuple splitting computation (to determine on which cluster node a tuple is to be processed), and tuple processing computation to the network processor results in significant performance improvement. Further, this work [10] also explored the benefits of attaching the disks to network interface.

In this paper, we first study the performance of a base case where all the application related tuple processing is performed by the host processor (HP). Using a Bloom filter [7] reduces the host processor workload as it eliminates the processing of tuples that are not selected (whose join key attribute do not match) by the join operation. Further, the Bloom filter reduces the data transferred over the network. Our simulation results indicate that although query execution time reduces significantly, HP utilization still remains high. We therefore propose offloading the Bloom filter operations from the host processor (HP) to the network processor (NP). Performance evaluation indicates a reduction in execution time by about 8%. Further offloading of tuple processing activities is possible by attaching the disk directly to NI as in [10]. This results in a performance improvement of upto 13%. With this offloading the disk becomes the bottleneck. To overcome this we study the impact of caching of tuples of frequently used relations or intermediate relations. With a cache hit ratio of 0.5, our schemes result in an execution time speedup of 1.24 over the Base scheme with an identical cache hit ratio. Further the utilization of key resources *viz*, HP, NP, and Disk are well balanced.

In the following section we provide the necessary background. In Section 3 we describe the Petri net model developed in [10] for the *Base Scheme* and extend the same for the Base scheme with Bloom filter. Section 4 discusses the *NP Bloom* scheme where Bloom filter operations are offloaded to NP. In Section 5 we describe the *Network Interface with attached Disk and Bloom Filter* scheme and evaluate its performance. Concluding remarks are provided in Section 7.

2 Background

Clusters: A cluster of workstations is a distributed memory machine where each node is a stand-alone system, with CPU and memory connected by the memory bus, and peripherals like disk, and network interface attached to the I/O bus. Typical high performance cluster interconnects like the Myrinet network interface (NI) [13] have NIs with a programmable network processor (NP), on board memory (SRAM), a host DMA engine (HDMA), an EBUS (External Bus) Interface (64-bit), a send DMA engine (SDMA) and a receive DMA engine (RDMA). The HDMA is used to transfer data across the I/O bus to the node memory. SDMA and RDMA are used to transfer data from the NI SRAM to the communication network (switch), and vice-versa. It is also equipped with a memory-to-memory copy engine (MCE). The switch employs wormhole routing to transfer packets between network interfaces. Research [14] in communication layers for high performance scientific applications has led to the development of user-level communication techniques which have reduced the involvement of HP in communication to deliver better application performance. We, therefore, assume such a user-level communication layer in our system.

Parallel Query Processing: In a relational database system queries composed of relational operators like select and join are used to manipulate data. The join operator, which combines tuples from two relations based on a common attribute, is the most crucial and expensive operator [8]. Hash-based join algorithms are more efficient than other join algorithms, such as sort-merge or nested-loop, in systems with large main memories [8]. So, in this work, join operators in queries are executed using hash-join

algorithms. Several parallel query processing techniques have been devised and employed in parallel database machines to improve query execution time [4]. A cluster of workstations is essentially a shared-nothing architecture, in which intra-operator parallelism is better exploited [4, 11] with horizontally partitioned relations. We therefore consider only exploiting intra-operator parallelism and horizontally partitioned relations.

When a query involves multiple joins, a query tree or a query execution plan is used to represent the scheduling sequence of the constituent operations. Query trees are characterized as left-deep, right-deep or bushy trees. Right-deep and bushy trees have multiple operators simultaneously active, and are suitable for pipelined implementations in multiprocessor systems [12, 3]. Left-deep trees plan allows only one operator is active on all the nodes and is well suited to shared-nothing architectures. Hence, we adopt a left-deep query tree plan.

In the hash join, there are two phases, namely (i) the Build Phase, where the inner relation is hashed on the join attribute and the hash table is built and (ii) the Probe Phase, where the outer relation is hashed on the join attribute using the same hash function used to probe the hash table, and result tuples are generated on successful matches. In the parallel version, first a separate hash function is used to determine the cluster node where the tuples will be processed. We refer to this activity as tuple splitting. Tuples are routed to their cluster nodes and then the join operation (build or probe) takes places on that node. Thus both phases of parallel join involve communication between cluster nodes. In case select or project operation exists along with join, select or project is first applied on the tuples before the join operation.

Bloom Filter: Bloom filters [2] are used in distributed databases to improve the join performance by reducing the amount of tuple processing being performed by the host processor and the amount of data transferred over the network [7]. Bloom filter is a bit vector representation of the set of keys which can be queried to check if a key is present. This is used in the join process as follows: Initially the bit vector is initialized to all zeros. Each tuple of the inner relation is hashed on the join attributes to the corresponding bit in the bit vector. Then the same hash function on the join attribute of the second relation is used to generate an index. If the corresponding bit of the bit vector is zero, then the actual hash table probe operation can be avoided. On the other hand a successful check in the bloom filter does not necessarily indicate that the key is present in the hash table, and so the probe operation has to be done. Thus, the Bloom filter can give rise to false positives.

3 Base Scheme

In our Base Scheme all activities related to query processing are performed by the host processor (HP). The network processor (NP) is involved only in message sends and receives among the cluster nodes. We evaluated the performance of the Base Scheme through simulation of a Petri Net (PN) model.

3.1 Base Scheme Description

Figure 1 shows our Petri net model of a single node in the cluster performing the parallel query execution on a cluster. Since our modifications are centered around the hash join, for clarity, the PN model for the join operations are described. We use the name of a timed transition *e.g.*, T_Build, to represent the duration of the timed transition.

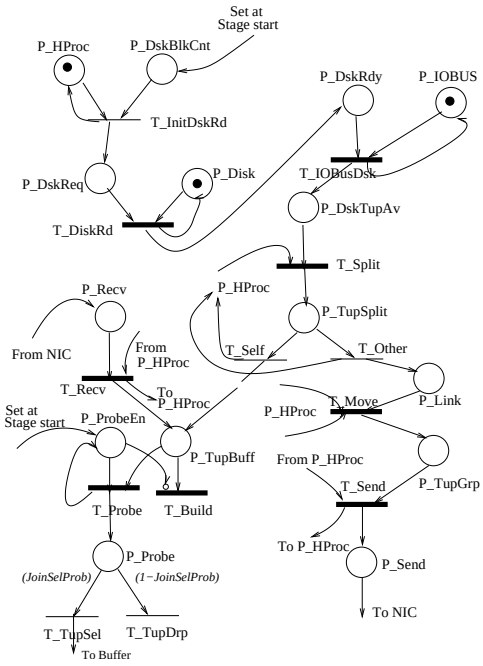

Fig. 1. Petri net model for Disk and Join operations

The number of tuples to be processed is modeled by available tokens in place P_TupAv. T_Split transition models the duration to compute the cluster $node\_id$ to which tuples are to be routed. T_Split fires depending on the availability of HP (modeled by place P_HProc). We assume that the tuples are uniformly distributed across nodes and hence T_Self fires with probability $1/N$ and and T_Other with a probability $(N-1)/N$, where N is the number of nodes in the cluster. T_Build represents the duration of per tuple build operation, which fires with the availability of HP, during the build phase. T_Probe represents the HP duration for per-tuple probe operation which is enabled during the probe phase. Based on the join selectivity (denoted by $JoinSelProb$), certain tuples qualify in the join process (modeled by T_TupSel) while others are dropped (modeled by T_TupDrp). Tuples which fire T_Other are grouped into messages (T_Move) and enqueued by HP to be transmitted by NI. T_Send and T_Recv model the software overhead incurred by HP for initiating a send/recv operation.

The architectural parameters of our model are set to represent contemporary high performance computing nodes. We measured the time taken for various tasks on a 2.4GHz Pentium 4 processor based system and scaled them to 3.6GHz, so as not to overestimate the benefits of offloading. The parameters for host communication overheads were obtained from measurements on Myrinet user-level messaging software running on Myrinet LANai 9.2 processor [6]. Network Interface Parameters values are estimated assuming a NI SRAM bandwidth of 2.664GB/s (333MHz, 64 bit bus). The network link bandwidth is assumed to be 4GB/s based on the Myrinet specifications [13]. This value is also used to fix the switch delay for a packet; the bisection bandwidth of the switch must be greater than the link bandwidth to support simultaneous connection between input and output ports of the switch.

The I/O bus transfer time parameters are set assuming 2GB/s, similar to the PCI-Express-1X bus [9] bandwidth. The disk I/O time is set assuming the ability to deliver 1.28GB/s (4 * 320MB/s SCSI disk), in 64KB chunks.

We set the database related parameters based on the TPC-H benchmark [16]. TPC-H queries 3,5,7-10 were modeled, each using a separate PN model. We assume a tuple size of 128B for all relations. The number of tuples per table is set so that the horizontal partition of the relation in each node occupies 1GB. Further, we assume that there is no skew in the data, *i.e.*, the frequency of all key values used in the join attributes occur with equal frequency. When join is performed, a part of the tuple is projected out. The size of the projected tuple is assumed to be 32 bytes. The model parameter values for selectivity ratios were obtained using measurements from query execution on a single node running PostgreSQL. The values for the various architectural parameters and those for the TPC-H queries used in our performance evaluation study can be found in [10].

We validated our Petri net model with performance measurements from a uniprocessor implementation of Hash-join, and MPI based implementations on 2 nodes and 4 nodes [10].

3.2 Performance of Base Scheme

We simulated our Petri net models using CNET, an event-driven petri net simulator [17]. The simulator reports the total simulation time for the Petri net model, as well as the total firing times for each timed transition. All reported results are for 8 node cluster[3] averaged across 3 independent runs for each query.

We use relative speedups of query execution times for performance comparison and the utilization of resources like host processor (HP), Disk, I/O Disk, I/O NIC, Switch (SW) and network processor (NP) to identify the bottleneck resources. In discussing the results, we report the average of the execution time of all the queries. Relative speedup of a scheme, is the ratio of the average query execution time in the Base scheme (with BS-1X parameters) to that in the proposed model.

Table 1 shows that query execution time is dominated by the tuple processing cost of HP, which has the maximum resource utilization of 97.4%. We found that doubling

[3] We have studied the performance for 4 node, 8 node and 16 node clusters in our earlier study [10], and found that the performance study exhibits similar results.

tuple processing cost of HP for Base Scheme (BS-2X) power yielded a speedup of 1.78 with respect to BS-1X configuration, showing that tuple processing activities done by HP are a significant factor in query execution time. This motivated us to study the effect of Bloom filter operations on the performance of query execution time.

3.3 Base Scheme with Bloom Filter

In the context of a parallel join operation, the Bloom filter can be incorporated in two ways depending on whether the filter operations are performed before the tuples are transmitted to their destination node (referred to as Global Bloom Filter), or after they arrive at the destination node (referred to as Local Bloom Filter). The advantage with Global Bloom Filter (GBF) is that, it does not transmit unwanted tuples to remote nodes by performing the filter operation first. This reduces I/O and network traffic. However it becomes necessary to exchange the Bit filters after the build phase. In contrast, in the Local Bloom Filter (LBF), the filter is built for tuples that arrive at the destination node.

To estimate Bloom filter model parameters we used an implementation of Jenkins's hash function[5] and measured the execution time for 6M keys, requiring 1 bit for 1 key [4]. We found that per key time for filter operations is 0.051μs, with the computation component of filter operation (hash function computation) is 0.0195μs and memory access time is 0.0315μs [5].

Table 1. Comparison of Resource Utilization of Base Schemes with Bloom Filter

Model	Utilization						Exec	Relative
	Disk	HP	I/O Disk	I/O NIC	SW	NP	Time(s)	Speedup
BS-1X	35.7	97.4	22.5	7.9	16.4	0.1	2.05	1.00
BS-2X	63.5	86.6	39.9	14.0	29.2	0.1	1.15	1.78
GBF	60.5	73.6	38.1	5.4	11.2	0.1	1.21	1.69
LBF	58.9	85.2	37.1	13.0	27.1	0.1	1.24	1.65

Table 1 shows the relative speedup and resource utilizations for Global and Local Bloom filter implementation referred to as GBF and LBF schemes, compared to the Base scheme (BS-1X). We find that both schemes give a significant speedup in execution time (1.69 for GBF and 1.65 LBF). Also, HP utilization reduces from 97.4% for the Base scheme to 73.6% and 85.2% for GBF and LBF respectively due to the early dropping of tuples and the reduction in probe operations during the Bloom filter operation.

Further, we observe that utilization of HP for GBF scheme (73.6%) is lower than that of LBF (85.2%) scheme. This is because GBF reduces the communication overhead incurred by HP. This results in an improved speedup in execution time for GBF scheme (1.69) as compared to LBF scheme (1.65).

[4] The Bloom filter gives a false positive rate of 22.1%

[5] We found that the time for BuildFilter and CheckFilter operations are approximately the same. This is reasonable due to the high memory access cost, which is required for both operations

While Bloom filter achieves a significant speedup in query execution, we find that HP has a high utilization and NP a low utilization. Hence we ran additional experiments for the GBF by doubling the host processing power and found an increase in speedup from 1.69 to 2.25, implying that the host processing power is the dominant factor with BS-1X configuration for the GBF scheme. We next study the impact of offloading Bloom filter operations to NP.

4 Network Processor Running Bloom Filter

Offloading the Bloom filter operations to the NP results in two schemes, NP-LBF and NP-GBF, depending on whether its a local or global filter scheme. In the NP-GBF, since the filter operations are to be performed before the transmitting the tuples to the destination, all the tuples would have be transferred to the NI. Earlier work [10] has shown that this could cause the I/O bus to be a bottleneck, restricting the performance enhancements possibilities. So we consider only the NP-LBF scheme where the filter operations are performed by the NP after the tuples arrive at the destination node.

Table 2. Comparison of Resource Utilization and Speedup: GBF vs. NP-LBF

Scheme	HP:NP	Utilization							Relative Speedup
		Disk	HP	I/O Disk	I/O NIC	SW	NP	MCE	
GBF		60.5	73.6	38.1	5.4	11.2	0.1	0.0	1.00
NP-LBF	1:1/8	56.9	63.3	35.8	2.3	26.2	59.8	3.10	0.97
	1:1/6	63.2	69.2	39.8	2.5	28.9	50.6	3.40	1.04
	1:1/4	65.3	72.7	41.1	2.6	30.0	36.5	3.50	1.08
	1:1/2	65.3	72.7	41.1	2.6	30.0	21.0	3.50	1.08
	1:1	65.3	72.7	41.1	2.6	30.0	13.2	3.50	1.08

Table 2 shows the resource utilizations and speedup for GBF and NP-LBF. We use GBF for comparison purposes as it was better of the schemes compared in Table 1. When $HP : NP$ ratio, which models the relative processing power of HP and NP, is 1:1/8, NP-LBF does not perform as well as GBF, having a relative speedup of 0.97. As the processing power of NP is increased, and at $HP : NP$ ratio of 1:1/4, the speedup improves to 1.08. Further, we see a reduction in NP utilization (from 59.8% to 36.5%), implying that it is no longer a bottleneck and its processing power is sufficient for Bloom filter operations to be performed by NP. The HP utilization increases to 72.7% suggesting that tuple processing costs are a dominant factor in query execution as NP processing power increases. Disk utilization is also comparable to that of HP, indicating that it is also a bottleneck resource.

Since HP is the resource with highest utilization, we also simulated the NP-LBF scheme for $HP : NP$ ratio of 1:1/4, with double the HP and NP processing powers. We observed that the relative speedup increases from 1.08 (NP-LBF-1X) to 1.42 (NP-LBF-2X). This motivated us to next consider further options to offload some tuple processing work from HP to NP.

5 Network Interface with Attached Disk and Bloom Filter

Here we consider the idea of attaching the disk to the network interface directly instead of to the system bus, proposed in our earlier work [10], which gives opportunity for additional tuple processing to be offloaded to the NP. The operations of $node\_id$ computation and communication of tuples can then be performed by NP, along with the Bloom filter activities.

There are two ways to organize the Bloom filter in the NID model as in the Base Scheme a) Local Bloom filter with NID (NID-LBF) b) Global Bloom filter with NID (NID-GBF). In both NID-GBF and NID-LBF schemes, the NP does the job of reading the tuples directly from the disk, routing the tuples to the destination node and the filter operations, and the HP performs the hash build and hash probe operations. They differ only in when the filter operations are performed, i.e., before sending the tuples to the remote node or on receiving the tuples at the remote node.

Table 3. Comparison of Resource Utilization and Speedup for NID Schemes

Scheme	HP:NP	Utilization							Relative Speedup
		Disk	HP	I/O Disk	I/O NIC	SW	NP	MCE	
GBF		60.5	73.6	38.1	5.4	11.2	0.1	0.0	1.00
NID-LBF	1:1/8	55.4	41.3	0.0	3.5	25.5	59.3	29.9	0.92
	1:1/6	61.1	45.5	0.0	3.8	28.1	50.4	33.0	1.01
	1:1/4	64.8	48.4	0.0	4.1	29.8	37.6	35.0	1.07
	1:1/2	67.5	50.3	0.0	4.2	31.0	23.2	36.4	1.12
	1:1	68.7	51.3	0.0	4.3	31.6	15.5	37.1	1.13
NID-GBF	1:1/8	57.3	42.9	0.0	3.6	9.4	61.2	6.4	0.95
	1:1/6	63.4	47.5	0.0	4.0	10.4	52.1	7.1	1.05
	1:1/4	68.2	51.0	0.0	4.3	11.2	39.3	7.6	1.13
	1:1/2	68.2	51.0	0.0	4.3	11.2	23.0	7.7	1.13
	1:1	68.2	51.1	0.0	4.3	11.2	14.9	7.6	1.13

Table 3 compares the resource utilizations and speedup for NID-LBF, NID-GBF, and GBF schemes. When for $HP : NP$ ratio is 1:1/8 both NID-LBF and NID-GBF schemes have a speedup less than 1. Further NP is the resource with highest utilization (59.3% for NID-LBF, 61.2% for NID-GBF), suggesting that NP's is the bottleneck resource. With an increase in the processing power of NP from 1:1/8 to 1:1/4, NID-LBF and NID-GBF attain speedups of 1.07 and 1.13 respectively. Also, NP utilization decreases to 37.6% and 39.3% for NID-LBF and NID-GBF respectively, indicating that NP is no longer the bottleck resource. Further increase in NP's processing power does not yield any benefit, as the disk unit becomes the resource with higher utilization (greater than 64.8%).

6 Performance with Disk Caching

The discussions in the previous sections indicate that when the processing power of NP is greater than 1/4th the processing power of HP, the bottleneck shifts to disk. Using

large memory caches for tuples of often used relations can help by reducing the load on the disk. Such caches can be modeled by means of a Buffer Hit Probability (BHP). or intermediate relations (in a sequence of database operations) improve performance. In this section we explore the possibility of using such a cache in GBF, NP-LBF, NID-LBF and NID-GBF schemes. For the later two schemes the cache has to be maintained in the NIC.

Table 4. Comparison of Resource Utilizations and Speedup with Memory Cache for BHP=0.5

Scheme	HP:NP	Utilization							Relative Speedup
		Disk	HP	I/O Disk	I/O NIC	SW	NP	MCE	
GBF	38.1	90.8	24.0	6.7	13.9	0.1	0.0	0.0	1.00
NP-LBF	1:1/8	34.1	68.4	21.5	2.4	29.6	68.8	3.6	0.89
	1:1/6	37.1	78.5	23.4	2.8	33.0	57.7	3.9	0.97
	1:1/4	40.9	88.2	25.7	3.2	36.6	44.5	4.3	1.07
	1:1/2	41.7	91.1	26.2	3.3	37.6	26.3	4.4	1.09
	1:1	41.7	91.1	26.2	3.3	37.6	16.5	4.4	1.10
NID-LBF	1:1/8	31.3	45.8	0.0	3.9	28.3	65.7	33.2	0.82
	1:1/6	34.9	51.4	0.0	4.3	31.7	56.8	37.1	0.92
	1:1/4	35.4	52.0	0.0	4.4	32.0	40.5	37.5	0.93
	1:1/2	40.6	59.5	0.0	5.0	36.7	27.4	43.1	1.07
	1:1	41.8	61.0	0.0	5.1	37.6	18.4	44.0	1.09
NID-GBF	1:1/8	32.5	48.1	0.0	4.1	10.5	68.6	7.2	0.86
	1:1/6	36.8	54.1	0.0	4.6	11.8	59.4	8.1	0.97
	1:1/4	41.6	61.7	0.0	5.2	13.5	47.5	9.2	1.10
	1:1/2	47.2	69.3	0.0	5.8	15.2	31.2	10.4	1.24
	1:1	47.4	69.6	0.0	5.9	15.2	20.3	10.4	1.24

We observe from Table 4 that the speedup is less than 1 when NP's processing power lower than 1/4th of HP. At lower processing power, NP is the bottleneck resource, as seen from the high utilization of NP $>$ 56.7%. As NP's processing power increases to 1:1, we see a relative speedup of 1.10, 1.09, and 1.24 for NP-LBF, NID-LBF and NID-GBF respectively. The additional improvement in performance is due to the buffer cache. We also find the the key resources HP, NP, Disk achieve balanced resource utilizations.

7 Conclusions

Optimizing the performance of these parallel query processing clusters is commercially important. In our earlier work we had evaluated both software and hardware modifications, exploiting the programmable features of NP to a achieve higher performance with balanced utilization of system resources. Using a Bloom filter reduces the host processor workload as it eliminates the processing on tuples that are not selected (whose join key attribute do not match) by the join operation. Further the Bloom filter also reduces the data transferred over network. In this work we offload the Bloom filter activities to the network processor and evaluated its benefits. We evaluate the performance of the

proposed modifications using timed Petri net models. We find that offloading Bloom filter processing from the host processor to the network processor results in execution time speedup of upto 1.24, and achieves a balance resource utilization. We suggest that if future network interfaces are equipped with programmable processor of high power, applications should be able to exploit them in improving system performance.

References

1. T. Anderson, D. Culler, and D. Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 16(1):54-64, Feb 1995.
2. B. Bloom. Space/time trade-offs in hash coding with allowable errors, *Communications of the ACM* 13(7):422-426, July 1970
3. M.-S. Chen, M.-L. Lo, P. S. Yu, and H. C. Young. Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins. *In Proc. of 18th Very Large Data Bases*, pages 15-26, Aug 1992.
4. D. DeWitt and J. Gray. Parallel Database Systems: The future of High Performance Database Systems. *Communications of the ACM* 35(6):85-98, Jun 1992.
5. B. Jenkins. http://burtleburtle.net/bob/c/lookup2.c 1996
6. J. Liu, B. Chandrasekaran, W Yu *et al* . Microbenckmark Performance Comparison of High-Speed Cluster Interconnects. *IEEE Micro*, 24(1):42-51, Jan-Feb 2004.
7. L. F. Mackert, and G. M. Lohman. R* Optimizer Validation and Performance Evaluation for Distributed Queries. In *Proc. of 12th Intl. Conf. on Very Large Data Bases*, 149-159, Aug 1986.
8. P. Mishra and M. H. Eich. Join Processing in relational databases. *ACM Computing Surveys*, 24(1):63-113, Mar 1992.
9. PCI-SIG Home, http://www.pcisig.com/, 2004.
10. V. Santhosh Kumar, M. J. Thazhuthaveetil, R. Govindarajan. Exploiting Programmable Network Interfaces for Parallel Query Execution in Workstation Clusters. TR-HPC-10/2005, LHPC, SERC, IISc, 2005 (`http://hpc.serc.iisc.ernet.in/Publications/gvsk2005.ps`).
11. D. Schneider and D. DeWitt. A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment. In *Proc. of ACM SIGMOD Conference*, Jun 1989.
12. D. Schneider and D. DeWitt. Tradeoffs in processing complex queries via hashing in multiprocessor database machines. In *Proc. of 16th Intl. Conf. on Very Large Data Bases*, Aug 1990.
13. C. L. Seitz. Myrinet Technology Roadmap. *Myrinet Users Group Conf.* May 2002.
14. V. Karamcheti, and A. Chien. Software Overhead in Messaging Layers: Where Does the Time Go? In *Proc. of 6th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct 1994.
15. T. Tamura, M. Oguchi, and M. Kitsuregawa. Parallel Database Processing on a 100 Node PC Cluster: Cases for Decision Support Query Processing and Data Mining. In *Proc. of Supercomputing*, Nov 1997.
16. TPC Benchmark^{TM} H (Decision Support) Standard Specification Revision 1.3.0. Transaction Processing Performance Council(TPC), 1999.
17. W. M. Zuberek. Modeling using Timed Petri Nets - event-driven simulation, Technical Report No. 9602, Dept. of Computer Science, Memorial Univ. of Newfoundland, St. John's, Canada, 1996 (`ftp://ftp.cs.mun.ca/pub/techreports/tr-9602.ps.Z`).