Evaluating Global Delta Correlation (G/DC) Prefetcher on Simulated Hardware

Håkon Harnes

Vetle Harnes

***Abstract*—The advancements in processor performance in recent years by far exceeds the improvements in memory access latency. Numerous techniques have been developed to compensate for this, among others prefetching. This paper presents a prefetching algorithm called Global Delta Correlation (G/DC). G/DC is a correlation based distance prefetcher that uses a Global History Buffer (GHB) to keep track of memory accesses and an Index Table (IT) for faster look-ups into the GHB. The M5 hardware simulator was used to simulate the prefetcher implementation. The SPEC CPU2000 benchmark suite was used to evaluate its performance against other prefetchers. The paper also compares the three different implementations of the G/DC scheme, which is width, depth and hybrid. We found the G/DC prefetcher was able to achieve an average speedup of 5%.**

I. Introduction

In the last half-century microprocessor and semiconductor technology has faced a rapid development in an effort to match the ever growing dependence on technology. This has allowed us to put exponentially more transistors into integrated circuits to significantly increase the clock frequency of microprocessors [1]. This yielded an on average growth in clock frequency of 40% between 1988 and 2002, and even though it is currently stagnating it has increased by 2% yearly [2]. On the other hand, the improvement in main memory bandwidth and latency has been lagging behind. This is what is referred to as the ”Memory wall” [3]. This indicates that the main performance bottleneck in modern computers is memory access latency, not the theoretical amount of Instructions Per Cycle (IPC) the processor can execute.

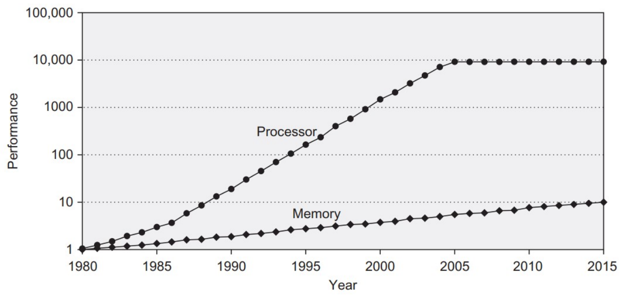

Fig. 1. The memory wall represents the increasing gap between processor and memory performance [3].

To compensate for this bottleneck multiple levels of high performance caches were introduced. These caches took advantage of the principles of spatial and temporal locality. The idea was to store the required data in smaller and faster memory devices closer to the processor [4]. Although it was a good solution, limited cache sizes and associativity led to the eviction of useful data. This introduced the need for predicting what data accesses would be required in the future, and then fetch them in advance. This is referred to as prefetching. Prefetching exploits memory access patterns to decrease the memory bottleneck.

This paper evaluates the performance of the Global Delta Correlation (G/DC) prefetcher. G/DC is a distance prefetcher that uses delta-correlation to predict future memory accesses. It stores a list of global miss addresses in a Global History Buffer (GHB). The GHB is stored as a linked list and be accessed through an indirect indexed hash table for faster look-ups.

The prefetcher has been simulated using a modified M5 hardware simulator [5] and evaluated using the SPEC CPU2000 benchmark suite [6]. The G/DC prefetcher was able to achieve a 5% average speedup. On single tests, it was able to achieve speedups of up to 37%. In addition, it requires little hardware overhead to implement.

II. Background

Cache prefetching is an efficient way to decrease the memory access latency. When implementing a cache prefetcher, it is important that the performance gain outweighs the silicone overhead. If not, it would be more beneficial to add more cores [7].

Simple prefetchers like adjacent and strided prefetchers are based on spatial locality. The adjacent prefetcher fetches the next block in addition to the one the processor needs. The strided prefetcher is a bit more advanced. It detects memory access patterns and fetches blocks with a constant stride between them [8]. However, these prefetchers are unable to detect more complex memory access patterns. More sophisticated prefetchers take complex correlations between memory accesses into account. To manage this added complexity, a Global History Buffer (GHB) with an Index Table (IT) can be used.

A. Global History Buffer (GHB)

In 2004 Nesbit and Smith [9] proposed using a GHB for prefetching. The GHB is a circular First In First Out (FIFO) table, that holds the addresses of the most recent cache misses. The entries in the table stores a link pointer, which makes the GHB entries a linked list, and a global miss address. The addresses with equal indices is stored in a time-ordered sequence. This allows correlation based prefetching algorithms to perform, while needing considerably less memory. Increasing the size of the GHB allows it to store more entries, but comes at the price of using more silicone area on the chip. The GHB is depicted in Fig. 2.

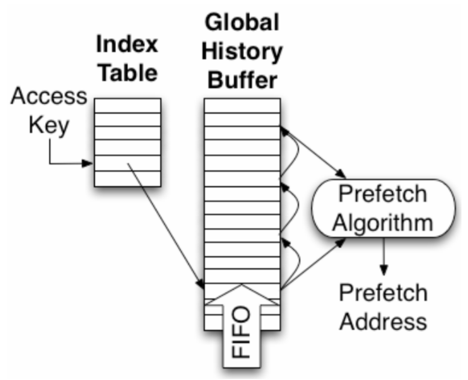

Fig. 2. Global History Buffer that is indexed by an index table. The data in the GHB is used as input to the prefetch algorithm [9].

B. Index Table (IT)

The Index Table (IT) is used to access the GHB, as seen in Fig. 2. The IT stores a list of key-pointer pairs. The pointers in the IT point to the previous GHB entry associated with the same key. This decreases the overhead of accessing the GHB substantially.

C. Markov prefetching

The Markov Prefetching is a correlation based prefetcher. Correlation based prefetching associates the given prefetched address with a correlation table that is based on earlier memory accesses [10]. If the address does not exist in the table, it is added to it. For instance, with a miss address of A, and a correlation table as depicted in Fig. 2, Markov prefetching would prefetch the data in C and B.

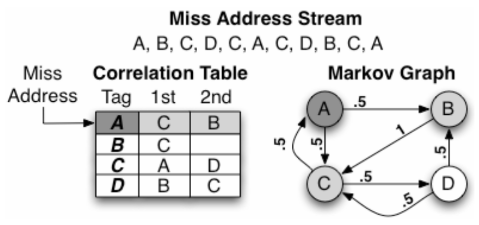

Fig. 3. Markov prefetching [9].

D. Distance prefetching

Distance prefetching tries to detect many of the patterns the Markov Prefetching can accommodate while taking advantage of the strided behavior in execution [11]. Distance prefetching keeps track of the difference between successive addresses to make predictions. This allows it to reuse memory access patterns for different miss addresses. This improves on Markov's high memory requirement. When combined with the GHB one can achieve even lower memory requirements without loosing the ability to recognize dynamic memory access patterns.

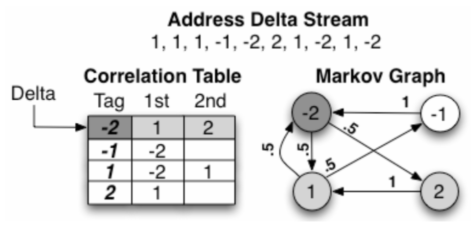

Fig. 4. Distance prefetching [9].

III. Global Delta Correlation (D/DC)

As proposed by Nesbit and Smith [9], our implementation uses GHB indexed by an IT in combination with a distance based prefetcher. The GHB address list and coordinates is maintained by a simple state machine. The GHB is implemented as a circular buffer in a FIFO manner. This means that data elements are inserted into the bottom of the list and removed from the top. This gives priority to the history with the most recent and common occurrence. When a cache miss occurs, the address is inserted into the GHB at the head pointer, and the link entry is given the current value in the index table. Then, the IT link entry is updated with a pointer to the head pointer. This process is illustrated in Fig. 5.

The IT uses the delta as an access key to get the correct entry in the GHB. Once the correct index is obtained we check the delta stream adjacent to the delta key or similar delta keys to find the addresses we wish to prefetch. This can be seen in Fig. 5. Here we use a delta of -2 to predict the future addresses needed. In this example we end up checking all the matching deltas (dark gray) and fetch the adjacent deltas that is later in the delta stream (light gray). We then use this to find our predicted address and prefetch it.

If the IT does not contain a pointer to a corresponding GHB entry, the prefetcher will act as a sequential prefetcher. This allows the prefetcher to keep fetching potentially useful data, even if there is no useful history present.

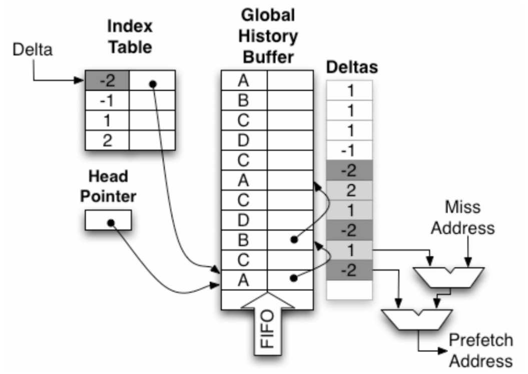

Fig. 5. Global Delta Correlation (G/DC) [9].

In this paper we have implemented three variants of the G/DC prefetcher: Width, depth and hybrid.

A. Width

Width limits the number of previous entries that are visited. For instance, with a a width of two, the prefetcher will look at the history of the two previous entries in the GHB with the same delta value. This is reflected in Fig. 5 by the curved arrows. This favours recent behaviour when deciding prefetch candidates.

B. Depth

Depth limits the number of adjacent deltas that are visited. This is reflected in Fig. 5 by the gray shaded deltas. This enables the prefetcher to run further ahead of the actual address stream [12].

C. Hybrid

The hybrid approach combines both width and depth.

IV. METHODOLOGY

A. Simulator

The simulator is a modified version of the M5 hardware simulator. It is an open-source simulator based on the architecture of the DEC Alpha Tsunami system, specifically the Alpha 21264 microprocessor. It is a four-issue superscalar, out-of-order processor that can rearrange a large number of instructions, and perform speculative execution [13]. The specifications of the simulator used is presented in Table I.

The modified version of the M5 simulator presents a simplified interface to the second level (L2) cache. The interface contains methods for prefetching data, as well as information about cache accesses [5].

Specification	Size
L1 data cache	64 KiB
L1 instruction cache	32 KiB
L1 cache block size	32 KiB
L2 cache size	1 MiB
L2 cache block size	32 KiB
Memory bus clock	400 MHz
Memory bus latency	30 ns
Memory bus width	8

TABLE I
M5 SIMULATOR SPECIFICATIONS

B. Benchmark suite

The SPEC CPU2000 benchmark suite has been used to evaluate the performance of the prefetcher. The benchmark suite contains a wide range of tests that are based on real-word user applications. They are compute-intensive applications that measure the performance of the processor, memory and compiler [6]. The benchmark suite is presented in Table II.

The tests in the benchmark suite often do some initialization and setup on startup. The prefetcher evaluation is therefore performed after the tests have run for some time. The number of warm up instructions used in this paper was one billion (10^9) instructions.

Benchmark	Description
ammp	Computational chemistry
applu	Partial differential equations
apsi	Weather prediction
art	Neural network simulation
bzip2	Compression
galgel	Fluid dynamics
swim	Shallow water modeling
twolf	Place and route simulator
wupwise	Quantum chromodynamics

TABLE II
SPEC CPU2000 BENCHMARK SUITE

C. Statistics

The statistics is used to evaluate the prefetcher performance. The terminology used in the statistics is presented in Table III.

Term	Description
IPC	Instructions per cycle
Cache miss	Number of L2 cache misses
Good prefetch	The prefetched block is referenced by the application before it is replaced.
Bad prefetch	The prefetched block is replaced without being referenced

TABLE III
TERMINOLOGY

The speedup is is a commonly used proxy for overall performance. It measures the relative performance gain.

$$\text{Speedup} = \frac{\text{IPC}_{\text{With prefetcher}}}{\text{IPC}_{\text{Without prefetcher}}}$$

The accuracy measures the number of useful prefetches issued by the prefetcher.

$$\text{Accuracy} = \frac{\text{Good prefetches}}{\text{Total prefetches}}$$

The coverage measures how many potential candidates for prefetches that were actually identified by the prefetcher.

$$\text{Coverage} = \frac{\text{Good prefetches}}{\text{Cache misses without prefetching}}$$

The harmonic mean is a kind of average used to aggregate each benchmark speedup score into an average speedup.

$$\text{Harmonic mean} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

V. Results

In this section, the width, depth and hybrid variants of the G/DC prefetcher are evaluated and compared against other prefetching schemes. First, the test results are presented. The prefetcher was tested using a prefetching degree of 4, and a GHB and IT size of 256. Second, the parameter testing results are described. The prefetch degree, GHB and IT size have been tested and evaluated.

A. Test performance

The G/DC hybrid prefetcher achieved an average speedup of 5%, as can be seen in Fig. 6. All three variants of G/DC outperformed the simple prefetchers, like the sequential on access, sequential on miss and tagged schemes. The hybrid variant was even able to perform on-par with DCPT. However, both DCPT-P and RPT were able to achieve higher speedups than the all of the G/DC variants. This may be caused by the fact that DCPT-P and RPT are suited for recognizing strided memory access patterns [14], while G/DC is not.

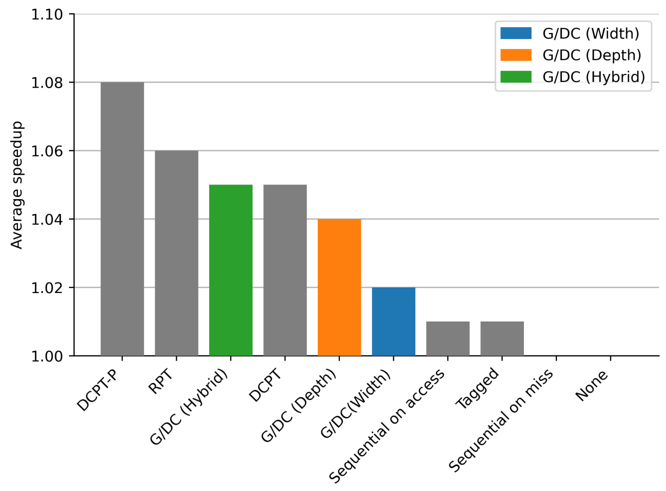

Fig. 6. Average speedups for the different prefetcher algorithms.

As illustrated in Fig. 7, the three G/DC variants performs similarly on the majority of the tests. The two big outliers are the ammp and swim tests. In the ammp test, the hybrid variant vastly outperformed the the others with a speedup of 37%, followed by the depth variant at 27%, and the width variant at 0%. On the other hand, in the swim test, the order was reversed. This can be attributed to the hybrid prefetcher being able to access more delta-correlations before it turns into a sequential prefetcher. We could also theorize that the depth variant is more likely to have an available node with enough history to satisfy the degree than the width variant. This is likely due to the FIFO nature of the GHB, causing older entries to be evicted. This indicates that the ammp test has a higher degree of delta-correlation in it than swim does.

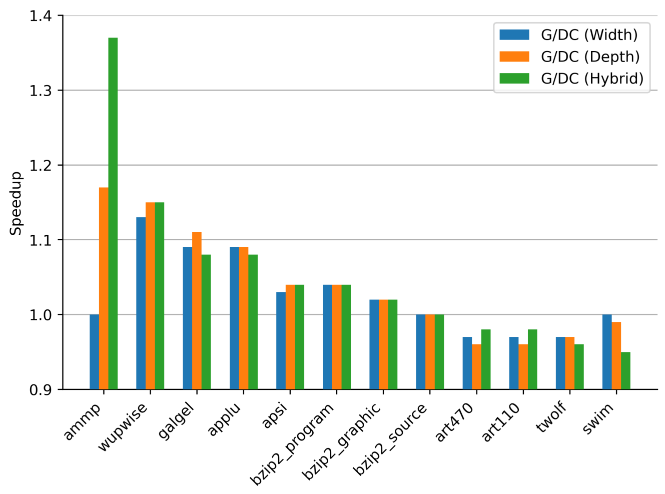

Fig. 7. Speedup for the different benchmark tests.

B. Parameters

The average speedup when varying the prefetch degree is presented in Fig. 8. Prefetch degrees one and two yield a decent speedup. It seems prefetch degree four is the most optimal for all three variants. Increasing the prefetch degree further has a negative impact on the speedup. This decrease in speedup is likely caused by several factors. First, the higher the prefetcher degree, the more data is prefetched. A result of this is that less relevant data is prefetched, and useful cache lines may be evicted. This wastes processor cycles and pollutes the cache. Second, when there is not enough data in the GHB, the G/DC prefetcher becomes sequential. This is an inferior prefetcher scheme that does not yield a substantial speedup. Our findings show there is a clear trade-off between prefetching too much data, and not prefetching enough. The optimal prefetch degree in our implementation has been found to be four.

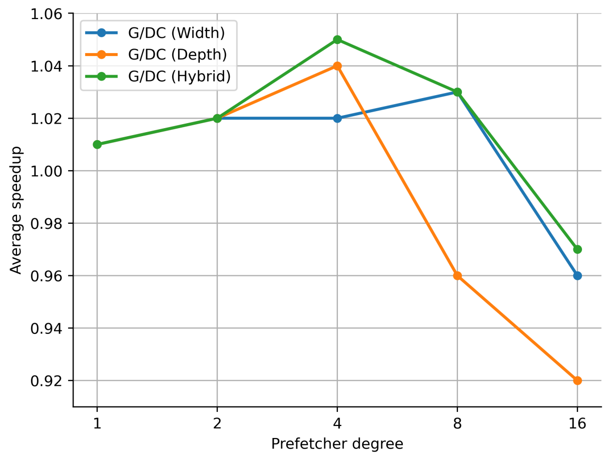

Fig. 8. Average speedup for different prefetch degrees.

The average speedup achieved with different GHB and IT sizes is shown in Fig. 9. The graph clearly shows that sizes smaller than 16 does not yield substantial speedups. It seems the minimum size for the GHB and IT is 16, and that going above this yields little performance gains. The highest speedup was achieved with size 256. These results are different from Nesbit and Smith's [9]. Their findings was that the ideal size was 512. However, our tests have been conducted with prefetch degree four, which may have impacted the results. Nesbit and Smith do not state how they found out 512 was the ideal size, but one can speculate this was found with a higher prefetching degree. The reason for this is that they found that the higher the prefetcher degree, the higher the speedup. We did not find that, and therefore our tests have been conducted with prefetch degree four.

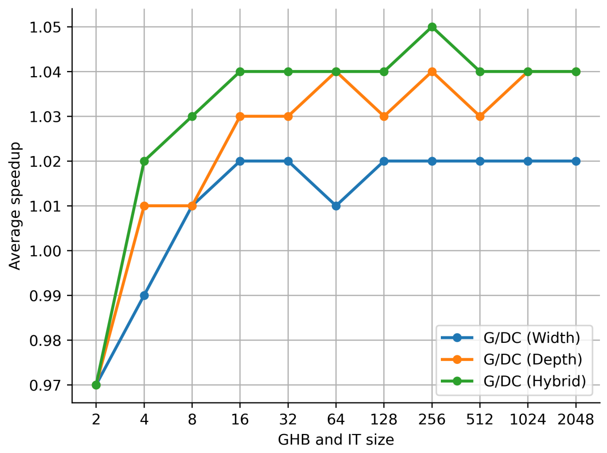

Fig. 9. Average speedup for different GHB and IT sizes.

VI. DISCUSSION

The best speedup was achieved with a GHB and IT size of 256. Going by Nesbit and Smith's calculations, this would require 4KB of storage [9]. In addition, a 2014 paper states that the size of the GHB and IT cause little hardware overhead [15]. Also, the prefetcher has been shown to achieve a speedup of 4% with an IT and GHB size of 16. This means our prefetcher implementation should be realistic to implement in a processor without any substantial hardware overhead.

A weakness with our test is the fact that we assumed the optimal GHB to IT ratio is 1:1. This is the ratio that Nesbit and Smith found to give the best performance [9]. However, our implementation runs on different hardware, so that ratio may not be optimal in our setup. Considering we were unable to achieve the same speedup as Nesbit and Smith, it could be fair to assume that our prefetcher could benefit from a different configuration. This could either improve the performance or enable us to achieve the same performance with even less silicone overhead.

Another weakness with our method is that we could have experimented more with the sequential fallback prefetcher. When there is no relevant data in the GHB, the prefetcher becomes sequential, and proceeds by prefetching enough data to satisfy the prefetch degree. We tested with and without the sequential fallback prefetcher, but found that generally the implementation benefited from it. We could have tweaked it, and for instance only used the sequential fallback prefetching when the MSHR queue was empty, or when it was not full. This would ensure the prefetcher would not prefetch too much data, even at higher prefetching degrees.

VII. RELATED WORK

There are many other delta-correlation based prefetchers that aim to solve the same problem as G/DC. These all have different complexity costs and performance, and hence serve different purposes.

A. PC/CD

Also proposed by Nesbit and Smith [9], PC/DC is a delta-correlation prefetcher. It uses delta pairs as correlation key by searching the delta stream in reverse order. This allows it to take better advantage of the GHB, hence lowering the size requirement for a given speedup. They found that PC/DC was able to to achieve a slightly better performance while only occupying half the silicone real overhead of G/DC.

B. DCPT

Jahre et al. [14] proposed combining the table based design of RPT and delta-correlating design of PC/DC. This was done by using an RPT table in combination with a GHB and IT. The prefetcher used 4KB of silicone and was able to achieve a 27.2% speedup over PC/DC, which is considered significant increase.

C. Runahead Metadata

Runahead MetaData (RMD) [8] is a pairwise-correlating data prefetcher. RMD uses two runahead metadata tables to predict what addresses/deltas might occur. This prefetcher has a small overhead, which is motivated by the movement towards multi-core processors where prefetchers needs to be simple and as low-overhead as possible [16]. RMD is considered state-of-the-art when it comes to pairwise-correlating data prefetchers [8].

VIII. CONCLUSION

This paper shows that Global Delta Correlation (G/DC) is a viable option for improving cache utilization. It has been tested and evaluated using the SPEC CPU2000 benchmark suite, and has been found to achieve an average speedup of 5%. On single tests it has been able to achieve speedups of up to 37%. This is on-par with many of the other prefetcher schemes it has been evaluated against. However, it seems G/DC is not able to recognize strided access patterns efficiently, as evident by the fact that it is outperformed by both DCPT-P and RPT. That being said, the G/DC prefetcher provides a reasonable speedup with little hardware overhead.

References

[1] G. E. Moore. (2006) Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. Accessed: 03.02.2022. [Online]. Available: https://ieeexplore.ieee.org/document/4785860

[2] J. L. Hennsey and D. a. Patterson, *Computer Architecture: A quantitative approach*. Morgan Kaufmann Publishers In, 2017, ISBN: 978-0128119051.

[3] W. Wulf and S. McKee. (1994, Dec) Hitting the memory wall: Implications of the obvious. Accessed: 27.01.2022. [Online]. Available: https://dl.acm.org/doi/10.1145/216585.216588

[4] S. Przybylski, M. Horowitz, and J. Hennessy. (1989) Characteristics of performance-optimal multi-level cache hierarchies. Accessed: 15.02.2022. [Online]. Available: https://ieeexplore.ieee.org/document/714545

[5] NTNU, "M5 simulator system tdt4260 computer architecture user documentation," Jan 2022, accessed: 27.01.2022.

[6] Spec cpu2000. Accessed: 10.02.2022. [Online]. Available: https://www.spec.org/cpu2000/

[7] C. Kaynak, B. Grot, and B. Falsafi. (2013) Shift: Shared history instruction fetch for lean-core server processors. Accessed: 15.02.2022. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7847632

[8] S. Mittal. (2016, aug) A survey of recent prefetching techniques for processor caches. New York, NY, USA. Accessed: 10.02.2022. [Online]. Available: https://doi.org/10.1145/2907071

[9] K. Nesbit and J. Smith. (2004) Data cache prefetching using a global history buffer. Accessed: 03.02.2022. [Online]. Available: https://ieeexplore.ieee.org/document/1410068

[10] D. Joseph and D. Grunwald. (1999) Prefetching using markov predictors. Accessed: 12.02.2022. [Online]. Available: https://ieeexplore.ieee.org/document/752653

[11] G. Kandiraju and A. Sivasubramaniam. (2002) Going the distance for tlb prefetching: an application-driven study. Accessed: 04.02.2022. [Online]. Available: https://ieeexplore.ieee.org/document/1003578

[12] Y. Solihin, J. Lee, and J. Torrellas. (2002) Using a user-level memory thread for correlation prefetching. [Online]. Available: https://ieeexplore.ieee.org/document/1003576

[13] "Alpha 21264," Feb 2022, accessed: 27.01.2022. [Online]. Available: https://en.wikipedia.org/wiki/Alpha_21264

[14] M. Grannaes, M. Jahre, and L. Natvig, "Storage efficient hardware prefetching using delta correlating prediction tables," *Journal of Instruction-Level Parallelism*, vol. 13, pp. 1–16, 01 2011.

[15] B. Falsafi and T. Wenisch, "A primer on hardware prefetching," *Synthesis Lectures on Computer Architecture*, vol. 9, pp. 1–67, 05 2014.

[16] P. Esmaili-Dokht, M. Bakhshalipour, B. Khodabandeloo, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Scale-out processors & energy efficiency," 08 2018, accessed: 24.03.2022. [Online]. Available: https://www.researchgate.net/publication/327050323_Scale-Out_Processors_Energy_Efficiency