# Data Prefetch Support

## Contents

## Introduction

The framework for data prefetch in GCC supports capabilities of a variety of targets. Optimizations within GCC that involve prefetching data pass relevant information to the target-specific prefetch support, which can either take advantage of it or ignore it. The information here about data prefetch support in GCC targets was originally gathered as input for determining the operands to GCC's `prefetch` RTL pattern, but might continue to be useful to those adding new prefetch optimizations.

Existing data prefetch support in GCC includes:

- A generic prefetch RTL pattern.
- Target-specific support for several targets.
- A `__builtin_prefetch` function that does nothing on targets that do not support prefetch or for which prefetch support has not yet been added to GCC.
- An optimization enabled by `-fprefetch-loop-arrays` that prefetches arrays used in loops.

Possibilities for future work include:

- Greedy prefetch [22] of data referenced by pointer variables, controlled by an option like `-fprefetch-pointers`. Jan Hubicka has said he is interested in doing this.
- Prefetch support for additional targets.
- Running benchmarks and analyzing results on various targets to validate prefetch optimization heuristics.
- Using profile information to guide prefetching of data.
- Other optimizations.

- Adding support for AltiVec-style streaming data prefetch.

This document is a work in progress. Please copy any comments about it to Janis Johnson, <janis187@us.ibm.com>.

## Elements of Data Prefetch Support

Data prefetch, or cache management, instructions allow a compiler or an assembly language programmer to minimize cache-miss latency by moving data into a cache before it it accessed. Data prefetch instructions are generally treated as hints; they affect the performance but not the functionality of software in which they are used.

### Locality

Data prefetch instructions often include information about the *locality* of expected accesses to prefetched memory. Such hints can be used by the implementation to move the data into the cache level where it will be the most good, or the least harm. Prefetched data in the same cache line as other data likely to be accessed soon, such as neighboring array elements, has *spatial locality*. Data with *temporal locality*, or *persistence*, is expected to be accessed multiple times and so should be left in a cache when it is prefetched so it will continue to be readily accessible. Accesses to data with no temporal locality are *transient*; the data is unlikely to be accessed multiple times and, if possible, should not be left in a cache where it would displace other data that might be needed soon.

Some data prefetch instructions allow specifying in which level of the cache the data should be left.

Locality hints determined in GCC optimization passes can be ignored in the machine description for targets that do not support them.

### Read or Write Access

Some data prefetch instructions make a distinction between memory which is expected to be read and memory which is expected to be written. When data is to be written, a prefetch instruction can move a block into the cache so that the expected store will be to the cache. Prefetch for write generally brings the data into the cache in an exclusive or modified state.

A prefetch for data to be written can usually be replaced with a prefetch for data to be read; this is what happens on implementations that define both kinds of instructions but do not support prefetch for writes.

### Size of block to access

The amount of data accessed by a data prefetch instruction is usually a cache line, whose size is usually implementation specific, but is sometimes a specified number of bytes.

### Base update

At least one target's data prefetch instructions has a *base update* form, which modifies the prefetch address after the prefetch. Base update, or pre/post increment, is also supported on load and store instructions for some targets, and this could be taken into consideration in code that uses data prefetch.

### Faulting v. Non-faulting

Some architectures provide prefetch instructions that cause faults when the address to prefetch is invalid or not cacheable. The data prefetch support in GCC assumes that only non-faulting prefetch instructions will be used.

### Miscellaneous Features

Some prefetch instructions have requirements about address alignment. These can be handled in the machine description; optimization passes do not need to know about them.

Optimizations will need information about various implementation dependent parameters of data prefetch support, including:

- number of simultaneous prefetch operations
- number of bytes prefetched

## Guidelines for Prefetching Data

Prefetch timing is important. The data should be in the cache by the time it is accessed, but without a delay that would allow other data to displace it before it is used.

Using prefetches that are too speculative can have negative effects, because there are costs associated with data prefetch instructions. These include wasting bandwidth, kicking other data out of the cache and causing additional conflict misses, consuming slots for memory instructions [26], and increasing code size, which can bump useful instructions out of the instruction cache.

Similarly, prefetching data that is already in the cache increases overhead without providing any benefit [25]. Data might already be in the cache if it is in the same cache line as data already prefetched (spatial locality), or if the data has been used recently (temporal locality).

On some (but not all) targets it makes sense to combine prefetching arrays in loops with loop unrolling [23] [26].

## Data Prefetch Support on GCC Targets

Variants of prefetch commands that fault are not included here. Some implementations of these architectures recognize data prefetch instructions but treat them as `nop` instructions. They are generally ignored for pages that are not cacheable. The exception to this is prefetch instructions with base update forms, for which the base address is updated even if the addressed memory cannot be prefetched.

The descriptions that follow are meant to describe the basic functionality of data prefetch instructions. For complete information about data prefetch support on a particular processor, refer to the technical documentation for that processor; the references provide a starting point for that information.

### Summary

| Target | Prefetch amount | Read/write | Locality hints | Other features to consider |
|---|---|---|---|---|
| 3DNow! | cache line; at least 32 bytes | yes | | |
| Alpha | cache line | yes | separate instruction for transient loads | |
| AltiVec | specified unit size, count, stride | yes | temporal locality | prefetch instruction must specify one of four data streams |
| IA-32 SSE | cache line; at least 32 bytes | no | temporal locality and cache level | |
| IA-64 | cache line; at least 32 bytes | yes | temporal locality and cache level | base update form with implicit prefetch; cache control hints on load and store instructions |

| | | | | |
|---|---|---|---|---|
| MIPS | cache line | yes | temporal locality (streamed or retained) | |
| MMIX | specified number of bytes | yes | | |
| PA-RISC | cache line | yes | spatial locality | cache control hints on load and store instructions; pre/post increment (base update) forms of some load and store instructions |
| PowerPC | cache line | yes | | |
| SuperH | cache line; 16 bytes for SH-3, 32 bytes for SH-4 | no | | |
| SPARC | cache line; 64 bytes for UltraSPARC-II | yes | temporal locality | |
| XScale | cache line; 32 bytes | no | | |

## 3DNow!

The 3DNow! technology from AMD extends the x86 instruction set, primarily to support floating point computations. Processors that support this technology include Athlon, K6-2, and K6-III.

The instructions `PREFETCH` and `PREFETCHW` prefetch a processor cache line into the L1 data cache [1]. The first prepares for a read of the data, and the second prepares for a write.

There are no alignment restrictions on the address. The size of the fetched line is implementation dependent, but at least 32 bytes.

The Athlon processor supports `PREFETCHW`, but the K6-2 and K6-III processors treat it the same as `PREFETCH`. Future AMD K86 processors might extend the `PREFETCH` instruction format.

## Alpha

The Alpha architecture supports data prefetch via load instructions with a destination of register `R31` or `F31`, which prefetch the cache line containing the addressed data [2][3]. Instruction `LDS` with a destination of register `F31` prefetches for a store.

| | |
|---|---|
| `LDBU`, `LDF`, `LDG`, `LDL`, `LDT`, `LDWU` | Normal cache line prefetches. |
| `LDS` | Prefetch with modify intent; sets the dirty and modified bits. |
| `LDQ` | Prefetch, evict next; no temporal locality. |

Addresses used for prefetch should be aligned to prevent alignment traps.

Data prefetch instructions are ignored on pre-21264 implementations of Alpha.

The Alpha architecture also defines the following instructions [2]:

| | |
|---|---|
| `FETCH` | Prefetch Data |

| | |
|---|---|
| `FETCH_M` | Prefetch Data, Modify Intent |

These instructions are meant to help with very long memory latencies and are not useful on existing Alpha implementations (through 21264).

## AltiVec

Data prefetch support in the AltiVec instruction set architecture is quite different from that of other architectures that GCC supports. Rather than prefetching a single block of data, it prefetches a *data stream* made up of the following elements [4].:

| | |
|---|---|
| EA | the effective address of the first unit in the sequence; there are no alignment restrictions |
| unit size | the number of quad words *(16 bytes?)* in each unit; between 0 and 31 |
| count | the number of units in the sequence; between 0 and 255 |
| stride | the number of bytes between the effective address of one unit and the effective address of the next unit in the sequence; this can be negative, but should not be smaller than 16 bytes |

The instructions that operate on these data streams are:

| | |
|---|---|
| `dst` | (Data Stream Touch); data marked as most recently used (temporal locality |
| `dstst` | (Data Stream Touch for Store); data marked as most recently used (temporal locality) |
| `dstt` | (Data Stream Touch Transient); data marked as least recently used (no temporal locality) |
| `dststt` | (Data Stream Touch Transient for Store); data marked as least recently used (no temporal locality) |
| `dss` | (Data Stream Stop); stop a data stream if no more data from it is needed |
| `dssall` | (Data Stream Stop All); stop all data streams |

A prefetch instruction specifies one of four data streams, each of which can prefetch up to 128K bytes, 12K bytes in a contiguous block. Reuse of a data stream aborts prefetch of the current data stream and begins a new one. The data stream stop instructions can be used when data from a stream is no longer needed, for example for an early exit of a loop processing array elements.

Additional AltiVec instructions for cache control are `lvxl` (Load Vector Indexed LRU) and `stvxl` (Store Vector Indexed LRU), which indicate that an access is likely to be the final one to a cache block and that the address should be treated as least recently used, to allow other data to replace it in the cache.

The differences between AltiVec's cache control instructions and The PowerPC instructions `dcbt` and `dcbtst` are discussed in section 5.2.1.7 of [4].

GCC data prefetch support for AltiVec could use the PowerPC prefetch support, which fits into the prefetch framework. Using a constant unit size and always using a count of 1 would make a data stream touch behave like data prefetch instructions on other targets, allowing it to fit in GCC's data prefetch framework, but this would require specifying a data stream for each prefetch and keeping track of which ones are in use.

## IA-32 SSE

The IA-32 Streaming SIMD Extension (SSE) instructions are used on several platforms, including the Pentium III and Pentium 4 [6] and IA-32 support on IA-64 [8]. The SSE prefetch instructions are included in the AMD extensions to 3DNow! and MMX used for x86-64 [5].

The SSE `prefetch` instruction has the following variants:

| | |
|---|---|
| `prefetcht0` | Temporal data; prefetch data into all cache levels. |
| `prefetcht1` | Temporal with respect to first level cache; prefetch data in all cache levels except 0th cache level. |
| `prefetcht2` | Temporal with respect to second level cache; prefetch data in all cache levels, except 0th and 1st cache levels. |
| `prefetchnta` | Non-temporal with respect to all cache levels; prefetch data into non-temporal cache structure, with minimal cache pollution. |

There are no alignment requirements for the address. The size of the line prefetched is implementation dependent, but a minimum of 32 bytes.

### IA-64

The `lfetch` (Line Prefetch) instruction has versions for read and write prefetches, and an optional modifier to specify the locality of the memory access and the cache level to which the data would best be allocated [8].

The possible values for the locality hint are:

| | |
|---|---|
| none | Temporal locality for cache level 1 and higher (all levels). |
| `nt1` | No temporal locality for level 1, temporal for level 2 and higher. |
| `nt2` | No temporal locality for level 2, temporal for levels above 2. |
| `nta` | No temporal locality, all levels |

There are two base update forms of `lfetch`, which increment the register containing the address and then implicitly prefetch the new address, as well as the original address. The increment value is either in a second general register or is an immediate value.

Line size is implementation dependent; it is a power of 2, at least 32.

Load and store instructions can also be used to prefetch data. The base update forms of these instructions imply a prefetch, and have a completer that specifies the locality of the memory access.

### MIPS

The `PREF` (Prefetch) instruction, supported by MIPS32 [9] and MIPS64 [10], takes a hint with one of the following values:

| | |
|---|---|
| `load` | data is expected to be read, not modified |
| `store` | data is expected to be stored or modified |
| `load_streamed` | data is expected to be read but not reused |
| `store_streamed` | data is expected to be stored but not reused |
| `load_retained` | data is expected to be read and reused extensively |
| `store_retained` | data is expected to be stored and reused extensively |
| `writeback_invalidate` | data is no longer expected to be used |

| | |
|---|---|
| `PrepareForStore` | prepare the cache for writing an entire line |

The "streamed" versions place the prefetched data into the cache in such a way that it will not displace data prefetched as "retained". The "retained" versions place the data in the cache so that it will not be displaced by data prefetched as "streamed."

The prefetch moves a block of data into the cache. The size is implementation specific.

There are no alignment restrictions.

The `PREFX` (Prefetch Indexed) instruction, supported by MIPS64, differs in the addressing mode and is for use with floating point data.

## MMIX

MMIX has the following data prefetch instructions [11][12]:

| | |
|---|---|
| `PRELD` | preload a specified number of bytes of data |
| `PRELDI` | preload data immediate |
| `PREST` | prestore (prefetch for write) a specified number of bytes of data |
| `PRESTI` | prestore data immediate |

There are also load and store instructions, `LDUNC` and `STUNC`, which request that the data not be cached because it is unlikely to be accessed again soon.

## PA-RISC

A normal load to register `GR0` prefetches data. The data prefetch instructions are [13]:

| | |
|---|---|
| `LDW` | Prefetch cache line for read. |
| `LDD` | Prefetch cache line for write. |

Prefetch and cache control are also supported for accesses of semaphores.

Some load and store instructions modify the base register, providing either pre-increment or post-increment, and some provide a cache control hint; a load instruction can specify spatial locality, and a store instruction can specify block copy or spatial locality. The spatial locality hint implies that there is poor temporal locality and that the prefetch should not displace existing data in the cache. The block copy hint indicates that the program is likely to store a full cache line of data.

There are no alignment requirements on the address of prefetched data; the low order part of the address is ignored.

## PowerPC

The PowerPC provides the following data prefetch instructions [14]:

| | |
|---|---|
| dcbt | Data Cache Block Touch |
| dcbtst | Data Cache Block Touch for Store |

There are no alignment restrictions on the address of the data to prefetch.

### SuperH

The SuperH RISC engine architecture defines the `PREF` (Prefetch Data to the Cache) instruction.

For the SH-3, the address should be on a longword boundary. The number of bytes prefetched is 16 [16].

For the SH-4, the instruction moves 32 bytes of data starting at a 32-byte boundary into the operand cache [17].

### SPARC

The SPARC version 9 instruction set architecture defines the `PREFETCH` (Prefetch Data) and `PREFETCHA` (Prefetch Data from Alternate Space) [15] instructions, whose variants are specified by the *fcn* field:

| | | |
|---|---|---|
| 0 | prefetch for several reads | Move the data into the cache nearest the processor (high degree of temporal locality). |
| 1 | prefetch for one read | Prefetch with minimal disturbance to the cache (low degree of temporal locality). |
| 2 | prefetch for several writes (and possibly reads) | Gain exclusive ownership of the cache line (high degree of temporal locality). |
| 3 | prefetch for one write | Prefetch with minimal disturbance to the cache (low degree of temporal locality). |
| 4 | prefetch page | Shorten the latency of a page fault. |

UltraSPARC-I treats these instructions as nops [18]. UltraSPARC-II and UltraSPARC-IIi support them by mapping the variants listed above onto two variants for read and write prefetch with no or low temporal locality [19][20].

There are no alignment restrictions on the address to prefetch; the instructions ignore the 5 least significant bits.

### XScale

The Intel XScale processor includes ARM's DSP-enhanced instructions, including the `PLD` (Preload) instruction. This instruction prefetches the 32-byte cache line that includes the specified data address.

NOTE: More investigation is necessary; [23] has an example that implies that base update might be available.

## References

These references need cleanup and should actually be used in the text above that uses the information. Many of the links will likely be out of date soon, but they'll stay here until the initial rush of prefetch work is done.

References to cache control instructions for specific architectures:

[1] *3DNow![tm] Technology Manual*, AMD, 29128G/0, March 2000.

[2] *Alpha Architecture Handbook*, Compaq, Version 4, October 1998, Order Number EC-QD2KC-TE; see pages 4-139 and A-8.

[3] *Alpha 21264 Hardware Reference Manual*, July 1999; see section 2.6.

[4] *AltiVec Technology Programming Environments Manual*, 11/1998, Rev. 0.1; Page 5-9 has usage recommendations.

[5] *AMD Extensions to the 3DNow![tm] and MMX[tm] Instruction Sets*, AMD, Publication 22466D, March 2000.

[6] *The IA-32 Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*.

[8] *Intel Itanium[tm] Architecture Software Developer's Manual Vol. 3 rev. 2.1: Instruction Set Reference*.

[9] *MIPS32[tm] Architecture for Programmers; Volume II: The MIPS32[tm] Instruction Set*, MIPS Technologies, Document Number MD00086, Revision 0.95, March 12, 2001 search from www.mips.com.

[10] *MIPS64[tm] Architecture for Programmers; Volume II: The MIPS64[tm] Instruction Set*, MIPS Technologies, Document Number MD00087, Revision 0.95, March 12, 2001; search from www.mips.com.

[11] MMIX Op Codes, Don Knuth.

[12] *The Art of Computer Programming, Fascicle 1: MMIX*, Don Knuth, Addison Wesley Longman, 2001; http://www-cs-faculty.stanford.edu/~uno/fasc1.ps.gz.

[13] *PA-RISC 2.0 Instruction Set Architecture*; see *Memory Reference Instructions* in Chapter 6.

[14] *PowerPC Microprocessor 32-bit Family: The Programming Environments*, page 5-8.

[15] *The SPARC Architecture Manual*, Version 9, SPARC International, SAV09R1459912, 1994-2000; see A.42.

[16] *SuperH[tm] RISC Engine SH-3/SH-3E/SH3-DSP Programming Manual*, ADE-602-096B, Rev. 3.0, 9/25/00, Hitachi, Ltd.

[17] *SuperH[tm] RISC Engine SH-4 Programming Manual*, ADE-602-156D, Rev. 5.0, 4/19/2001, Hitachi, Ltd.

[18] *UltraSPARC[tm] User's Manual*, Sun Microsystems, Part No: 802-7720-02, July 1997, pages 36-37.

[19] *UltraSPARC[tm]-II High Performance 64-bit RISC Processor*, Sun Microelectronics Application Notes, section 5.0: Software Prefetch and Multiple-Outstanding Misses

[20] *UltraSPARC[tm]-IIi User's Manual*, Sun Microsystems, Part No: 805-0087-01, 1997.

References to uses of data prefetch instructions:

[21a] *Compiler Writer's Guide for the Alpha 21264*, Order Number EC-RJ66A-TE, June 1999.

[21b] *Compiler Writer's Guide for the 21264/21364*, Order Number EC-0100A-TE, January 2002.

[22] *Compiler-Based Prefetching for Recursive Data Structures*, Chi-Keung Luk and Todd C. Mowry, linked from https://www.toddcmowry.org/publications-1/. That location also has links to several other papers about data prefetch by Todd C. Mowry.

[23] *Intel(r) XScale[tm] Core Developer's Manual*, December 2000; section A.4.4 is "Prefetch Considerations" in the Optimization Guide.

[24] *Optimizing 3DNow! Real-Time Graphics*, Dr. Dobb's Journal August 2000, Max I. Fomitchev.

[25] *An Overview of the Intel IA-64 Compiler*, Carole Dulong, Rakesh Krishnaiyer, Dattatraya Kulkarni, Daniel Lavery, Wei Li, John Ng, and David Sehr, all of Microcomputer Software Laboratory, Intel Corporation, *Intel Technology Journal*, 4th quarter 1999.

[26] *UltraSPARC[tm]-II Enhancements: Support for Software Controlled Prefetch*, Sun Microsystems, July 1996, WPR-0002.