The Alpha 21264 Microprocessor Architecture

R. E. Kessler, E. J. McLellan[1], and D. A. Webb

Compaq Computer Corporation
334 South St., Shrewsbury, MA 01545
richard.kessler@digital.com

Abstract

The 21264 is the third generation Alpha microprocessor from Compaq Computer (formerly Digital Equipment) Corporation. This microprocessor achieves the industry-leading performance levels of 30+ Specint95 and 50+ Specfp95. In addition to the aggressive 600 MHz cycle time in a 0.35 um CMOS process, there are also many architectural features that enable the outstanding performance level of the 21264. This paper discusses many of these architectural techniques, which include an out-of-order and speculative execution pipeline coupled with a high-performance memory system.

1 Introduction

The Alpha Microprocessor has been the performance leader since its introduction in 1992. An unequalled cycle time at the time, facilitated by a clean RISC architecture and leading edge design techniques, provided much of the performance difference. The 21264 (EV6) is the third generation super-scalar Alpha microprocessor. (See [Dob92][Edm95] for descriptions of the prior two generations.) In this design, absolute performance leadership was again a project goal. The 21264 achieves this goal using a unique combination of advanced circuit and architectural techniques.

Detailed architectural and circuit analysis in the early stages of the 21264 project showed that more aggressive micro-architectural techniques were possible while continuing the leadership clock frequencies that have become an Alpha tradition. The 21264 shows that a clean RISC architecture not only allows for a very fast clock rate, currently up to 600 MHz, but it also allows for sophisticated micro-architectural techniques that maximize the number of instructions executed every cycle. This combination results in industry-leading performance levels for the third consecutive Alpha generation.

[1] Work performed while a Digital Equipment (now Compaq) employee. Currently employed by C-Port Corporation.

The 21264 is a super-scalar microprocessor with out-of-order and speculative execution. Out-of-order execution implies that instructions can execute in an order that is different from the order that the instructions are fetched. In effect, instructions execute as soon as possible. This allows for faster execution since critical path computations are started and completed as soon as possible. In addition, the 21264 employs speculative execution to maximize performance. It speculatively fetches and executes instructions even though it may not know immediately whether the instruction will be on the final execution path. This is particularly useful, for instance, when the 21264 predicts branch directions and speculatively executes down the predicted path. The sophisticated branch prediction in the 21264 coupled with speculative and dynamic execution extracts the most instruction parallelism from applications.

The 21264 memory system is another enabler of the high performance levels of the 21264. On chip and off-chip caches provide for very low latency data access. In addition, many memory references can be serviced in parallel to all caches in the 21264 as well as to the off-chip memory system. This allows for very high bandwidth data access.

This paper describes many of the micro-architectural techniques used to achieve the high performance levels in the 21264 Alpha microprocessor. (Two other references to the 21264 are: [Gie97][Lei97].). Figure 1 shows a high-level overview of the 21264 pipeline. Stage 0 is the *instruction fetch* stage that provides four instructions per cycle from the instruction cache. Stage 1 assigns instructions to slots associated with the integer and floating-point queues. The *rename* (or *map*) stage (2) maps instruction "virtual" registers to internal "physical" registers and allocates new physical registers for instruction results. The *issue* (or *queue*) stage (3) maintains an inventory from which it dynamically selects to issue up to 6 instructions – this is where instruction issue reordering takes place. Stages 4 and higher constitute the instruction execution stages that support all arithmetic and memory operations. Each stage is described in more detail below.

2 Instruction Fetch

The instruction fetch stage is the beginning of the 21264 instruction pipeline. Four instructions are fetched each cycle to be delivered to the out-of-order execution engine. The 21264 uses many architectural techniques to provide maximum fetch efficiency. One important enabler is a large, 64K byte, two-way set-associative instruction cache. This offers much improved hit rates as compared to the 8K direct-mapped instruction cache used in the Alpha 21164.

2.1 Line and Set Prediction

The 21264 instruction cache implements two-way associativity via a line and set prediction technique that combines the speed advantage of a direct-mapped cache with the lower miss ratio of a two-way set-associative cache. Each fetch block of four instructions includes a line and set prediction. This prediction indicates where to fetch the next block of four instructions from, including which set (i.e. which of the two choices allowed by two-way associative cache) should be used. These predictors are loaded on cache fill and dynamically re-trained when they are in error. The mispredict cost is typically a single-cycle bubble to re-fetch the needed data. Line and set prediction is an important speed enhancement since the mispredict cost is so low and the line/set mispredictions are rare (the hit rates are typically 85% or higher in simulated applications/benchmarks).

In addition to the speed benefits of direct cache access, there are other benefits that come from line and set prediction. For example, frequently encountered predictable branches, such as loop terminators, will avoid the mis-fetch penalty often associated with a taken branch. The 21264 also trains the line predictor with the address of jumps that use direct register addressing. Code using DLL (dynamically linked library) routines will benefit after the line predictor is trained with the target.

The 21264 line predictor does not train on every mispredict. There is a 2-bit hysteresis associated with each line that only enables training after the predictor has been in error several times recently. This avoids some unnecessary training and misprediction

2.2 Branch Prediction

Another important contributor to fetch efficiency is branch prediction. The 21264 speculative execution capabilities make branch prediction a more important contributor to overall performance than with previous microprocessor generations. Studies show that branch pattern behavior sometimes correlates with the execution of a single branch at a unique PC location (i.e. local history), and pattern behavior sometimes correlates with execution of all previous branches (i.e. global history). Both correlation techniques are needed to extract the maximum branch prediction efficiency.

The 21264 implements a sophisticated tournament branch prediction scheme that dynamically chooses

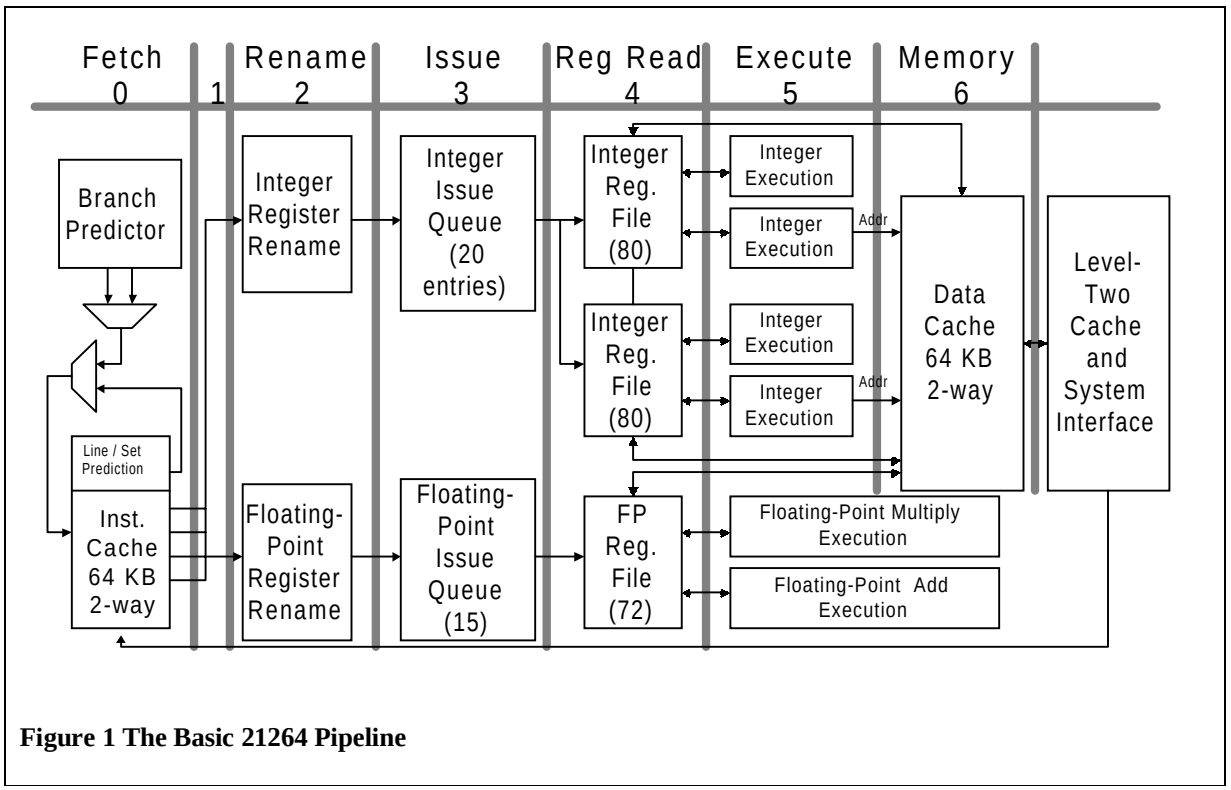

Figure 1 The Basic 21264 Pipeline

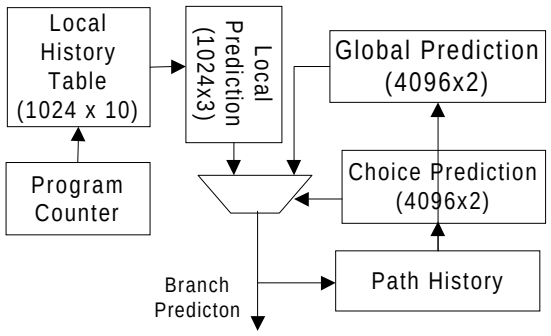

Figure 2 The 21264 Tournament Branch Predictor

This figure is a block-diagram of the 21264 branch predictor. The local prediction path is on the left. The global prediction path and the chooser are on the right.

between local and global history to predict the direction of a given branch [McF93]. The result is a branch predictor that produces a better prediction accuracy than larger tables of either individual method, 90-100% on most simulated applications/benchmarks.

Figure 2 shows the structure of the 21264 tournament branch predictor. The local history table holds 10 bits of branch history for up to 1024 branches, indexed by the instruction address. The 21264 uses the 10-bit local history to pick from one of 1024 prediction counters. The local prediction is the most-significant bit of the prediction counter. After branches issue and retire the 21264 inserts the true branch direction in the local history table and updates the referenced counter (using saturating addition) to train the correct prediction.

The local prediction will be very useful, for example, with an alternating taken/not-taken sequence from a given branch. The local history of the branch will eventually resolve to either 1010101010 or 0101010101 (the alternating pattern of zeroes and ones indicates the success/failure of the branch on alternate invocations). As the branch executes multiple times, it will saturate the prediction counters corresponding to these local history values and make the prediction correct. Any repeating pattern of 10 branch invocations can be trained this way.

The global predictor is a 4096 entry table of two-bit saturating counters that is indexed by the global, or path, history of the last twelve branches. The prediction is the most-significant bit of the indexed prediction counter. Global history is useful when the outcome of a branch can be inferred from the direction of previous branches. For example, if a first branch that checks for a value equal to ten succeeds, a second branch that checks for the same value to be even must also always succeed. The global history predictor can learn this pattern with repeated invocations of the two branches; eventually, the global prediction counters with indices that have their lower-most bit set (indicating that the last branch was taken) will saturate at the correct value. The 21264 maintains global history with a silo of thirteen branch predictions and the 4096 prediction counters. The silo is backed up and corrected on a mispredict. The 21264 updates the referenced global prediction counter when the branch retires.

The 21264 updates the chooser when a branch instruction retires, just like the local and global prediction information. The chooser array is 4096 two-bit saturating counters. If the predictions of the local and global predictor differ, the 21264 updates the selected choice prediction entry to support the correct predictor.

The instruction fetcher forwards speculative instructions from the predicted path to the execution core after a branch prediction. The 21264 can speculate through up to 20 branches.

3 Register Renaming and Out-of-Order Issue

The 21264 offers out-of-order efficiencies with much higher clock speeds than competing designs. This speed, however, is not accomplished by the restriction of dynamic execution capabilities. The out-of-order issue logic in the 21264 receives four fetched instructions every cycle, renames/re-maps the registers to avoid unnecessary register dependencies, and queues the instructions until operands and/or functional units become available. It dynamically issues up to six instructions every cycle - four integer instructions and two floating-point instructions.

3.1 Register Renaming

Register renaming assigns a unique storage location with each write-reference to a register. The 21264 speculatively allocates a register to each register-result-producing instruction. The register only becomes part of the architectural register state when the instruction commits/retires. This allows the instruction to speculatively issue and deposit its result into the register file before the instruction is committed. Register renaming also eliminates write-after-write and write-after-read register dependencies, but preserves all the read-after-write register dependencies that are necessary for correct computation. Renaming extracts the maximum parallelism from an application since only necessary dependencies are retained and speculative execution is allowed in the instruction flow.

In addition to the 64 architectural (i.e. software-visible) registers, up to 41 integer and 41 floating point registers are available to hold speculative results prior to

instruction retirement in a large 80 instruction in-flight window. This implies that up to 80 instructions can be in partial states of completion at any time, allowing for significant execution concurrency and latency hiding. (Particularly since the memory system can track an additional 32 in-flight loads and 32 in-flight stores.) The 21264 tracks outstanding unretired instructions (and their associated register map information) so that the machine architectural state can be preserved in the case of a mis-speculation.

3.2 Out-of-Order Issue Queues

The issue queue logic maintains a list of pending instructions. Each cycle the separate integer and floating-point queues select from these instructions as they become data-ready using register scoreboards based on the renamed register numbers. These scoreboards maintain the status of the renamed registers by tracking the progress of single-cycle, multiple-cycle, and variable cycle (i.e. memory load) instructions. When functional unit or load data results become available, the scoreboard unit notifies all instructions in the queue that require the register value. These dependent instructions can issue as soon as the bypass result becomes available from the functional unit or load. Each queue selects the oldest data-ready and functional-unit-ready instructions for execution each cycle. The 20-entry integer queue can issue four instructions, and the 15-entry floating-point queue can issue two instructions per cycle.

The 21264 cannot schedule each instruction to any of the four integer execution pipes. Rather, it statically assigns instructions to two of the four pipes, either upper or lower, before they enter the queue. The issue queue has two arbiters that dynamically issue the oldest two queued instructions each cycle within the upper and lower pipes, respectively. This static assignment to upper/lower fits well with the integer execution engine - some functional units do not exist in both the upper and lower pipelines, and the dynamic scheduling of the issue queue minimizes cross-cluster delays. Section 4 discusses this more.

The queues issue instructions speculatively. Since older instructions are given priority over newer instructions in the queue, speculative issues do not slow down older, less speculative issues. The queue is collapsing – an entry becomes immediately available once the instruction issues or is squashed due to mis-speculation.

4 Execution Engine

To support the high frequency goals of the project, the design of the integer register file was a particular challenge. Typically, all execution units require access to the register file, making it a single point of access and a potential bottleneck to processor performance. With as many as fourteen ports necessary to support four simultaneous instructions in addition to two outstanding load operations, it was clear that the register file would be large and an implementation challenge. Instead, the 21264 splits the file into two clusters that contain duplicates of the 80-entry register file. Two pipes access a single register file to form a cluster, and the two clusters are combined to support 4-way integer instruction execution.

The incremental cost of this design is an additional cycle of latency to broadcast results from each integer cluster to the other cluster. Performance simulation shows this cost to be small – a few percent or less performance difference from an idealized un-clustered implementation with most applications. The integer issue queue dynamically schedules instructions to minimize the one cycle cross-cluster communication cost; An instruction can usually first issue on the same cluster that produces the result. This architecture provides much of the implementation simplicity and lower risk of a two-issue machine with the performance benefits of four-way integer issue. There are two floating-point execution pipes organized in a single cluster with a single 72-entry register file. Figure 3 shows the configuration.

The 21264 includes new functionality not present in prior Alpha microprocessors: a fully-pipelined integer multiply unit, an integer population count and leading/trailing zero count unit, a floating-point square-root functional unit, and instructions to move register values directly between floating-point and integer registers. It also provides more complete hardware support for the IEEE floating-point standard, including precise exceptions, NaN and infinity processing, and support for flushing denormal results to zero.

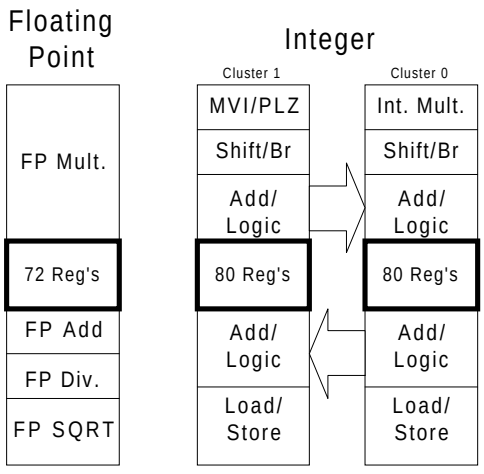

Figure 3 The 21264 Execution Pipes

This figure shows the four integer execution pipes (upper and lower for each cluster) and the two floating-point pipes in the 21264, together with the functional units in each.

5 Memory System

The memory system in the 21264 is high-bandwidth, supporting many in-flight memory references and out-of-order operation. It receives up to two memory operations (loads or stores) from the integer execution pipes every cycle. This means that the on-chip 64 KB two-way set-associative data cache is referenced twice every cycle, delivering 16 bytes every cycle. In effect, the data cache operates at twice the frequency of the processor clock. Since the cache is double-pumped this way there is full support for two memory references every cycle without conflict. The off-chip (level-two) cache provides a very fast backup store for the primary caches. This cache is direct-mapped, shared by both instructions and data, and can range from 1 to 16 MB. The off-chip clock-forwarded cache interface can support a peak data transfer rate of 16 bytes every 1.5 CPU cycles.

The latency of the virtual-indexed on-chip data cache is three cycles for integer loads and four cycles for floating-point loads. The latency to the physical-indexed off-chip cache is twelve cycles, depending on the speed of the cache. The 21264 supports many SRAM variants, including late-write synchronous, PC-style, and dual-data for very high frequency operation.

The 21264 also has a fast interface that allows the memory system surrounding the microprocessor to provide data quickly, typically from DRAM, upon a cache miss. The peak bandwidth of the clock-forwarded system interface is 8 bytes of data per 1.5 CPU cycles.

The 21264 memory system supports up to 32 in-flight loads, 32 in-flight stores, 8 in-flight (64-byte) cache block fills, and 8 cache victims. This allows a high degree of memory system parallel activity to the cache and system interface. It translates into high memory system performance, even with many cache misses. For example, we have measured a 1 GB/sec sustained memory bandwidth on the STREAMS benchmark [Str98].

5.1 Store/Load Memory Ordering

The 21264 memory system supports the full capabilities of the out-of-order execution core, yet maintains an in-order architectural memory model. This is a challenge, for example, when there are multiple loads and stores that reference the same address. It would be incorrect if a later load issued prior to an earlier store and, thus, did not return the value of the earlier store to the same address. This is a somewhat infrequent event, but it must be handled correctly. Unfortunately, the register rename logic cannot automatically handle this read-after-write memory dependency as it does other register dependencies because it does not know the memory address before the instruction issues. Instead, the memory system dynamically detects the problem case *after* the instructions issue (and the addresses are available).

The 21264 has hazard detection logic to recover from a mis-speculation that allowes a load to incorrectly issue before an earlier store to the same address. After the first time a load mis-speculates in this way, the 21264 trains the out-of-order execution core to avoid it on subsequent executions of the same load. It does this by setting a bit in a load wait table that is examined at fetch time. If the bit is set, the 21264 forces the issue point of the load to be delayed until all prior stores have issued, thereby avoiding all possible store/load order violations. This load wait table is periodically cleared to avoid unnecessary waits.

This example store/load order case shows how the 21264 memory system produces a result that is the same as in-order memory system execution while utilizing the performance advantages of out-of-order execution. Almost all of the major 21264 functional blocks are needed to implement this store/load order solution: fetch, issue queue, and memory system. This implementation provides the highest performance for the normal case when there are no dependencies since loads can be issued ahead of earlier stores. It also dynamically adjusts to perform well in the less frequent case where a load should not be scheduled before a prior store.

5.2 Load Hit / Miss Prediction

There are mini-speculations within the 21264 speculative execution pipeline. In order to achieve the three-cycle integer load hit latency, it is necessary to speculatively issue consumers of integer load data before knowing if the load hit or missed in the on-chip data cache. The consumers that receive bypassed data from a load must issue the same cycle as the load reads the data cache tags, so it is impossible for the load hit/miss indication to stop the issue of the consumers. Furthermore, it really takes another cycle after the data cache tag lookup to get the hit/miss indication to the issue queue. This means that consumers of the results produced by the consumers of the load data can also speculatively issue – even though the load may have actually missed!

The 21264 could rely on the general mechanisms available in the speculative execution engine to abort the speculatively executed consumers of the integer load data, but that requires a restart of the entire instruction pipeline. Given that load misses can be frequent in some applications, this technique would be too expensive. Instead, the 21264 has a mini-restart to handle this case. When consumers speculatively issue three cycles after a load that misses, two integer issue cycles (on all four integer pipes) are squashed and all integer instructions that issued during those two cycles are pulled back into the issue queue to be re-issued later. This means that both the consumer of the load data and the consumer of the consumer will be restarted and re-issued.

While this two-cycle window is less costly than a full restart of the processor pipeline, it still can be expensive

for applications that have many integer load misses. Consequently, the 21264 predicts when loads will miss and does not speculatively issue the consumers of the load data in that case. The effective load latency is five cycles rather than the minimum three for an integer load hit that is (incorrectly) predicted to miss.

The 21264 load hit/miss predictor is the most-significant bit of a 4-bit counter that tracks the hit/miss behavior of recent loads. The saturating counter decrements by two on cycles when there is a load miss, otherwise it increments by one when there is a hit.

The 21264 treats floating-point loads differently than integer loads for load hit/miss prediction. Their latency is four cycles and there are no single-cycle operations, so there is enough time to resolve the exact instruction that used the load result.

5.3 Cache Prefetching / Management

The 21264 provides cache prefetch instructions that allow the compiler and/or assembly programmer to take full advantage of the parallelism and high-bandwidth capabilities of the memory system. These prefetches are particularly useful in applications that have loops that reference large arrays. In these and other cases where software can predict memory references, it can prefetch the associated (64-byte) cache blocks to overlap the cache miss time with other operations. Software prefetches can also eliminate unnecessary data reads, and control cache-ability. The prefetch can be scheduled far in advance because the block is held in the cache until it is used.

Instruct.	Description
Normal Prefetch	The 21264 fetches the (64-byte) block into the (level one data and level 2) cache.
Prefetch with Modify Intent	The same as the normal prefetch except that the block is loaded into the cache in dirty state so that subsequent stores can immediately update the block.
Prefetch and Evict Next	The same as the normal prefetch except that the block will be evicted from the (level one) data cache as soon as there is another block loaded at the same cache index.
Write Hint 64	The 21264 obtains write access to the 64-byte block without reading the old contents of the block. The application typically intends to over-write the entire contents of the block.
Evict	The cache block is evicted from the caches.

Table 1 The 21264 Cache Prefetch and Management Instructions

Table 1 describes the cache prefetch and management instructions used in the 21264. The normal, modify-intent, and evict-next prefetches perform similar operations but are used in different specific circumstances. For each of them, the 21264 fills the block into the data cache if it was not already present in the cache. The write-hint 64 instruction is similar to a prefetch with modify intent except that the previous value of the block is not loaded. For example, this is very useful for zeroing out a contiguous region of memory. The evict instruction evicts the selected cache block from the cache.

6 Performance

We have currently measured performance levels of 30 Specint95 and 50 Specfp95 on early 21264 systems. These performance levels clearly place the 21264 as the highest-performer in the industry. System tuning, new system designs, compiler tuning, and faster 21264 variants will continue to increase the performance lead.

7 Conclusion

The 21264 is the fastest microprocessor available. It reaches excellent performance levels using a combination of the expected high Alpha clock speeds together with many advanced micro-architectural techniques, including out-of-order and speculative execution with many in-flight instructions. The 21264 also includes a high-bandwidth memory system to quickly deliver data values to the execution core, providing robust performance for many applications, including those without cache locality.

8 References

[Dob92] D. Dobberpuhl, et. al., "A 200 Mhz 64-bit Dual Issue CMOS Microprocessor", IEEE Journal of Solid State Circuits, Vol. 27, No. 11, November 1992, pp. 1555-1567.

[Edm95] J. Edmondson, et. al., "Superscalar Instruction Execution in the 21164 Alpha Microprocessor", IEEE Micro, Vol. 15, No. 2, April 1995.

[Gie97] B. Gieseke, et. al., "A 600 Mhz Superscalar RISC Microprocessor With Out-of-Order Execution", IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers, Feb. 1997, pp. 176-177.

[Lei97] Daniel Leibholz and Rahul Razdan, "The Alpha 21264: A 500 Mhz Out-of-Order Execution Microprocessor", Proceedings of IEEE COMPCON '97, pp. 28-36.

[McF93] S. McFarling, "Combining Branch Predictors", Technical Note TN-36, Digital Equipment Corporation Western Research Laboratory, June 1993. <www.research.digital.com/wrl/techreports/abstracts/TN-36.html>

[Str98] http://www.cs.virginia.edu/stream