VICTORIA UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF COMPUTER AND MATHEMATICAL SCIENCES

Limits to Database Speedup for Data Partitioned in Unary Relational Operations

C. H. C. Leung and K. H. Liu

(72 COMP 26)

April 1996

(AMS : 68P15)

TECHNICAL REPORT

VICTORIA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER AND MATHEMATICAL SCIENCES
P O BOX 14428
MCMC
MELBOURNE, VICTORIA 8001
AUSTRALIA

TELEPHONE (03) 9688 4492
FACSIMILE (03) 9688 4050

Footscray Campus

Limits to Database Speedup for Data Partitioned Parallelisation in Unary Relational Operations

C. H. C. Leung and K. H. Liu

TERABYTE DATABASE GROUP
Department of Computer & Mathematical Sciences,
Victoria University of Technology,
Ballarat Road, PO Box 14428, MMC,
Melbourne, Victoria 3000, AUSTRALIA

Abstract

Data partitioning of a given relation consists of distributing its data tuples horizontally over the available processors for simultaneous processing. Here, we are concerned with intra-operation parallelism which consists of the application of a given unary relational operation to data located across different processors. A homogeneous distributed memory configuration is used in which inter-processor communication is kept to a minimum, and all the participating processors are assumed to operate at the same speed. Stochastic variation is unavoidable in most data partitioning schemes which can lead to data skew. Such skew comes from the uneven data load assigned to different processors, and can cause significant erosion in parallel processing efficiency. Conventional performance estimates often gives an improvement factor of k, where k is the number of processors participating in the processing. This study shows that in most data partitioning schemes, the actual performance can depart significantly from this ideal, and closed-form expressions for quantifying the execution time, processor utilisation, and load imbalance caused by skewness are provided.

1. Introduction

Data partitioning provides a particularly effective way of achieving performance improvements for data intensive applications. For large database systems, *data parallelism* offers much greater scope for concurrent operation than *control parallelism* since the number of data fragments that can be partitioned is generally much higher than the number of possible control threads which can execute in parallel. In the processing of database queries at the processor level, three forms of parallelism can be identified (see Figure 1). The first is *inter-query parallelism* whereby a number of concurrent queries are processed in parallel. Typically, in this form of parallelism, a single processor is allocated to a query and no attempt is made to decompose the individual queries further for parallelisation. The second type is *inter-operation parallelism,* which goes beyond the first type by decomposing a query into a number of relational algebraic operations and executing these operations in parallel. With this form of parallelism, only a single processor is allocated to an operation and no attempt is made to parallelise the individual operations. The third type of parallelism is *intra-operation parallelism* whereby the individual algebraic operations are also parallelised. Apart from the first type of parallelism, the second and third type may both be regarded as *intra-query parallelism.*

Parallelisation of database operations can focus on either the disk component and the processor component. At the disk level, parallelism can be introduced by special purpose hardware which not only allows the reading of multiple tracks or cells in parallel, but also the filtering off of irrelevant data before they are transferred to main memory [Leun85, Ozka86]. The substantial difference in processor speed and the mechanical secondary storage access speed makes the introduction of disk parallelism especially worthwhile. However, as increasing flexibility is demanded of current database systems, disk access time is no longer the sole contributor to performance delays and main memory processing is approaching a magnitude comparable to disk processing. Here, we concentrate on parallel execution at the processor level, and assume that data is already retrieved from disk, so that I/O delays will not be considered to be directly relevant. Partly because of the steady increase in DRAM capacity, main memory based database processing has been increasingly common [Litw92, Moss92]. Moreover, with increasing database sizes and processor intensive

operations, the processor time required for executing complex queries can be substantial.

In order to achieve the best performance improvement from parallelisation, it may be necessary to first transform database queries into forms that are amenable to parallel processing at the internal level. Heuristics for doing so have been considered in [Leun93]. Here, we focus on performance improvement at the internal level, and mathematical models are employed to study the limits to performance speedup caused by load imbalance.

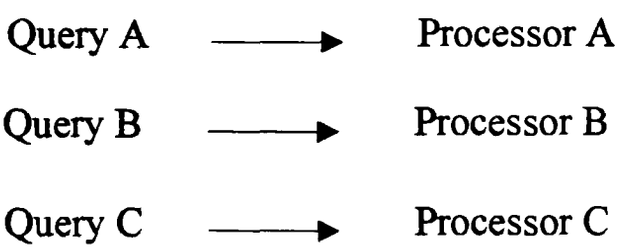


Fig. 1a ***Inter-query Parallelism***

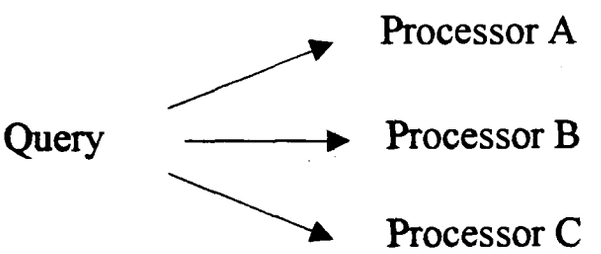


Fig. 1b ***Inter-operation Parallelism***

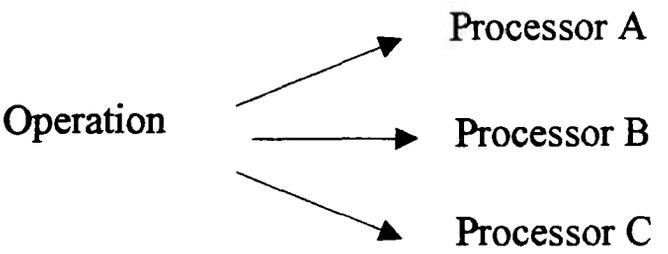


Fig. 1c ***Intra-operation Parallelism***

2. Relationship with Other Work

Skew existence causes load imbalance, curtails the scalability of multiprocessor systems, and degrades the system performance dramatically. Skew management involves two basic steps, skew estimation and skew handling. Skew estimation attempts to identify the existence of the skew among parallel processors and, if exist, estimate the extent of the skew. Based on this knowledge, the skew handling then attempts to avoid or resolve skewed load distribution using various approaches. A classification of the skew estimation and handling approaches are presented in Figure 2. The main features of each approach as well as existing algorithms are discussed briefly below.

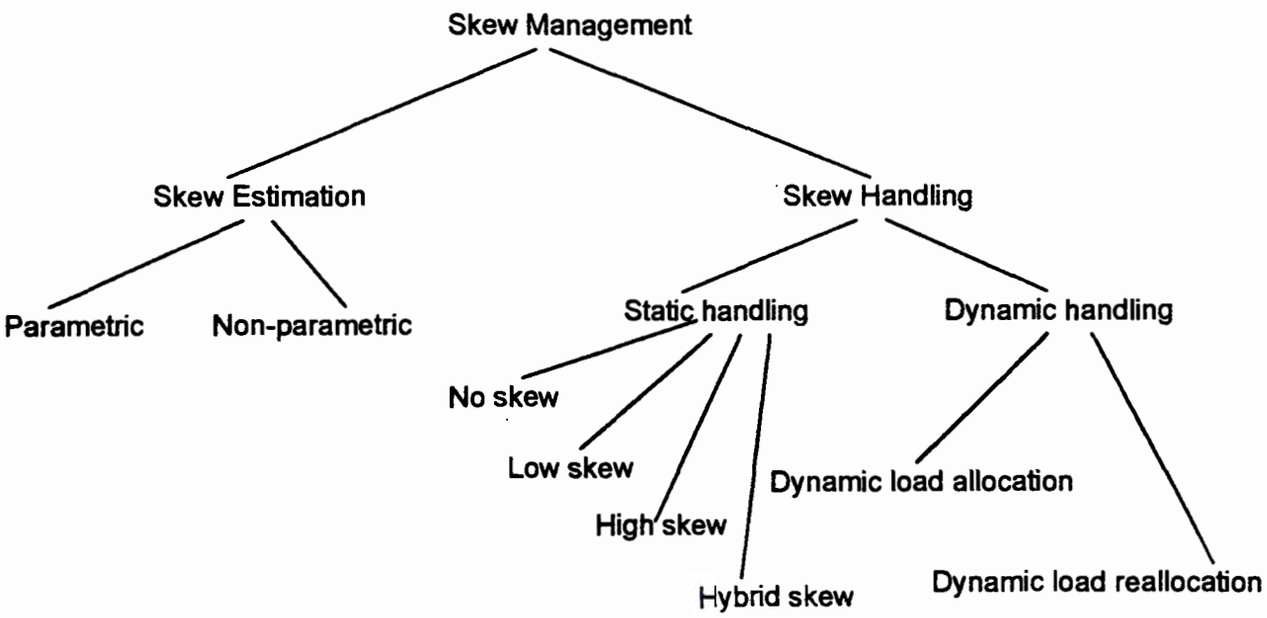


Fig. 2

Most works have been concerned with skew handling whereas a relatively small number are related to selectivity and skew estimation. [Chri83, Seli79, Sesh92, Sun93]. Statistical methods and probability play an important role in skew estimation. Statistical methods can be classified into parametric and nonparametric methods depending on how much is known about the shape of the distribution. If

there are only a few unknown parameters with a known basic distribution, the methods involved are referred to parametric methods. On the other hand, in many cases an experimenter does not know the form of the basic distribution and needs statistical techniques which are applicable regardless of the form of the density [Mura88].

Skew handling can be classified into static handling methods and dynamic handling methods. All methods have their limitations, assumptions, and usually targeted at specific situations. Moreover, most of the proposed algorithms involve considerable computation and communication overheads. Static skew handling aims at declustering the total workload evenly over parallel processors before operation processing begins, and static skew handling algorithms have been studied in [DeWi92b, Grae94, Hua91, Kits90, Omie91, Schn89, Wolf93a, Wolf93b]

Unlike static skew handling, dynamic skew handling attempts to either allocate workload dynamically during operation execution or reallocate the workload from the busy processor to others. The algorithms divide the work into parts which normally exceed the number of processors. The first k (the number of processors) parts will be sent to candidate processors, and others wait in a queue. If one of the processors finishes work early, it will be assigned another part from the queue. This process repeats until the queue is empty. The idea behind this method is partitioning the load and wait for skew to occur, and dynamic skew handling algorithms have been studied in [Kell91, Lu92].

Most of the existing skew handling algorithms cannot deal with situations in which there is no intrinsic data skew, and in general no consideration is given to determining the stochastic nature of skew behaviour. In this study we focus on skew estimation and we concentrate on skewness caused by data partitioning [DeWi92a] of unary relational operations. The skewness arising in binary operations is the subject of a separate study [Leun94c], and the parallelising of object-oriented databases is studied in [Leun94b]. It is felt that skew estimation should be emphasised along with skew handling, so that the workload of the system as a whole can be properly balanced and the system performance can be effectively optimised.

4. Skewness in Data Partitioning

The processing of a database query can generally be decomposed into three phases: data loading, processing, and consolidation. Data loading is concerned with the loading of data to the memory of a processor before processing can take place. The environment here is assumed to be a distributed memory configuration consisting of a number of heterogeneous processors. Processing consists of the manipulation of the data in memory so as to produce the desired result. Consolidation is concerned with the putting together of the results of processing and present them to the users in a form acceptable to them. For example, the removal of duplicate tuples resulting from projections is assumed to be done at consolidation. In this paper, we shall only concentrate on the processing phase as this is where the introduction of processor parallelism is most appropriate. The data loading and the consolidation phases usually require some inter-processor communication, while such communication may be avoided or kept to a minimum in the processing phase.

In the case of unary operations such as select, parallelism is effected primarily through *data partitioning*. Assuming there are k processors, then the single relation X could be distributed over these processors, so that the ith processor is responsible for processing x_i tuples allocated to it. In a perfectly balanced load allocation each $x_i = |X|/k$ of tuples, where $|X|$ is the cardinality of X. Figure 3 shows a situation where the relation X is partitioned into three parts and distributed to $k = 3$ processors. Under ideal allocation, an operation would yield an improvement by a factor of k, since the processing of the k subrelations could take place in parallel, and they all take the same amount of time to complete.

In this method, we are assuming that the data are fragmented by row - i.e. horizontal fragmentation - and the data distribution is based on splitting a single relation into k smaller relations. In fact, this method of splitting up a relation is easily accomplished since each tuple is usually stored as a record in external memory and forms a logical unit of transferable data.

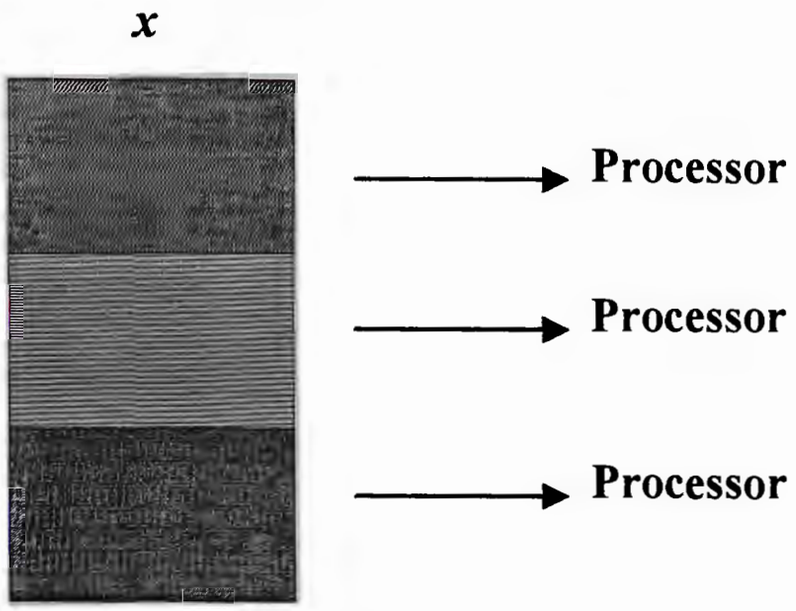


Fig. 3 Data Partitioning

A number of data partitioning methods are available [DeWi92a]. However, some degree of skewness is present in most of these methods. In particular, with range partitioning and hashing, there frequently exists some elements of randomness due to either the partitioning algorithm or the intrinsic data values or a combination of both factors. Such randomness will give rise to uneven distribution of data volumes, resulting in imbalanced workload. This will contribute to situations where some processors are heavily utilised, while others may be idle for a large proportion of the time. This will erode the benefits of parallel processing, and in extreme cases, may lead to a deterioration in performance as compared to uniprocessor processing [Leun92].

If we denote by N_i the random variable representing the number of X-tuples routed to processor P_i, then the execution time of the entire operation will be determined by the average of

$$N_{max} = \max(N_1, \ \ldots \ , N_k),$$

since the execution time is governed by the worker which has to process the largest number of X-tuples.

Thus, the average total execution time T is given by

$$T = wE(N_{\max}),$$

where w represents the amount of processing incurred per tuple. To determine $E(N_{\max})$, we proceed as follows. The process of routing X-tuples to different processors can be viewed as *partitioning* the integer x into a maximum of k parts. For a given partition σ of x into r parts

$$x = x_1 + \ldots + x_r,$$

with $x_i \geq 1$, and $r \leq k$, it corresponds to the following situation: irrespective of the identity among the k workers, one of them is assigned x_1 tuples, another worker is assigned x_2 tuples and so on, with *(k-r)* workers having no tuple assigned to them. From this consideration, it can be shown that [Leun94b] the average maximum loading is given by

$$E(N_{\max}) = \sum_{\sigma \in \prod_x} \frac{x!(k)_r}{x_1! \ldots x_r! r_1! \ldots r_{g(r)}!} \times \left[\frac{1}{k}\right]^x \times V_\sigma,$$

where $g(r)$ is the number of subgroups in the partition of x, obtained by grouping together all parts of the same magnitude, and r_i is the number of members in subgroup i, and

$$(k)_r = k(k-1) \ldots (k-r+1),$$

and

$$V_\sigma = \max(x_1, \ldots x_r) \quad ,$$

with Π_x representing the set of all partitions of x into at most k parts.

One of the main considerations in calculating $E(N_{\max})$ *is* the systematic enumeration of partitions of x. Algorithms for generating unrestricted partitions with no constraints on the number of parts may be found in [Page79]. However, in our situation, we need to partition x into no more than k parts. Here, we use the

convention that the parts are arranged in ascending order. For a partition into a fixed number of m parts, we start with m-1 parts of unity, followed by the last part of size x-m+1. Next, we generate new partitions having the same number of parts by successively equalising the rightmost two parts, with the result that these two parts differing by at most one. Such equalising is then extended leftwards to the (m-2)th part until the rightmost three parts differ by at most one. This equalising process of subtracting one from a part and adding it to a part on its left in generating new partitions continues while observing the constraint that no descending sequence is introduced within a partition. By systematically varying m, then all partitions of x into at most k parts may be obtained, and so $E(N_{\max})$ and hence T may be evaluated.

Although the above formula provides an exact evaluation of the execution time, it will entail considerable computational overhead. As an illustration, Appendix I shows the detailed calculation for k = 5, and x = 8, where the number of tuples is deliberately kept small to ensure manageability for illustration purposes. In this example, the average actual loading obtained is 3.18, while the average ideal loading is 8/5 = 1.6. In this case, the average actual loading is nearly twice the average ideal loading, which results in a severe over-estimation of performance.

For large k, however, it is possible to obtain closed-from approximations as follows. From [Trun88], it is shown that the variable

$$Z = \left[\ln(1+\frac{k}{x}) \right] N_{\max} - \ln k \ ,$$

has the following limiting cumulative distribution function

$$H(z) = \exp(-e^{-z}) \ .$$

Thus,

$$E(N_{\max}) \approx \frac{E(Z) + \ln k}{\ln\left(1+\frac{k}{x}\right)} .$$

The distribution $H(z)$ is known as the Gumbel distribution [Mood73]. Its expectation

is given by

$$E(Z) = \gamma,$$

where γ is Euler's constant. Thus, the approximation takes the form

$$E(N_{\max}) \approx \frac{\gamma + \ln k}{\ln(1+\frac{k}{x})}.$$

For $x >> k$, we have

$$\ln(1+\frac{k}{x}) \approx \frac{k}{x}.$$

The error in introducing this approximation, from [Gold64], is bounded by

$$\frac{1}{2}\left[\frac{k}{x}\right]^2,$$

which is less than 5×10^{-3} if $k < x/10$. Thus, the approximation will be correct to two decimal places whenever the tuple to processor ratio is more than 10. Therefore, we have

$$E(N_{\max}) \approx \frac{x}{k}(\gamma + \ln k). \qquad (1)$$

The average execution time is

$$T \approx \frac{xw}{k}(\gamma + \ln k). \qquad (2)$$

The additional multiplicative factor for this situation is

$$(\gamma + \ln k),$$

which may be regarded as the *skew factor*. A cruder approximation will be to drop

Euler's constant but this is unlikely to be acceptable. Even for MPP (massively parallel processing) systems with, say, $k = 2^{10}$, we have $\ln k = 6.93$; the constant 0.58 is clearly significant in relation to $\ln k$.

As for the processor with minimum loading

$$N_{\min} = \min(N_1, \ldots, N_k),$$

it is shown in [Trun88] that for large k and x, the variable

$$\frac{k^2 N_{\min}}{x + k}$$

asymptotically tends to the exponential distribution with unit mean, so that we have

$$E(N_{\min}) \approx \frac{x + k}{k^2},$$

which for $x >> k$, may be taken to be

$$E(N_{\min}) \approx \frac{x}{k^2}$$

The approximate average shortest execution time is thus

$$t \approx \frac{xw}{k^2}$$

The load imbalance is

$$I = \frac{T}{t} \cong k(\gamma + \ln k),$$

in the present situation. Since for sufficiently large k,

$$\gamma \leq \ln k ,$$

we have,

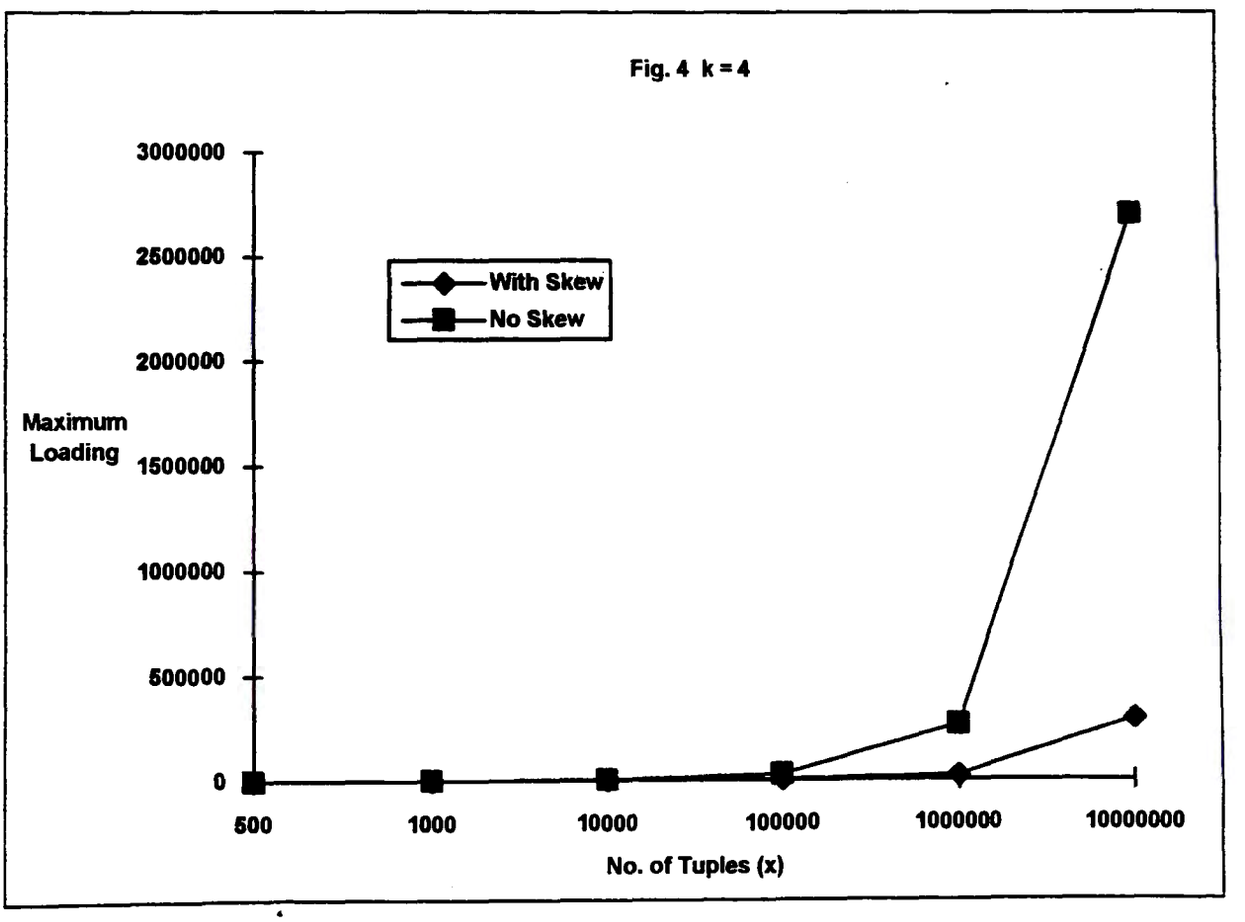
$$I \leq 2k \ln k \quad ,$$

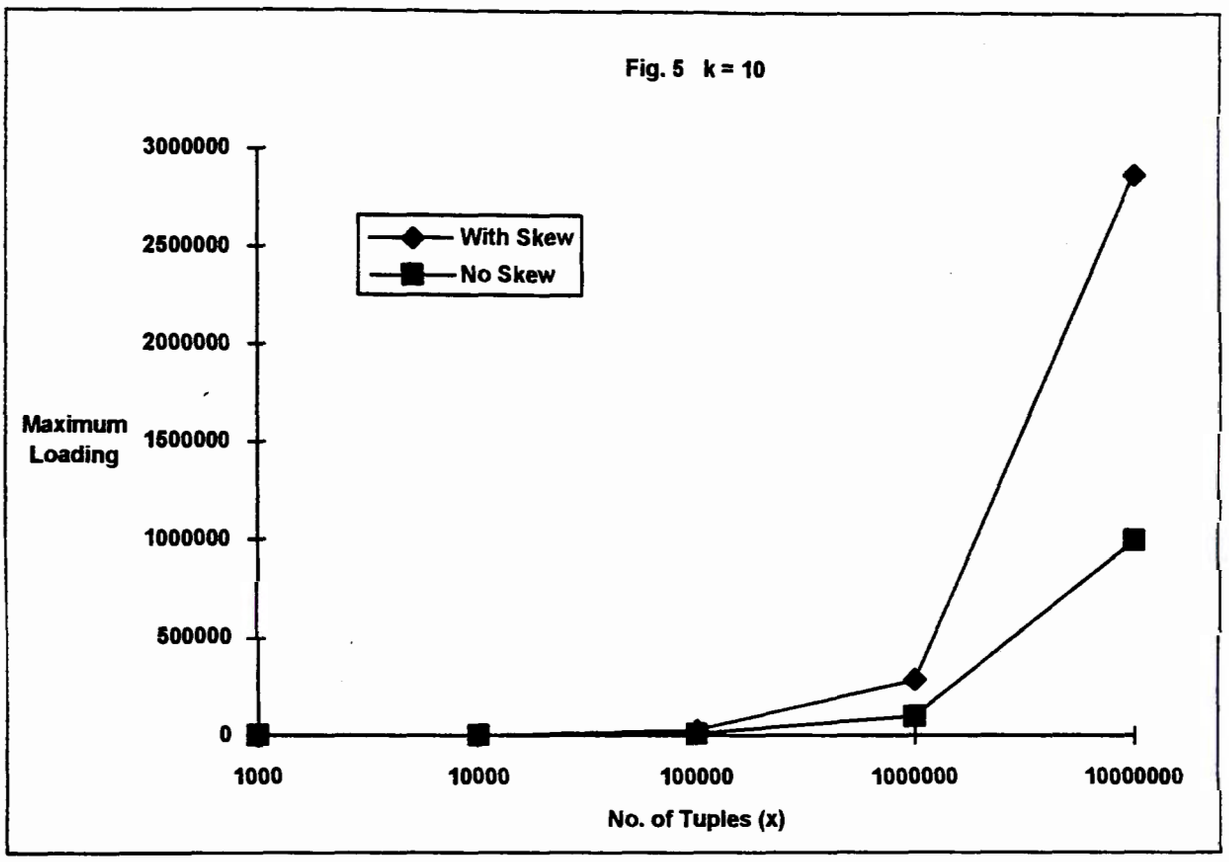
or $I = O(k \ln k)$. The processor utilisation ρ is given by the fraction of time within T that a processor is engaged in active processing. For a given processor in the collection, this is bounded by

$$\frac{t}{T} \leq \rho \leq 1 \ .$$

Using the previous relationship, this gives

$$\frac{1}{k(\gamma + \ln k)} \leq \rho \leq 1 \ .$$





Figures 4 and 5 compares the number of tuples allocated to the processors under ideal loading and actual loading where skew is present for $k = 4$, and $k = 10$ respectively. We observe that the under-estimation of performance by assuming perfectly balanced loading is again quite significant.

Summary and Conclusion

We have examined the effect of load skew on the performance of parallel unary relational operations based on the SIMD paradigm operating in a homogeneous distributed memory configuration. We see that the horizontal partitioning of a given relation often leads to uneven distribution of workload among the participating processors, and since the overall execution time is governed by that of the most heavily loaded processor, erosion of the performance benefits gained through parallel processing frequently results. This places a limit on the average speedup achievable

in unary parallel relational operations. This paper shows that in most data partitioning schemes, the actual performance can depart significantly from the ideal loading condition where each processor receives the same number of data tuples for processing. We also find that the number of processors increases, the load imbalance between the least utilised and the most utilised processors will tend to increase as $O(k \ln k)$. Closed-form expressions for quantifying the execution time, processor utilisation, and load imbalance caused by skewness are provided.

Acknowledgment

This work has been supported in part by the ESPRIT Parallel Computing Action Initiative under Project 4082.

References

[Chri83] Christodoulakis S, Estimating block transfers and join sizes, *Proceedings of the 1983 ACM SIGMOD Conference on the Management of Data*, San Jose, May 1983, pp40-54

[DeWi92a] DeWitt D.J., Gray, J. Parallel Database Systems. *Comm. ACM*, Vol. 35, No. 6, 1992, pp. 85-98.

[DeWi92b] DeWitt D.J., Naughton J.F., Schneider D.A., Seshadri S., Practical Skew Handling in Parallel Joins, *Proceedings of the 18th International Conference on Very Large Data Bases*, Vancouver, British Columbia, Canada 1992

[Gold64] Goldberg, *R. Methods of Real Analysis*, Blaisdell, 1964.

[Grae94] Graefe G., Sort-Merge-Join: An idea whose time Has(h) passed, *Proceedings the 10th International Conference on Data Engineering*, Feb. 14-18, 1994, Houston, Texas

[Hua91] Hua K. A., Lee C., Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning, In *Proceedings of the 17th International Conference on Very Large Data Bases*, Page 525-535, Barcelona, September, 1991

[Ioan93] Ioannidis Y. E., Christodoulakis S., Optimal Histograms for Limiting Worst-Case Error Propagation in the size of join Results, *ACM Transactions on Database Systems*, No.4, Dec 1993

[Kell91] Keller A. M., Roy S., Adaptive Parallel Hash Join in Main-Memory Databases, *Proceedings of the first International Conference on Parallel and Distributed Information Systems*, Dec 1991

[Kits90] Kitsuregawa M., Ogawa Y., A New Parallel Hash Join Method with Robustness for Data Skew in Super Database Computer (SDC), *Proceedings of the 16th International Conference on Very Large Data Bases*, 1990, pp. 210-221

[Leun85] Leung, C. H. C. and K. S. Wong, "File processing efficiency on the content addressable filestore," *Proc. 11th Int. Conf. on Very Large Data Bases*, Stockholm, August 1985, pp. 282-291.

[Leun92] Leung, C. H. C. and H. T. Ghogomu. Configuration optimisation in parallel database processing, *Proc. International Conference on Parallel Computing and Transputer Applications*, Barcelona, September 1992, in *Parallel Computing and Transputer Applications*, M. Valero, E. Onate, M. Jane, J.L. Larriba, and B. Suarez (Eds.), pp. 1239-1248, IOS Press/CIMNE, Barcelona 1992.

[Leun93] Leung, C. H. C. and H. T. Ghogomu A high-performance parallel database architecture, *Proc. 7th ACM International Conference on Supercomputing*, Tokyo, July 1993, pp. 377-386.

[Leun94a] Leung, C. H. C. and D. Taniar. Parallel query processing in object-oriented database systems. *Working Paper, Terabyte Database Group*, Victoria University of Technology, 1994. **(accepted for publication**, *Proc. 6th Australasian Database Conference* ADC '95)

[Leun94b] Leung, C. H. C. and K. Liu. Skewness analysis of parallel join execution. *Working Paper, Terabyte Database Group, Victoria University of Technology*, 1994.

[Litw92] Litwin, W. and T. Risch. Main memory oriented optimisation of OO queries

using typed Datalog with foreign predicates. *IEEE Trans. Data and Knowledge Eng.*, Vol. 4, No. 2, 1992, pp. 517-528.

[Lu92] Lu H. J., Tan K. L., Dynamic and Load-balanced Task-Oriented Database Query Processing in Parallel Systems, *Advances in Database Technology -- EDBT'92, 3rd International Conference on Extending Database Technology*, Vienna, Austria, March, 1992 Proceedings, Springer-Verlag

[Mood73] Mood, A. Graybill and D. Boas, *An Introduction to the Theory of Statistics*, McGraw-Hill, 1973.

[Mura88] Muralikrishna M., DeWitt D. J., Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries, *Proceedings SIGMOD International Conference on Management of Data*, Chicago Illinois, June 1988

[Moss92] Moss, J. Working with Persistent Objects: To Swizzle or not to Swizzle, *IEEE Trans. Software Eng.*, Vol. 18, 1992, pp. 657-673.

[Omie91] Omiecinski E., Performance Analysis of a Local Balancing relational hash-join algorithm for a main-memory databases, *Proceedings of the first International Conference on Parallel and Distributed Information Systems*, Dec 1991, pp58-67

[Ozka86] Ozkarahan, E.: *Database Machines and Database Management.* Prentice-Hall, Englewood Cliff, New Jersey, 1986.

[Page79] Page, E.S. and L. Wilson, *An Introduction to Computational Combinatorics*, Cambridge University Press, 1979.

[Schn89] Schneider D. A., DeWitt D. J., A Performance of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment, *Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data*, Portland, Oregon, Vol. 18, No. 2, June 1989, pp110-121

[Seli79] Selinger P.G., Astrashan M.M., Chamberlin D.D., Lorie R.A., Price T.G., Access path selection in a relational database management system, *Proceedings of*

the ACM SIGMOD International Conference on the Management of Data, Boston, Mass., June, 1979, pp23-34

[Sesh92] Seshadri S., Naughton J.F., Sampling Issues in Parallel Database Systems, *Advances in Database Technology – EDBT'92, 3rd International Conference on Extending Database Technology, Vienna, Austria, March, 1992 Proceedings*, Springer-Verlag

[Sun93] Sun W., Ling Y. B., Rishe N., and Deng Y., An Instant and Accurate Size Estimation Method for Joins and Selection in a Retrieval-Intensive Environment, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington D. C., May 1993.

[Trun88] Trunov, A. "Limit theorems in the problem of distributing identical particles in different cells," *Proc. Steklov Institute of Mathematics*, Issue 4, 1988, pp. 157-175.

[Wolf93a] Wolf J. L., Dias D. M., and Yu P. S., A Parallel Sort-Merge Join Algorithm for Managing Data Skew, *IEEE Transactions On Parallel And Distributed Systems*, Vol. 4, No. 1, January 1993

[Wolf93b] Wolf J. L., Yu P. S., Turek J. and Dias D. M., A Parallel Hash Join Algorithm for Managing Data Skew, *IEEE Transactions On Parallel and Distributed Systems*, Vol.4, No. 12. December 1993

APPENDIX I : Allocation of 8 Tuples to 5 Processors (Exact Calculation)

Partition	Occurrence probability	Maximum loading	Average maximum
8	5/390625	8	40/390625
1+7	160/390625	7	1120/390625
2+6	560/390625	6	3360/390625
3+5	1120/390625	5	5600/390625
4+4	700/390625	4	2800/390625
1+1+6	1680/390625	6	10080/390625
1+2+5	10080/390625	5	50400/390625
1+3+4	16800/390625	4	67200/390625
2+2+4	12600/390625	4	50400/390625
2+3+3	16800/390625	3	50400/390625
1+1+1+5	6720/390625	5	33600/390625
1+1+2+4	50400/390625	4	201600/390625
1+1+3+3	33600/390625	3	100800/390625
1+2+2+3	100800/390625	3	302400/390625
2+2+2+2	12600/390625	2	25200/390625
1+1+1+4	8400/390625	4	33600/390625
1+1+1+2+3	67200/390625	3	210600/390625
1+1+2+2+2	50400/390625	2	100800/390625
Sum	1	-	3.18