Analysis of a Parallel Grace Hash Join Implementation on the Cell Processor

Werner Mach
University of Vienna
Vienna, Austria
Email: werner.mach@univie.ac.at

Erich Schikuta
University of Vienna
Vienna, Austria
Email: erich.schikuta@univie.ac.at

Benedikt Pittl
University of Vienna
Vienna, Austria
Email: benedikt.pittl@univie.ac.at

POSTER PAPER

***Abstract*—The Cell Processor is a widely-used processor type embedded into a wide variety of technical appliances, ranging from television sets and gaming consoles to high performance computing servers. Its high availability and cost efficiency makes it very interesting for computing tasks as parts of scientific and business workflows.**

In this paper we present an analysis of the implementation of a variant of one of the most important and effort consuming data management operation, the data join operator. Due to the parallel execution features of a Cell processor we focus specifically on a parallelized implementation of the Grace Hash Join.

***Keywords*—cell processor; database; Grace hash join; performance analysis**

I. Introduction

In practice Cell processors have proven as a cost-effective replacement for classical supercomputers in many application domains. The Cell Broadband Engine Architecture is based on heterogeneous chip multiprocessing [14] [13]. This architecture supports scalar and single-instruction, multiple-data (SIMD) execution equally well. The Cell Broadband Engine allows a design with smaller cores and is a data-processing-oriented architecture. Enabling the processing power of the Cell BE's is using a data-parallel processing engine called *synergistic processor unit* (SPU). This SPU provides parallelism at all abstraction levels.

Cell-based system can deliver a suitable environment for parallel database systems. There is also an urgent need for novel database architectures due to new stimulating application domains with huge data sets to administer, search and analyze. A typical example are high energy physics applications (see [26]), which are based heavily on relational database technology.

Due to its inherent expressive power the most important operation in a relational database system is the join. It allows to combine information of different relations according to a user specified condition, which makes it the most demanding operation of the relational algebra. Thus the join is obviously the central point of research for performance engineering in database systems.

Generally two approaches for parallel or distributed execution of database operations can be distinguished, inter- and intra-operational parallelism. Inter-operator parallelism, resembling the rather simple, distributed approach, is achieved by executing distinctive operators of a query execution plan in a parallel or pipelined parallel fashion. In contrary the intra-operator parallelism is focusing on the parallel execution of a single operation among multiple processors by applying smart algorithms.

In the past a number of paper appeared covering this topic, like [21], [1], [7], [9], [10], [22]. [27] proposed and analyzed parallel database algorithms for parallel database machines. [2] presents an adaptive, load-balancing parallel join algorithm implemented on a cluster of workstations. The algorithm efficiently adapts to use additional join processors when necessary, while maintaining a balanced load. [18] develops a parallel hash-based join algorithm using shared-memory multiprocessor computers as nodes of a networked cluster. [28] uses a hash-based join algorithm to compare the designed cluster-system with commercial parallel systems. In [20] an analytical discussion of parallel algorithms for relational database operations in a grid environment are presented and compared to classical generalized multiprocessor framework

In this paper we will present an analysis and evaluation of the Cell processor for join operations. The goal is the analysis and discussion of an implementation of the parallel Hash Join on a commodity Playstation 3, which is a PowerPC based computer with a aligned Cell processing unit. We will give a speed-up analysis and compare it to other parallel implementations on this hardware.

The paper is structured as follows: The next section II describes a motivational scenario for our processor choice. In section III we introduce the database join operation and its special variant of the parallel Grace Hash Join. Section IV will describe the hardware and software setting of our test

978-1-4799-5313-4/14/$31.00

environment and the implementation of the algorithm. This is followed by a speed-up analysis of our experiment and a discussion of our findings. The paper is closed by a summary and statement of the findings.

II. Motivational Scenario for the Cell Based Engine Architecture

The Cell BE is a single-chip multiprocessor with nine processors on a shared coherent memory. Compared to typical multi-core processors, cell processors have a heterogenouse set of cores. It integrates a power processor element (PPE) and eight synergistic processor elements (SPEs). [12] See figure 1.

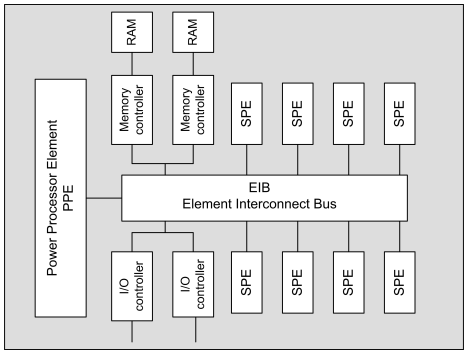

Figure 1. Cell system architecture [14]

The PPE is a 64-bit Power Architecture with vector media extensions. The PPE is optimized for common system functions whereas the SPEs perform optimized data-intensive processing. The SPEs are running independent application threads and are independent processors. Each SPE has a private local memory for efficient instruction and data access. This local store is limited to 256 KBytes and have influenced our decision using an Grace Hash Join Implementation. The SPEs have also full access to the coherent shared memory with memory-mapped I/O space [12].

The author are aware of the fact that the heyday of the Cell processor is over. However, the Cell processors was widely used both in commercial and academic communities. Therefore systems based on this processor type, for example Playstation 3s, are easily at hand. For example at the University of Vienna Cell processor systems were used for the introductory courses for parallel programming installed in quite large Cell processor labs.

In the course of the authors' research on parallel I/O [6], [11] and optimization of scientific workflows in heterogenous infrastructures [31], [25], [3] this available infrastructure delivered a feasible testbeds. Aspects of data service selection problems are studied in the context of the distributed, service-oriented system from ATLAS, a high-energy physics experiment at CERN, the European Organization for Nuclear Research [4], [29]. In the so-called TAG system, data and modular services, as (parallelized) relational database operations, are distributed world-wide and need to be selected and composed on the fly, as a user starts a request. Thus, in the context of this specific research endeavour the Cell processor has its value and delivers necessary computational resources.

III. Parallel Database Operations

A. Declustering

Declustering is the general method in a parallel database system to increase the bandwidth of the I/O operations by reading and writing the multiple disks in parallel. This denotes the distribution of the tuples (or records, i.e. the basic data unit) of a relation among two or more disk drives, according to one or more attributes and/or a distribution criteria.

Three basic declustering schemes can be distinguished:

- range declustering (contiguous parts of a data set are stored on the same disk)
- round-robin declustering (tuples are spread consecutively on the disk)
- hash-declustering (location of a tuple is defined by a hash function)

In this paper we denote R as the inner relation and S the outer relation of the performed Grace Hash Join. Where i is the declustering index of relation R. $i..N$ where N is the number of buckets. The disjoint property of the declustered sets R_i can be exploited by parallel algorithms based on the SPMD (single program, multiple data) paradigm. This means that multiple processors execute the same program, but each on a different set of tuples. In the database terminology this approach resembles the intra-operational parallelism.

B. The Join operation

Basically the join operation 'merges' two relations R and S via two attributes (or attribute sets) A or B (respective relations R and S) responding to a certain join condition. The join attributes have to have the same domain. Two different types of join operators are distinguished, the equi-join and the theta-join. In the following only the equi-join is discussed.

In the literature [8] three different approaches for join algorithms are distinguished:

- sort merge,
- nested loop, and
- hash based join.

Generally, the hash based join is the most efficient approach. Therefore we analysed the hash based join in section IV.

Basically the parallel versions of the these approaches are realized in a conventional client-server scheme. The server stores both relations to join and distributes the declustered tuples among the available clients. The clients perform the specific join algorithm on their sub relations and send the sub results back to the server. The server collects the result tuples and stores the result relation.

C. Grace Hash-Join

The Grace Hash Join have following important advantages for implementing on a Cell Based Architecture with limited memory:

- good if not more than $\sqrt{|R|}$ of memory blocks are available
- avoiding reading same blocks multiple (simple Hash-Join)

A well known representative of a hash join algorithm is the Grace join algorithm [19]. It works in three phases. In the first phase, the algorithm declusters relation R (inner relation) into N disk buckets by hashing on the join attribute of each tuple in R. In phase 2, relation S (outer relation) is declustered into N buckets using the same hash function. In the final phase, the algorithm joins the respective matching buckets from relation *R* and *S* and builds the result relation.

The number of buckets, N, is chosen to be very large. This reduces the chance that any bucket of relation R will exceed the memory capacity of the processor used to actually effect the join (the buckets of the outer relation may exceed the memory capacity because they are only used for probing the bucket of the inner relation and need not to fit in main memory). In the case that the buckets are much smaller than main memory, several will be combined during the third phase to form more optimal sized join buckets (referred to as bucket tuning in [19]).

The Grace algorithm differs from the sort merge and simple-hash join algorithms in that data partitioning occurs at two different stages – during bucket forming and during bucket-joining. Parallelizing the algorithm thus must address both of these data partitioning stages. To ensure maximum utilization of available I/O bandwidth during the bucket-joining stage, each bucket is partitioned across all available disk drives . A *partitioning split table* is used for this task. To join the ith bucket in R with the ith bucket of S, the tuples from the ith bucket in R are distributed to the available joining processors using a *joining split table* (which will contain one entry for each processor used to effect the join). As tuples arrive at a site they are stored in in-memory hash tables. Tuples from bucket I of relation S are then distributed using the same joining split table and as tuples arrive at a processor they probe the hash table for matches. The bucket-forming phase is completely separate from the joining phase under the Grace join algorithm. This separation of phases forces the Grace algorithm to write the buckets of both joining relations back to disk before beginning the join stage of the algorithm.

IV. Analysis and Discussion of the Implementation of a Grace Hash Join

As hardware platform a Playstation 3 was used which consists of central Power Processor Element (PPE), a 64 bit PowerPC with VMX, and eight coprocessing elements (Synergistic Processing Elements, SPEs), which are connected by an Element Interconnect Bus (EIB). In the Cell processor used in the Playstation two of the eight SPEs (Spe 7 and SPE 8) are due to fabrication cost reasons not available.

For the implementation of the Join algorithm we used the cell-sdk for the SPE processors with threads.

The implementation of the Grace Hash Join was based on the algorithm 1:

```
Data: relations R and S
Result: joined tuples of R and S
for each tuple r in R relation do
    apply hash function to the join attribute of r;
    put r into the appropriate bucket R_i;
end
for each tuple s in S relation do
    apply hash function to the join attribute of s;
    put s into the appropriate bucket S_i;
end
for each bucket i do in PARALLEL do
    build the hash table from R_i;
    for each tuple s in S_i do
        apply hash function to the join attributes of s;
        use s to probe the hash table;
        output any matches to the result relation;
    end
end
```

Algorithm 1: Parallel Grace Hash Join algorithm

The algorithm can be easily parallelized, because the generated buckets 1..n are disjoint. We spread the tasks of this algorithms onto the PPE and the available SPEs. The hashing of the tuples of relation R and S to determine the buckets was executed on the PPE, the parallel execution of finding matches between bucket tuples was distributed to the SPEs 1..6.

We executed the program for test sets of different sizes of the data set. One tiny data set size of 1048558 bytes with 30142 tuples and 262 buckets with bucket size of 4000 bytes. And a huge data set with size of 50901150 bytes, 1154961 tuples and bucket size 4000 bytes.

We also implemented two versions of the algorithm:

- *Materialized Version*: the hashed buckets were written to the disk by the PPE and read again form disks by the SPEs.
- *Pipelined Version*: the hashed buckets were transferred in memory from PPE to SPEs via DMA transfer.

However, due to the limitation of the bus the performance results for both version showed similar behaviour, which is presented in figure 2.

The speedup analysis is given in figure 3. It can be seen that using more than 3 SPEs the speedup drops significantly.

To compare the result we have also implemented a disk independent parallel algorithm, which is just doing some

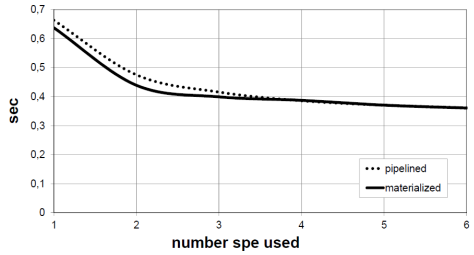

Figure 2. Performance of Grace Hash Join for Materialized and Pipelined Version

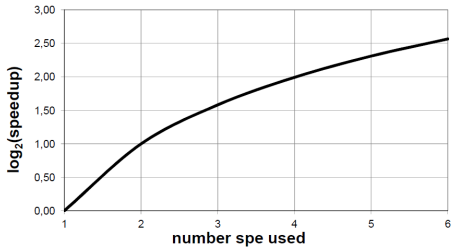

Figure 3. Speedup of Grace Hash Join for Materialized and Pipelined Version

floating point calculations only. We execute 95535000 floating point operations on one SPE, two SPE's in parallel, three SPE's an so on. The performance characteristics of this "data-less" algorithm is given in figure 4 and 5. Here a near linear speedup behavior can be identified as expected.

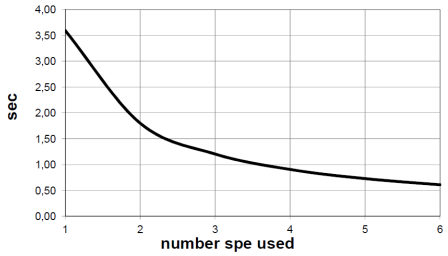

Figure 4. Performance of "data-less" floating point algorithm

So obviously big data set processing is limited by the Cell architecture, specifically by the 16kByte DMA transfer limit and the 256kByte limit of main memory per SPU. We also tried to replace the physical disk by a ram-disk, but even this "trick" did not solve the problem.

Even though that we could not reach linear speedup for data intensive tasks but at least a remarkable performance improvement could be achieved. It is to be noted that no complicated handcrafted parallelized code has been developed.

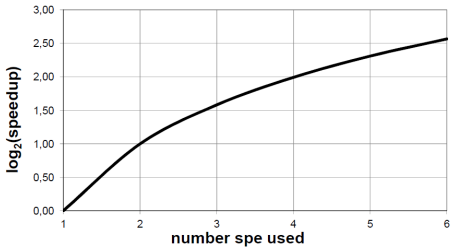

Figure 5. Speedup of "data-less" floating point algorithm

Only a quite simple declustered data set approach allowed to exploit the parallel features of the Cell processor, which makes it highly interesting for performance heavy tasks.

V. Conclusion and Future Work

We presented an analysis of a parallel implementation of the Grace Hash Join operation on a Cell processor. In this endeavor we used a simple declustered data set parallelization approach omitting expensive hand crafting of parallel code. It turned out the an acceptable speedup behaviour could be noticed. However linear speedup was out of scope of this data intensive application due to hardware limitations. But, summing up and simply stated: a cheap implementation on cheap hardware produced remarkable results.

The presented work complements our work in the area of computational science and computational intelligence, where we aim for a virtual organisation delivering ample resources, both hardware and software and knowledge to the scientific community. In course of this research we developed the N2Sky system [23], realizing neural networks as a service paradigm, which fosters cloud resources delivering a framework for the Computational Intelligence community to share and exchange neural network resources. Our research focus lies on the parallelization mechanism which transparently delivers available high performance computing resources, as clusters, multi core architectures and GPGPUs, and cell processors utilizing parallel processing schemes. N2Sky is a prototype system with quite some room for further enhancement. We are working on an enhancement of the neural network paradigm description language ViNNSL [5] to allow for automatic sharing of resources based on the formalized description. Key for fostering of cloud resources are service level agreements (SLAs) which give guarantees on quality of the delivered services. We are working on the embedment of our research findings on SLAs [16], [17], [15] into N2Sky to allow for novel business models [34], [24], [33], [32] on the selection and usage of neural network resources based on quality of service attributes [30]. A further important issue is to find neural network solvers for given problems, similar to a "Neural Network Google". In the course of this research we are

using ontology alignment by mapping problem ontology onto solution ontology.

REFERENCES

[1] P. America. Parallel database systems. In *Proc. PRISMA Workshop*. Springer Verlag, 1990.

[2] Minesh B. Amin, Donovan A. Schneider, and Vineet Singh. An adaptive, load balancing parallel join algorithm. In *Sixth International Conference on Management of Data (COMAD'94)*, Bangalore, India, 1994.

[3] Peter Paul Beran, Werner Mach, Erich Schikuta, and Ralph Vigne. A multi-staged blackboard query optimization framework for world-spanning distributed database resources. In *International Conference on Computational Science (ICCS 2011)*, Procedia Computer Science series, Singapore, June 2011. Elsevier Science.

[4] Peter Paul Beran, Elisabeth Vinek, Erich Schikuta, and Maria Leitner. An adaptive heuristic approach to service selection problems in dynamic distributed systems. In *13th ACM/IEEE International Conference on Grid Computing (Grid 2012)*, pages 66–75, Beijing, China, 2012. IEEE.

[5] Peter Paul Beran, Elisabeth Vinek, Erich Schikuta, and Thomas Weishäupl. ViNNSL - the Vienna Neural Network Specification Language. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2008, part of the IEEE World Congress on Computational Intelligence, WCCI 2008*, pages 1872–1879. IEEE, June 2008.

[6] Peter Brezany, Thomas Mück, and Erich Schikuta. A software architecture for massively parallel input-output. In Jerzy Wasniewski, Jack Dongarra, Kaj Madsen, and Dorte Olesen, editors, *Third International Workshop Applied Parallel Computing Industrial Computation and Optimization (PARA'96)*, volume 1184 of *Lecture Notes in Computer Science*, page 85–96, Lyngby, Denmark, 1996. Springer Berlin / Heidelberg. 10.1007/3-540-62095-8_10.

[7] G. Copeland, W. Alexander, E. Boughter, and T. Keller. Data placement in bubba. In *Proc. Of the ACM-SIGMOD Int. Conf. On Management of Data*. ACM, 1988.

[8] C. Date. *An Introduction to Database Systems*. Addison-Wesley, 1986.

[9] D.J. DeWitt, R.H. Gerber, G. Graefe, M.L. Heytens, K.B. Kumar, and M. Muralikrishna. GAMMA- a high performance dataflow database machine. In *Proc. Of the 12th Conf. On Very Large Data Bases*, 1986.

[10] D.J. DeWitt and J. Gray. Parallel database systems: The future of database processing or a passing fad? *ACM-SIGMOD record*, 19(4), 1990.

[11] Thomas Fürle, Oliver Jorns, Erich Schikuta, and Helmut Wanek. Meta-ViPIOS: harness distributed I/O resources with ViPIOS. *Iberoamerican Journal of Research "Computing and Systems", Special Issue on Parallel Computing*, 4(2):124–142, 2000.

[12] M. Gschwind, H.P. Hofstee, B. Flachs, M. Hopkin, Y. Watanabe, and T. Yamazaki. Synergistic processing in cell's multicore architecture. *Micro, IEEE*, 26(2):10–24, March 2006.

[13] Michael Gschwind. Chip multiprocessing and the cell broadband engine. In *Proceedings of the 3rd Conference on Computing Frontiers*, CF '06, pages 1–8, New York, NY, USA, 2006. ACM.

[14] Michael Gschwind, H. Peter Hofstee, Brian Flachs, Martin Hopkins, Yukio Watanabe, and Takeshi Yamazaki. Synergistic processing in cell's multicore architecture. *IEEE Micro*, 26(2):10–24, March 2006.

[15] Irfan Ul Haq, Rehab Alnemr, Adrian Paschke, Erich Schikuta, Harold Boley, and Christoph Meinel. Distributed trust management for validating sla choreographies. In Philipp Wieder, Ramin Yahyapour, and Wolfgang Ziegler, editors, *Grids and Service-Oriented Architectures for Service Level Agreements*, pages 45–55. Springer US, 2010.

[16] Irfan Ul Haq, Ivona Brandic, and Erich Schikuta. SLA validation in layered cloud infrastructures. In *Economics of Grids, Clouds, Systems, and Services, 7th International Workshop, GECON'10*, volume 6296 of *Lecture Notes in Computer Science*, page 153–164, Ischia, Italy, 2010. Springer Berlin / Heidelberg.

[17] Irfan Ul Haq and Erich Schikuta. Aggregation patterns of service level agreements. In *Frontiers of Information Technology (FIT'10)*, Islamabad, Pakistan, 2010. ACM.

[18] Y. Jiang and A. Makinouchi. A parallel hash-based join algorithm for a networked cluster of multiprocessor nodes. In *Proceedings of the COMPSAC '97 - 21st International Computer Software and Applications Conference*, 1997.

[19] M. Kitsuregawa, H. Tanaka, and T. Moto-Oka. Application of hash to data base machine and its architecture. *New Generation Computing*, 1(1), 1983.

[20] Werner Mach and Erich Schikuta. Parallel algorithms for the execution of relational database operations revisited on grids. *International Journal of High Performance Computing Applications*, 23(2):152–170, 2009.

[21] M. H. Nodine and J. S. Vitter. Optimal deterministic sorting on parallel disks. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '93)*, pages 120–129, Velen, Germany, 1993.

[22] H. Pirahesh, C. Mohan, J. Cheng, T.S. Liu, and P. Selinger. Parallelism in relational database systems: Architectural issues and design approaches. In *Proc. Of the IEEE Conf. On Distributed and Parallel Database Systems*. IEEE Computer Society Press, 1990.

[23] E. Schikuta and E. Mann. N2sky - neural networks as services in the clouds. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–8, Aug 2013.

[24] Erich Schikuta, Flavia Donno, Heinz Stockinger, Helmut Wanek, Thomas Weishäupl, Elisabeth Vinek, and Christoph Witzany. Business in the grid: Project results. In *1st Austrian Grid Symposium*, Hagenberg, Austria, 2005. OCG.

[25] Erich Schikuta, Helmut Wanek, and Irfan Ul Haq. Grid workflow optimization regarding dynamically changing resources and conditions. *Concurrency and Computation: Practice and Experience*, 20(15):1837–1849, 2008.

[26] Ben Segal. Grid computing: The european data project. In *IEEE Nuclear Science Symposium and Medical Imaging Conference*, Lyon, France, 2000. IEEE Computer Society Press.

[27] M. Stonebraker, P.M. Aoki, R. Devine, W. Litwin, and M. Olson. Mariposa: A new architecture for distributed data. In *Proc. Of the Int. Conf. On Data Engineering*. IEEE Computer Society Press, 1994.

[28] Takayuki Tamura, Masato Oguchi, and Masaru Kitsuregawa. Parallel database processing on a 100 node PC cluster. In *Proc. of the Supercomputing 97*. IEEE Computer Society Press, 1997.

[29] Ralph Vigne, Erich Schikuta, V. Garonne, G. Stewart, M. Barisits, T. Beermann, M. Lassnig, C. Serfon, L. Goosens, and A. Nairz. Atlas ddm workload emulation. In *20th International Conference on Computing in High Energy and Nuclear Physics (CHEP2013)*, October 2013.

[30] Elisabeth Vinek, Peter Paul Beran, and Erich Schikuta. Classification and composition of qos attributes in distributed, heterogeneous systems. In *11th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid 2011)*, Newport Beach, CA, USA, May 2011. IEEE Computer Society Press.

[31] Helmut Wanek and Erich Schikuta. Using blackboards to optimize grid workflows with respect to quality constraints. In *Fifth International Conference on Grid and Cooperative Computing Workshops (GCC'06)*, volume 0, page 290–295, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[32] Thomas Weishäupl, Flavia Donno, Erich Schikuta, Heinz Stockinger, and Helmut Wanek. Business in the grid: BIG project. In *Grid Economics & Business Models (GECON 2005) of Global Grid Forum*, volume 13, Seoul, Korea, 2005. GGF.

[33] Thomas Weishäupl and Erich Schikuta. Towards the merger of grid and economy. In *International Workshop on Agents and Autonomic Computing and Grid Enabled Virtual Organizations (AAC-GEVO04) at the 3rd International Conference on Grid and Cooperative Computing (GCC04)*, volume 3252/2004 of *Lecture Notes in Computer Science*, page 563–570, Wuhan, China, 2004. Springer Berlin / Heidelberg.

[34] Thomas Weishäupl, Christoph Witzany, and Erich Schikuta. gSET: trust management and secure accounting for business in the grid. In *6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'06)*, page 349–356, Singapore, 2006. IEEE Computer Society.