Dynamic Prefetching Using a Distance Prefetcher (G/DC) to Improve Cache Utilisation

Martin Rebne Farstad, Jarand Kennedy

***Abstract*—With the increasing gap between processor speeds and memory access latency, highly efficient prefetchers have become imperative for improving cache memory utilisation. This paper investigates and tests a global delta correlation (G/DC) prefetcher and how it compares to other prefetching methods. A modified version of the M5 hardware simulator is used to simulate the prefetcher implementation, and the results are evaluated using the SPEC CPU2000 benchmark suite. The G/DC method is capable of recognising complex memory access patterns, and this paper shows an implementation capable of increasing application performance significantly with limited hardware overhead.**

I. Introduction

With the first commercial release of the DRAM chip by Intel in 1970 and its continued usage throughout the decades since it has become apparent to researchers that the main memory bandwidth increase dwarfs the CPU speed. This phenomenon, shown in Fig. 1, is called the "Memory Wall" [1]. It implies that the memory access delay, not the processor speed, becomes the major performance bottleneck in modern high-performance computing. The instructions of an average program reference memory between 20% and 40% of the time [2]. Modern applications are increasingly more memory-intensive [3].

Developing architectures with high-performance multi-level memory cache hierarchies was an effort to compensate for this bottleneck. These cache memories rely on the principle of locality in memory accesses. Limited cache size and data locality was the motivation behind initial prefetch algorithms to fetch instructions and data into the cache, predicting that the CPU would access these locations shortly after. By not taking advantage of run-time information about memory access patterns, these static algorithms suffered low effectiveness when applications' memory access patterns change [3].

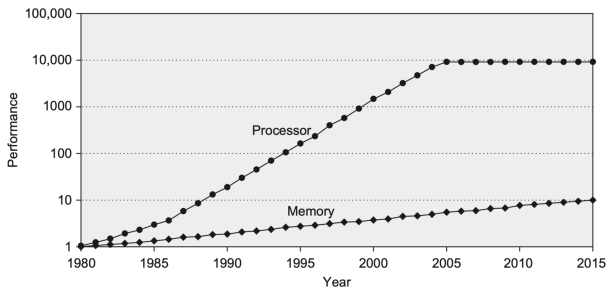

Fig. 1. The performance gap between processor and memory that constitutes the *Memory Wall*. From [4].

The paper discusses the viability of a hardware prefetcher that use a *global history buffer* to maintain memory access history. Storing dynamic run-time memory access history in the buffer can predict complex memory access patterns. This paper studies the implementation of a *Distance prefetcher*, a correlation-based prefetcher that uses this buffer [5]. This paper discusses performance obtained from implementing this prefetcher as a trade-off to the size and complexity of the hardware required.

The rest of this paper is structured as follows. Section II covers the necessary material to understand the proposed implementation, while section III includes more comprehensive implementation details. Section IV provides details regarding the simulator and benchmark suite used to implement and test the prefetcher, respectively. Section V presents and discusses the prefetcher's benchmark results, and section VI elaborates a discussion regarding implementation parameters. Section VII discusses similar alternatives to the proposed solution, as well as why the solution proposed in this paper is different. Lastly, section VIII concludes the discussions in this paper.

II. Background

Cache data prefetching is an effective solution to increase memory access performance and diminish the increasing gap between memory access bandwidth and CPU speed. Initial static prefetchers increased application performance based on memory access locality. Two such implementations are adjacent and strided prefetchers. For every new memory access, the prefetcher fetches data based on spatial locality with a constant distance between succeeding prefetches [6]. The static nature of the locality-based prefetchers raises an issue for applications with more complex memory access patterns. These applications have more correlations between the memory accesses, and therefore need a more sophisticated algorithm to be able to prefetch correctly. One data structure that tries to handle this increasing complexity is the global history buffer.

A. Global history buffer

Nesby and Smith originally proposed the global history buffer (GHB) to maintain the history of recent cache memory access misses. "Global History Buffer prefetching can increase correlation prefetching performance by 20% and cut its memory traffic by 90%" [5].

The data structure holds the addresses of the most recent cache memory access misses in First In First Out (FIFO) order, with each entry having a pointer to a previous entry having some common property (if it exists). As a consequence, the GHB entries represent the global cache miss address stream.

Traversing the pointers creates a linked list illustrated with curved arrows in Fig. 2. Every entry adjacent to previous entries represents their succeeding cache miss. With these pointers, any number of history-based prefetch algorithms can be implemented [5]. New entries are appended to the bottom of the table, and old entries are removed from the top when the number of entries exceeds the buffer size. Larger GHB sizes allow more memory access history but are a trade-off for increased memory and chip space required. However, trading chip space for lower data-access latency is a current trend [3].

B. Index table

When inserting new elements into the GHB, a search for the most recent previous entry occurs. The need for fast access to previous occurrences motivates the implementation of an index table [3]. This table, shown to the left in Fig. 2, stores a key and a pointer to the previous GHB entry associated with the same key. As a result, when inserting a new entry into the GHB, a lookup into the table using a hash function on the key provides fast access to the previous entry. This index table can be implemented with any associativity, such as 2-way or 4-way, to prevent conflict misses [7].

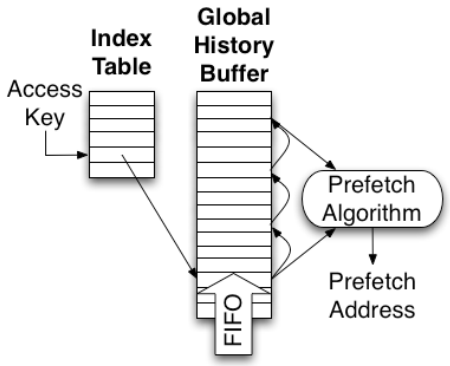

Fig. 2. An overview of the index table and global history buffer data structure. The curved arrows between GHB entries represent pointers to previous entries having some common property. From [5].

III. Implementation

Initially proposed for prefetching TLB entries, the Distance prefetcher is a correlation-based prefetcher that introduces the concept of a *delta*, the difference between two subsequent cache miss addresses [8]. This paper's implementation considers delta correlations between addresses in the global cache miss address stream (G/DC).

The Distance prefetcher models the global delta stream as the *Markov graph* shown to the right in Fig. 3, where nodes represent deltas and arcs represent the probability of a target delta being prefetched immediately after the source delta. The correlation table contains rows representing these relationships, as shown to the left in Fig. 3. The tag field contains the source node delta, and the rest of the row represent adjacent target node addresses with the highest probabilities [5]. The implementation of the correlation table uses an index table, and a GHB discussed in the previous section. On each

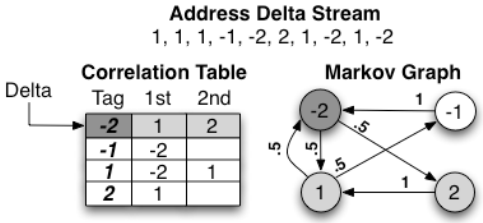

Fig. 3. To the left: A correlation table, showing subsequent deltas appearing after a given node. A decisive part of the Distance prefetcher implementation. To the right: The Markov graph, containing source and target nodes representing deltas, and arcs, representing the probability of a target delta being prefetched immediately after the source delta. From [5].

cache access miss, the prefetcher calculates the delta between the head of the GHB and the current cache miss address. A lookup into the index table gives a pointer to a previous entry with the same delta (if it exists). Calculating deltas to adjacent entries while traversing the linked list of previous entries in the GHB constitute each row of the correlation table. Each address in the row is then prefetched by first adding the delta to the current cache miss address. The most recent deltas are considered first for prefetching.

The prefetcher contains an additional feature to handle the case when the index table does not contain a pointer to the previous entry when accessed. Instead of remaining idle, the prefetcher issues a number prefetches for blocks adjacent to the cache miss address, limited by the *prefetching degree*. The prefetching degree is the maximum number of prefetched candidates in response to a single prefetch request [5].

Adding width, depth or a hybrid of the two adjust the prefetcher algorithm. Width limits the maximum number of previous entries visited, illustrated by the curved arrows in Fig. 4, and favours recent behaviour when deciding prefetch candidates. Depth increase the number of adjacent deltas visited, shown with lighter grey boxes in Fig. 4, and allows the prefetcher to run farther ahead of the actual address stream. The width-to-depth ratio is a trade-off for the prefetcher *timeliness*, i.e. whether prefetches are issued early enough to prevent processor stalls [5]. The results section scrutinise this trade-off and shows the results from the different versions.

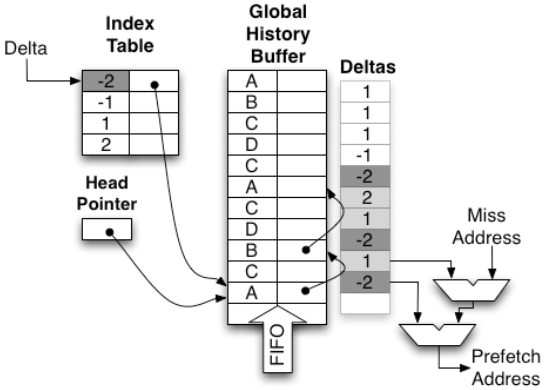

Fig. 4. A more detailed view of the index table and GHB used to implement the correlation table in Fig. 3. The deltas shown in the figure does not exist in GHB hardware. It is calculated as the difference between adjacent entries. From [5].

IV. Methodology

A. Simulator

A modified version of the open-source M5 hardware simulator was used to simulate the Distance prefetcher implementation in hardware. The simulator presents a simple interface for implementing prefetchers for M5's L2 cache. A subset of the interface is issued during system initialisation, for each cache access (hit or miss) and on a successfully prefetched block. The interface contains functions for prefetching data and getting information about the contents of the cache and prefetch queue [9].

The simulator has the specifications listed in table I and is based on the Alpha 21264 microprocessor, a part of the DEC Alpha Tsunami system. It is a superscalar, out-of-order CPU that is capable of reordering a large number of instructions and perform speculative execution [9].

TABLE I
THE SPECIFICATIONS OF THE M5 SIMULATOR [9].

L1 instruction cache	32 KiB
L1 data cache	64 KiB
L2 cache size	1 MiB
Cache block size	64 B
Memory bus frequency	400 MHz
Memory bus width	64 bits
Memory bus latency	30 ns

B. Benchmark suite

The prefetcher performance is evaluated by using the SPEC CPU2000 benchmark suite, which evaluates the performance of the CPU, memory and compiler on the tested system. The benchmark consists of multiple tests within two sub-components, *CINT2000* and *CFP2000*, which measure and compare compute-intensive integer (int) and floating-point (fp) performance, respectively [10]. The tests are based on real-world applications and shown in table II.

TABLE II
TESTS INCLUDED IN THE SPEC CPU2000 BENCHMARK SUITE [10].

Name	Type	Description
twolf	int	Place and route simulator
bzip2	int	Data compression utility
swim	fp	Shallow water modeling
ammp	fp	Computational chemistry
art	fp	Neural network simulation
wupwise	fp	Quantum chromodynamics
appsi	fp	Environment computation
applu	fp	Partial differential equations
galgel	fp	Fluid dynamics

The benchmark output consists of the number of cache accesses identified, issued and missed. It also includes Instructions Per Cycle (IPC), speedup, accuracy and coverage. The equations (1), (2) and (3) define the last three outputs, where a good prefetch is defined as a block being referenced by the application before it is replaced [9].

$$\text{speedup} = \frac{\text{IPC}_{\text{with prefetcher}}}{\text{IPC}_{\text{without prefetcher}}} \tag{1}$$

$$\text{accuracy} = \frac{\text{good prefetches}}{\text{total prefetches}} \tag{2}$$

$$\text{coverage} = \frac{\text{good prefetches}}{\text{cache misses without prefetching}} \tag{3}$$

The simulator output compares the Distance prefetcher results with other prefetchers having identical specifications. Each output is the harmonic mean speedup of the benchmark tests. The formula of this calculation is shown in equation (4).

$$H_{avg} = \frac{n}{\sum \frac{1}{x_i}} \tag{4}$$

V. Results

The graphs cover three versions of the Distance prefetcher. Two versions considering only width or depth, and a third hybrid version combining the parameters. All versions prefetch adjacent entries up to the prefetch degree when a previous entry is not found. The prefetch degree has identical value as the width or height. The results are evaluated by looking at the average performance increase (speedup) over all the tests with different data structure sizes. Afterwards, the performance of the individual tests in the benchmark suite for the best-performing prefetcher is scrutinised.

A. Average test performance

Fig. 5 shows the output after running the M5 simulator with different GHB and index table sizes on a logarithmic scale. The output consists of the harmonic mean of the speedups generated from running each test in the benchmark suite. The speedups are calculated relative to running the simulator without a prefetcher.

The output shows that the width prefetcher is most efficient overall. The width-to-depth ratio favours the width, which implies that recent behaviour is appreciated and that prefetcher timeliness is not an issue. Running farther ahead of the actual address stream with depth does not seem to be favoured by the benchmark tests.

The figure clearly shows the size needed to achieve a significant performance improvement from implementing the Distance prefetcher. Although the maximum speedup of ~1.04 can be achieved with a GHB and index table size of 1024 or 2048, keeping in mind the physical hardware constraints should desire a smaller size. As a result, a GHB and index table size of 256 gives a significant speedup of 1.03. Reducing the size to half severely reduces performance and results in a performance loss with a speedup of 0.98. Increasing the size gives no significant improved speedup before reaching four times the size. Additionally, changing the index table size relative to the GHB size gives in general worse performance versus overhead trade-off overall, and therefore, will not be presented here.

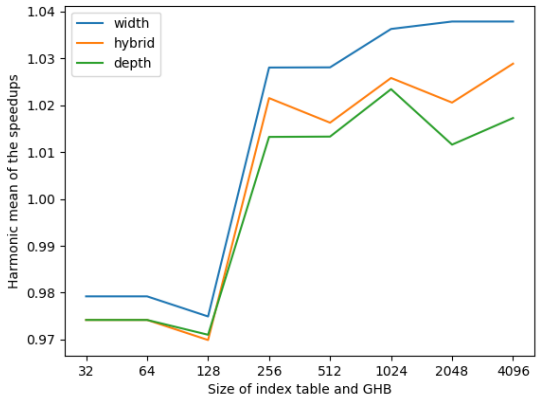

Fig. 5. Harmonic mean of test speedups plotted against index table and GHB sizes on a logarithmic scale. The graph includes the three versions of the Distance prefetcher introduced in section III: width, depth and hybrid.

B. Individual test performances

Fig. 6 shows the individual test results using a GHB and index table size of 256. The figure display uneven speedup results, heavily favouring specific tests with performance increases ranging between 9%-14%. The results imply that the favoured tests have delta correlations in the global miss stream that can be exploited for prefetching. The distance prefetcher should be a great candidate for applications having similar computations like wupwise, applu and galgel.

However, the neural network simulation tests, art470 and art110, obtain a performance decrease. The results imply that the Distance prefetcher does not give speedup increases for all types of applications, and its implementation for general use should be considered carefully.

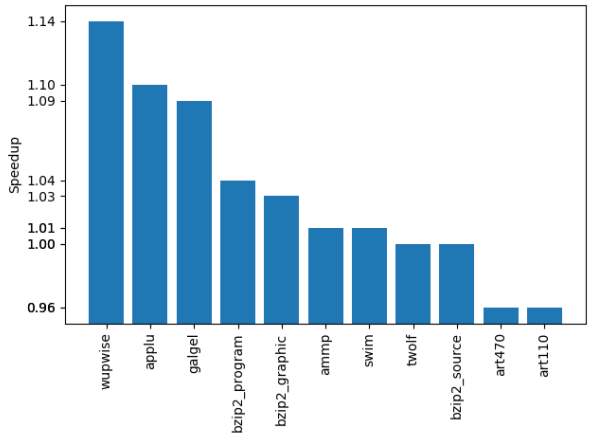

Fig. 6. Speedup per SPEC benchmark for the width implementation having an index table and GHB size of 256

The output of the SPEC benchmarks gives a notion of why the Distance prefetcher underperforms in some tests. Fig. 7 shows that RPT, DCPT and DCPT-P are the only prefetchers having increased speedup. The DCPT and DCPT-P combine RPT and PC/DC prefetching approaches [11]. The difference in speedup between DCPT/DCPT-P and RPT is not the most substantial. Therefore, this implies that the RPT prefetching approach has the most significant impact on the improved speedup. The RPT prefetcher achieves the best performance on applications having strided memory access patterns [6]. Therefore, these results imply that the Distance prefetcher does not recognise strided memory access patterns efficiently.

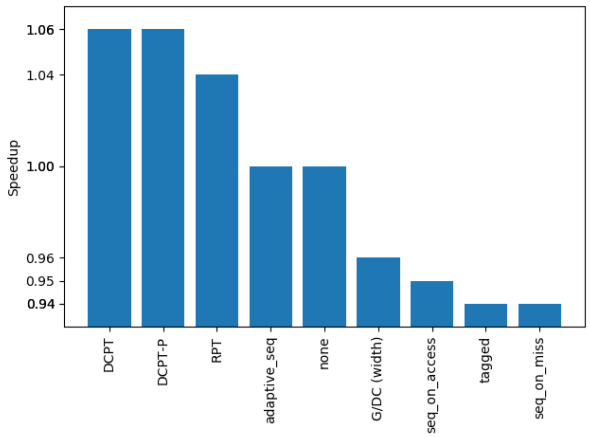

Fig. 7. Benchmark test output for the art110 and art470 benchmark tests, running neural network simulations.

VI. Discussion

The results section presented the Distance prefetcher with an index table and GHB size of 256 as the best implementation concerning hardware overhead. According to Falsafi and Wenisch, this size for the index table and GHB constitute little hardware overhead [12]. Combining this adequate size with the fact that the index table and GHB structures are relatively simple, the proposed prefetcher implementation should be applicable in a real processor.

What about achieving even higher speedups? Can doubling the index table and GHB size two times to reach a harmonic mean speedup of 1.04 be justified? The research implied that the uneven pattern of the different tests continued for increased sizes and that the tests mostly only achieved a minor speedup increase. Therefore, the Distance prefetcher does not give performance gains to all kinds of applications, even though the size of the data structures increases. Additionally, the processor's limited physical space contradict this justification.

The three different versions of the Distance prefetcher had predetermined values for width, height and degree. The parameters were chosen based on multiple performed simulations, and the candidates were chosen based on the average performances of all sizes. One could argue that the graph would have been more insightful had there been more versions with different parameters. Exploring the perfect combination of these parameters given each size of the data structures would have been optimal. It could alter the conclusions drawn in this paper. The fact that the width prefetcher is the best-performing implementation contradicts the results by Nesbit and Smith [5].

The results section did not describe any associativity of the index table. However, associativity was included in one of the initial implementations but got removed due to bad performance. The performance might have been inadequate due to the hash function not spreading the entries uniformly.

The hash function consisted of a simple modulo operator. One could argue that associativity is essential to reduce the number of conflict misses. Given our hash function, it seemed like the overhead of the added functionality exceeded the performance gains. As a result, associativity did not make any graphs.

VII. Related work

The static nature of hardware prefetchers is a concern for modern memory-intensive applications having complex correlations in memory access patterns. Correlation-based prefetchers try to overcome this arising issue. However, the Distance prefetcher discussed in this paper is only one of several correlation-based hardware prefetchers.

A. Markov prefetching (Global/Address Correlation)

Initially proposed by Joseph and Grunwald, the Markov prefetcher is very similar to the Distance prefetcher discussed in this paper [13]. It considers address correlations in the global miss address stream. The Distance prefetcher is a more generalised version of the Markov prefetcher [5].

The Markov prefetcher implementation also uses an index table and a GHB. The index table is indexed by the memory access miss addresses instead of deltas. As a result, the prefetcher has a more accurate description of the memory access pattern. Each entry in the GHB has a pointer to a previous occurrence of the same memory address. By consulting the index table and traversing previous memory accesses, the adjacent addresses are instantly regarded as prefetch candidates.

As multiple address pairs map to the same deltas compared to only indexing the addresses, the Markov prefetcher has a more significant memory requirement. According to Nesby and Smith, the distance prefetcher generally performs better than the Markov prefetcher with less storage overhead [5].

B. Program Counter/Delta Correlation (PC/DC)

The PC/DC prefetcher, when implemented using an index table and a GHB, is one of the most effective known prefetchers concerning storage efficiency and coverage [12]. PC/DC has shown impressive results for SPEC benchmarks having limited storage of 256 entries in the index table and GHB, respectively [12].

The index table is index by the PC of the memory access miss instructions. Consecutive misses by the same PC are linked together in the GHB. When a cache miss occurs, two successive deltas between the miss address and the two previous misses by the same PC are calculated by traversing the linked list. The traversal continues looking for the same delta pair. When finding a delta pair, the traversal reverses, starting at the match. Prefetch candidate addresses are calculated by applying the subsequent delta sequence to the base address of the miss encountered [12]. This process continues until reaching the prefetch depth. Note that the prefetcher does not regard any adjacent entries as prefetch candidates.

VIII. Conclusion

This paper showed that the Distance prefetcher (G/DC) is a decisive contribution to improve cache utilisation. It presented an implementation that increases SPEC2000 benchmark tests' harmonic mean speedup by 3% with limited hardware overhead. Additionally, it has the option of achieving a 4% performance increase (although with substantial overhead). For applications exhibiting useful delta correlations in the global miss address stream, the prefetcher can increase performance by 14%. Comparing it against other prefetchers on a wide variety of applications show that its performance is far from optimal. However, its performance is remarkable, and it has to be regarded as an adequate option when choosing prefetchers.

References

[1] S.-L. Lu, T. Karnik, G. Srinivasa, K.-Y. Chao, D. Carmean, and J. Held, "Scaling the "Memory Wall": Designer track," in *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2012, pp. 271–272, iSSN: 1092-3152.

[2] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH Computer Architecture News*, vol. 23, no. 1, pp. 20–24, Mar. 1995. [Online]. Available: http://portal.acm.org/citation.cfm?doid=216585.216588

[3] Y. Chen, H. Zhu, H. Jin, and X.-H. Sun, "Algorithm-level Feedback-controlled Adaptive data prefetcher: Accelerating data access for high-performance processors," *Parallel Computing*, vol. 38, no. 10, pp. 533–551, Oct. 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819112000488

[4] J. L. Hennessy, *Computer architecture: a quantitative approach*, sixth edition ed. Cambridge, MA: Morgan Kaufmann Publishers, 2019.

[5] K. Nesbit and J. Smith, "Data Cache Prefetching Using a Global History Buffer," in *10th International Symposium on High Performance Computer Architecture (HPCA'04)*. Madrid, Spain: IEEE, 2004, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/1410068/

[6] Tien-Fu Chen and Jean-Loup Baer, "Effective hardware-based data prefetching for high-performance processors," *IEEE Transactions on Computers*, vol. 44, no. 5, pp. 609–623, May 1995. [Online]. Available: http://ieeexplore.ieee.org/document/381947/

[7] Y. Chen, S. Byna, and X.-H. Sun, "Data access history cache and associated data prefetching mechanisms," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing - SC '07*. Reno, Nevada: ACM Press, 2007, p. 1. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1362622.1362651

[8] G. Kandiraju and A. Sivasubramaniam, "Going the distance for TLB prefetching: an application-driven study," in *Proceedings 29th Annual International Symposium on Computer Architecture*. Anchorage, AK, USA: IEEE Comput. Soc, 2002, pp. 195–206. [Online]. Available: http://ieeexplore.ieee.org/document/1003578/

[9] NTNU, "M5 simulator system tdt4260 computer architecture user documentation."

[10] "SPEC CPU2000." [Online]. Available: https://www.spec.org/cpu2000/docs/readme1st.html

[11] M. Grannæs, *Reducing Memory Latency by Improving Resource Utilization*. NTNU, 2010, accepted: 2014-12-19T13:36:05Z. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/252194

[12] B. Falsafi and T. F. Wenisch, "A Primer on Hardware Prefetching," *Synthesis Lectures on Computer Architecture*, vol. 9, no. 1, pp. 1–67, May 2014. [Online]. Available: https://doi.org/10.2200/S00581ED1V01Y201405CAC028

[13] D. Joseph and D. Grunwald, "Prefetching using Markov predictors," *IEEE Transactions on Computers*, vol. 48, no. 2, pp. 121–133, Feb. 1999. [Online]. Available: http://ieeexplore.ieee.org/document/752653/