Cortex-R82

Performance boost in powerful real-time Cortex-R processor using data prefetch control
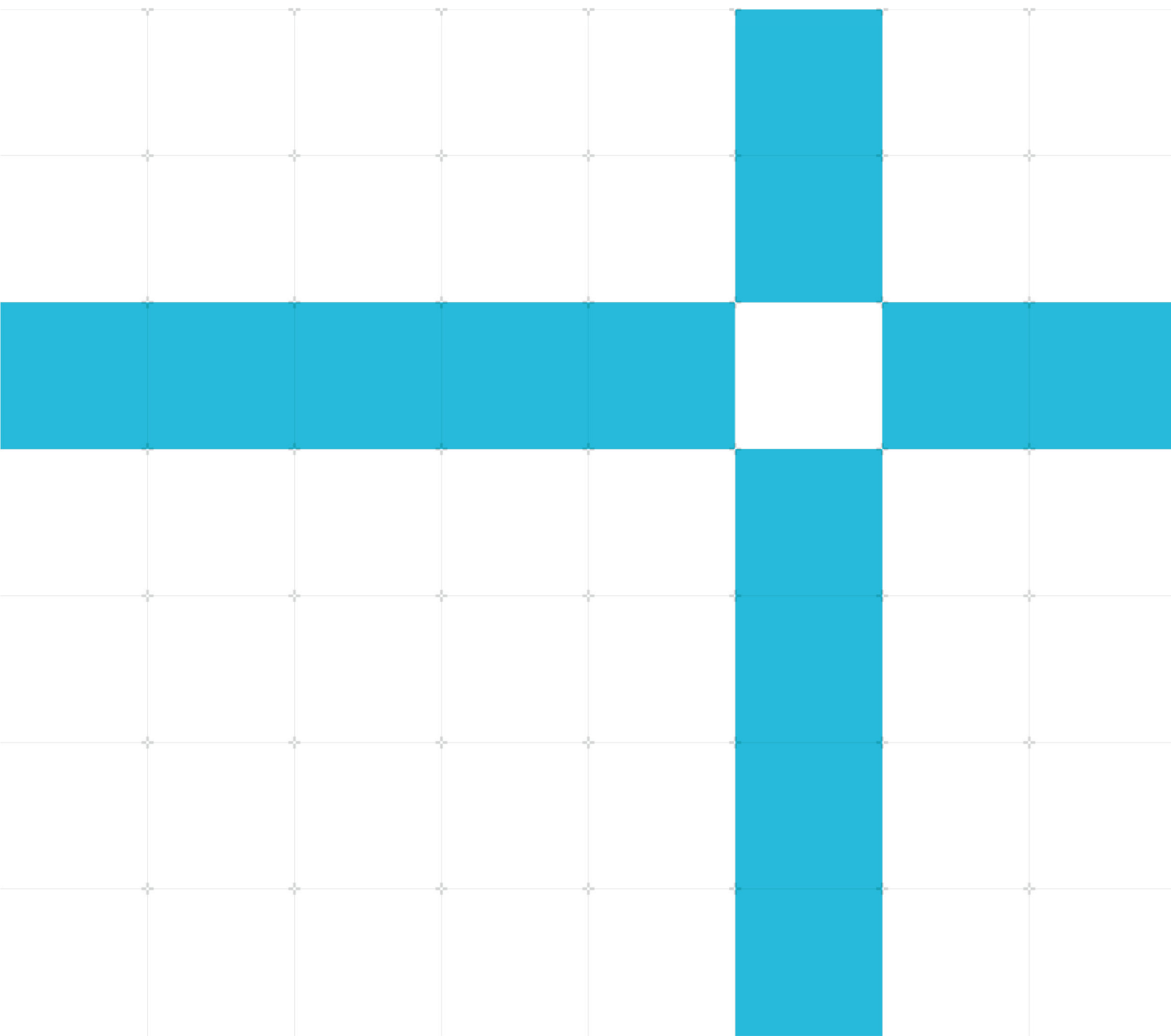
Non-Confidential

Issue 1.0

108646

Cortex-R82

Performance boost in powerful real-time Cortex-R processor using data prefetch control

Release information

Document history

Issue	Date	Confidentiality	Change
1.0	04-JUL-2023	Non-Confidential	First release

Non-Confidential Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on Cortex-R82, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change. We believe that this document contains no offensive language. To report offensive language in this document, email **terms@arm.com**.

Contents

1. Introduction

1.1. Product revision status

The rxpy identifier indicates the revision status of the product described in this book, for example, r1p2, where:

rx   identifies the major revision of the product, for example, r1.

py   identifies the minor revision or modification status of the product, for example, p2.

1.2. Intended audience

This document is for system designers, system integrators, and programmers who are designing or programming a System on Chip (SoC) that uses the Cortex®-R82 processor.

1.3. Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: **https://developer.arm.com/glossary**.

Typographical conventions

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace` **bold**	Language keywords when used outside example code.
`monospace` <u>underline</u>	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: ``` MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2> ```

Convention	Use
SMALL CAPITALS	Terms that have specific technical meanings as defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
Caution	Recommendations. Not following these recommendations might lead to system failure or damage.
Warning	Requirements for the system. Not following these requirements might result in system failure or damage.
Danger	Requirements for the system. Not following these requirements will result in system failure or damage.
Note	An important piece of information that needs your attention.
Tip	A useful tip that might make it easier, better, or faster to perform a task.
Remember	A reminder of something important that relates to the information you are reading.

1.4.   Useful resources

This document contains information that is specific to this product. See the following resources for other relevant information.

- Arm Non-Confidential documents are available on developer.arm.com/documentation. Each document link in the tables below provides direct access to the online version of the document.

- Arm Confidential documents are available to licensees only through the product package.

Arm products	Document ID	Confidentiality
Cortex-R82 Technical Reference Manual	101548_0002_06_en	Non-Confidential
Arm GNU Toolchain	-	Non-Confidential

Arm architecture and specifications	Document ID	Confidentiality
Armv8, for Armv8-R AArch64 architecture profile	062922	Non-Confidential

Non-Arm resources	Document ID	Organization
ConsumerBench™1.1 ™	Version 1.1	EEMBC
OABench™1.1 ™	Version 1.1	EEMBC
STREAM	-	University of Virginia

Note

Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at **http://www.adobe.com**.

2. Abstract

High-performance processors employ hardware data prefetching to reduce the negative
performance impact of large main memory latencies. An effective prefetching mechanism can
improve cache hit rate significantly. Data prefetching boosts the execution performance by fetching
data before it is needed. While prefetching improves performance substantially on many programs, it
can significantly reduce performance on others. Therefore, even though Cortex-R82 has some
control over the prefetchers, how it should be enabled is an implementation choice. This can be done
by using different control options to configure the prefetcher capabilities. In this paper, the
investigation has been done on prefetcher capabilities of Cortex-R82 with a baseline where no
prefetching is present. The investigation demonstrates the performance of the Cortex-R82
prefetcher in the following domains: no prefetching, minimum prefetching (conservative), moderate
prefetching, and maximum prefetching (aggressive). This study also illustrates the intensity of
prefetching required across different application areas using various industry standard benchmarks
to measure the performance of the processor.

2.1. Authors

Anisha Hazra [CPU Performance Verification Engineer]

Rajendra Kumar Saini [Principal Engineer]

3. Overview

Prefetching is a technique of fetching data or instruction from the main memory and storing it in cache memory before it is required. Prefetching data and then accessing it from caches is usually much faster than accessing it directly from main memory. Data prefetching fetches data before it is needed and instruction prefetching fetches instructions before they need to be executed. A hardware prefetcher is responsible for generating prefetches targeting one or more levels of cache hierarchy. In the Cortex-R82 processor the CPU Auxiliary Control Register provides the implemented configuration and control options for the processor. Therefore, the prefetcher can be controlled using this register to evaluate the performance. We have focused our analysis on those benchmarks where the scope of improving the performance is maximum even if the prefetcher is controlled moderately.

4. Data prefetcher

The Cortex-R82 processors have optional instruction and data caches which are independently implemented in the core. The processor includes both instruction side and data side prefetchers to improve the cache hit rate, and therefore, the performance. The prefetchers can anticipate several variants of cache access patterns, and request linefills to the L1 caches and to the shared L2 cache.

The data prefetch mechanism looks for cache line fetches with regular patterns. If the data prefetcher detects a pattern, then it signals to the memory system that memory accesses from a specified address are likely to occur soon. The memory system responds by starting new linefills to fetch the predicted addresses ahead of the instruction stream.

5. Programming the prefetcher in software

The DPEN, DPDEPTH and DPL2 fields of IMP_CPUACTLR_EL1 register of Cortex-R82 are used to enable/disable the L1 data cache prefetcher, configure its depth and determine whether prefetches to the L2 cache are permitted. For more details about Cortex-R82 data prefetchers read **Cortex-R82 Technical Reference Manual**.

5.1. Syntax to read IMP_CPUACTLR_EL1

The Cortex-R82 IMP_CPUACTLR_EL1 can be read using MRS and can be written using MSR with the following syntaxes:

```
MRS <Xt>, IMP_CPUACTLR_EL1
MSR IMP_CPUACTLR_EL1, <Xt>
```

This syntax is encoded with the settings in the instruction encoding as shown in Table 1 below.

op0	op1	CRn	CRm	op2
0b11	0b000	0b1111	0b0001	0b000

Table 1: Instruction encoding to access the CPU Auxiliary Control Register

5.2. Examples to program the prefetcher

Below are some examples that illustrates different scenarios of programming the prefetcher in bootcode.

Example 1 - Disable the L1 data cache prefetcher

```
MRS    x0,s3_0_c15_c1_0
BIC    x0, x0, #0xC0000000   //Clear DPEN or set bits [31:30] as 0b00
MSR    s3_0_c15_c1_0,x0
```

Example 2 - Enable L1 data cache prefetcher without power throttling

```
MRS     x0,s3_0_c15_c1_0
ORR      x0, x0, #(0x1 << 30)
BIC    x0, x0, #0x80000000 //Clear DPEN or set bits [31:30] as 0b01
MSR     s3_0_c15_c1_0,x0
```

Example 3 - Enable L1 prefetcher without power throttling along with minimum depth

```
MRS     x0,s3_0_c15_c1_0
BIC      x0, x0, #0x38000000 //Clear DPDEPTH or set bits [29:27] as 0b000
ORR     x0, x0, #(0x1 << 27)   //Set DPDEPTH at 0b001 or bit [27] as 0b1
MSR     s3_0_c15_c1_0,x0
```

IMP_CPUACTLR_EL1 can be programmed in a similar way to achieve higher number of outstanding prefetches by using appropriate value of DPDEPTH.

Refer Table 2 to understand more about prefetch enable options and controllability on the prefetcher depth that has been used in the paper. In Table 2, DEFAULT indicates the state where no additional programming is done to the prefetchers.

Configuration	DPEN [31:30]	DPDEPTH [29:27]	DPL2 [26:25]	Function
• L1 Prefetcher - Disabled • L2 Prefetcher - Disabled	0b00	L1 Prefetcher Disabled	0b00	• Both L1 and L2 Prefetchers turned off • Used as baseline for other configurations
• L1 Prefetcher - Enabled with minimum prefetching • L2 Prefetcher - Disabled	0b01	0b000	0b00	• Prefetcher enabled without power-aware throttling • Up to 1 outstanding L1 prefetch
• L1 Prefetcher - Enabled with moderate prefetching • L2 Prefetcher - Disabled	0b01	0b100 (DEFAULT)	0b00	• Prefetcher enabled without power-aware throttling • Up to 5 outstanding L1 prefetches
• L1 Prefetcher - Enabled with maximum prefetching • L2 Prefetcher - Disabled	0b01	0b111	0b00	• Prefetcher enabled without power-aware throttling • Up to 5 outstanding L1 prefetches • Do not prefetch into the L2
• L1 Prefetcher - Enabled with maximum prefetching • L2 Prefetcher - Enabled with maximum prefetching	0b01	0b111	0b11 (DEFAULT)	• Prefetcher enabled without power-aware throttling • Up to 5 outstanding prefetches • Initial L2 prefetch is 32 lines ahead of the L1 prefetch

Table 2 : Prefetch enable options and controllability on prefetcher depth

6. Hardware and software configurations

The Cortex-R82 system is built with L1 cache as 32KB, L2 cache is 1MB and NIC-400 as the Interconnect. For improved performance the Cortex-R82 has an optional L2 cache shared across the cores of a cluster. Data bus size is 128 bits and DRAM Bandwidth is 32 GB/s. Frequency of the core, interconnect, and memory controller are maintained at 1:1:1 ratio with the load to use latency as 188 CPU Cycles.

The performance data has been generated on Emulation platform. All the benchmark binaries are bare-metal application-based binaries. The binaries are built with GNU Arm Embedded Toolchain version 10.3-2021.10 using optimizations as -Omax -fno-lto. For more details about the features and release notes of the toolchain read **Arm GNU Toolchain**.

The benchmark used in this study are mainly EEMBC (Consumer and Office sub suites) and JMC Stream. The EEMBC-Consumer benchmark suite includes benchmarks hpg (High Pass Grey-Scale Filter), cjpeg/djpeg (JPEG Compression and Decompression), cmy (RGB to CMYK Conversion), yiq (RGB to YIQ Conversion). The EEMBC-Office benchmark suite includes dither (Dithering Benchmark), rotate (Image Rotation Benchmark), text (Text Processing Benchmark). JMC Stream benchmarks includes copy, scale, sum and triad operations with 16 MB array size. For more details about the benchmarks used, please read **ConsumerBench™1.1 ™, OABench™1.1 ™** and **STREAM** as found in the Useful Resources section.

7. Performance analysis with impact of prefetcher

7.1. L1 prefetcher enabled with minimum depth

To study the prefetcher performance, in this section the prefetcher is configured with minimum capability where L1 prefetcher is enabled with up to 1 outstanding prefetcher performance and L2 prefetcher is disabled. The results have been compared against the results obtained with both the prefetchers turned off which is the baseline. Figure 1 shows that even with minimum prefetches the performance has improved by a good margin.
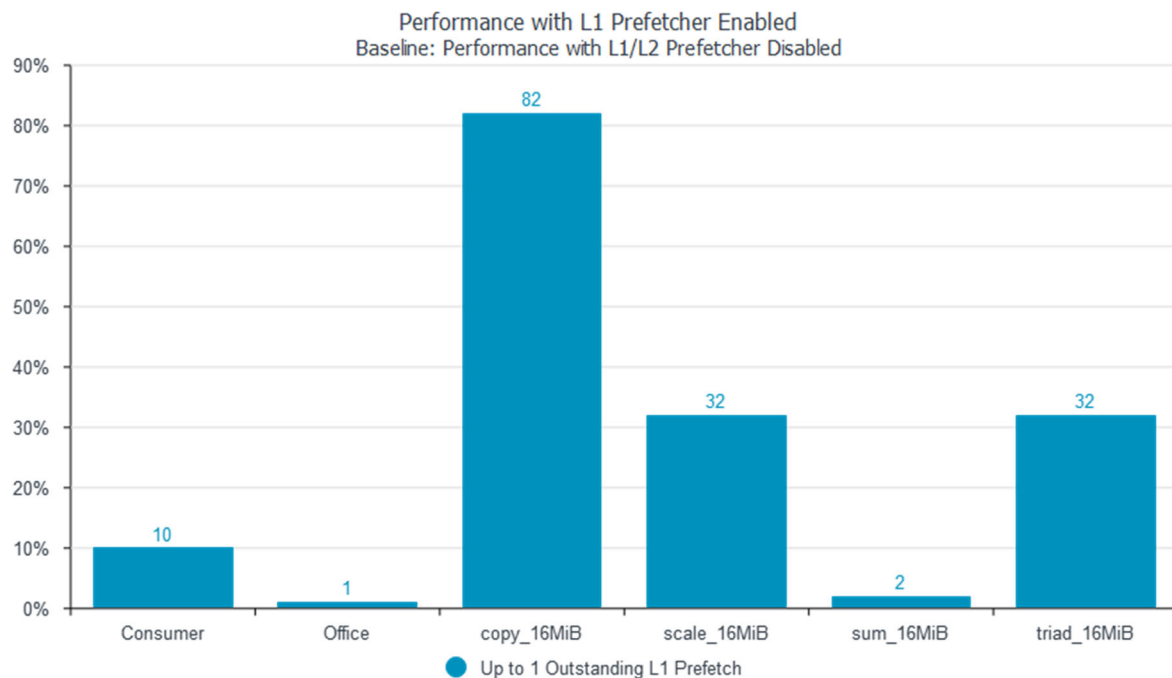

Figure 1 : Performance with L1 Prefetcher Enabled

The above data can be supported with Performance Monitoring Units such as L1D_CACHE_REFILL, L1D_CACHE etc. These PMUs can be used to derive the pattern of cache misses and cache hits. The impact in performance due to improved cache hit rate because of prefetching is explained in Figure 2 below.
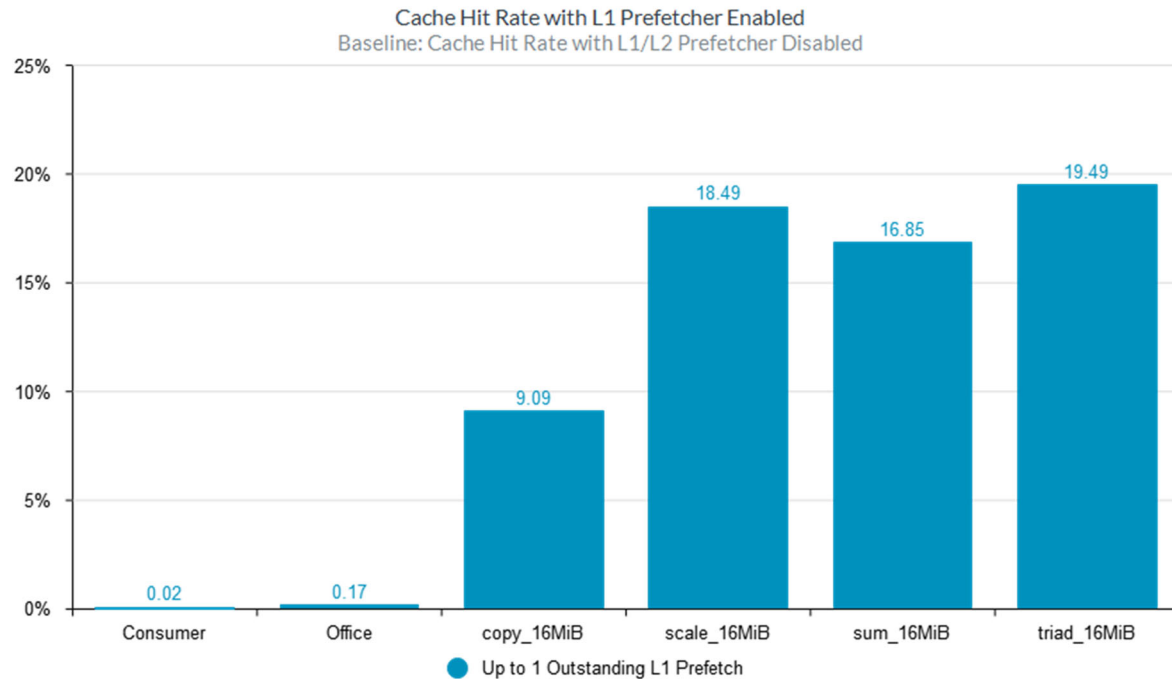
Figure 2 : Cache Hit Rate with L1 Prefetcher Enabled

7.2.   L1 prefetcher enabled with variable depth

This data shows the prefetcher performance with different possibilities of how the data prefetcher bits of the CPU Auxiliary Control Register can be controlled. In this section, the variation with L1 prefetcher depth is shown against the baseline data where L1/L2 prefetcher are turned off.

EEMBC Benchmark suites, consumer and office data is almost saturated with the increase in the outstanding L1 prefetches from minimum (conservative) to moderate till maximum (aggressive). Even with minimum prefetches the performance matches with the performance which is achieved with maximum outstanding prefetches. The moderate prefetching is also the case of default prefetching configuration in Cortex-R82.

With JMC-Stream benchmarks higher delta in performance can be observed with the increase in L1 prefetcher depth and the maximum performance is achieved with very aggressive prefetching.
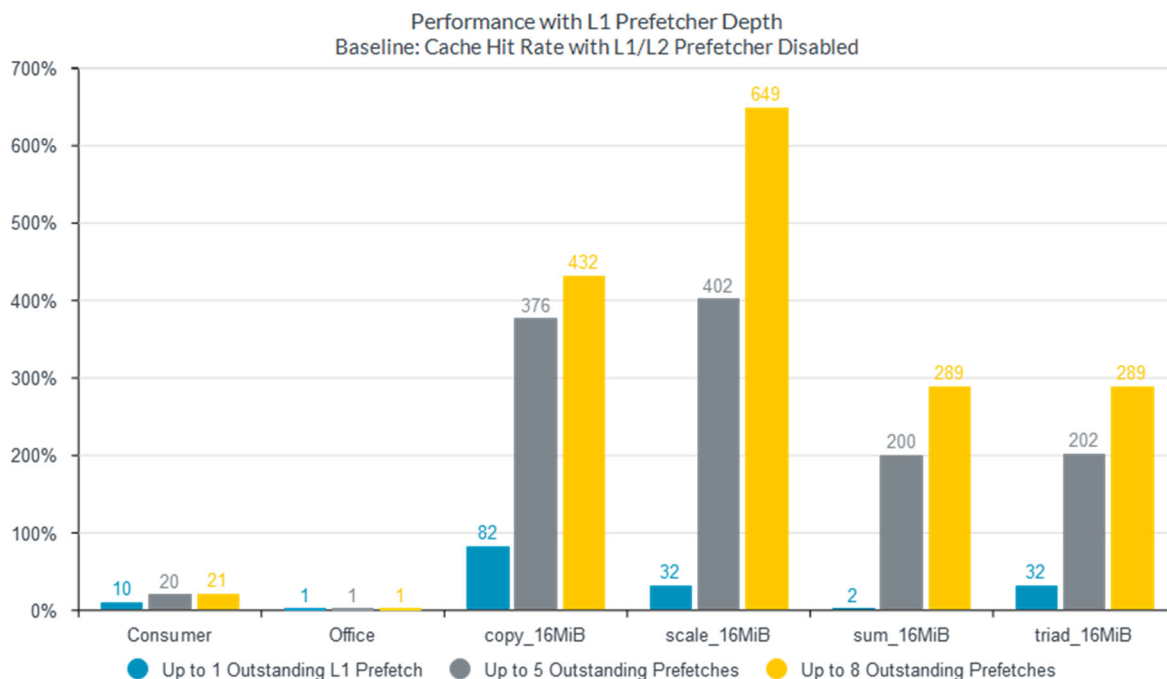
Figure 3 : Performance with L1 Prefetcher Depth

With consumer and office benchmarks cache hit rate is almost constant with the increase in
prefetcher depth. While in JMC Steam benchmarks there is a improvisation in cache hit rate in the
outstanding prefetches are increased from minimum to moderate. If number of outstanding
prefetches are increased further to maximum the cache hit rate saturates. Figure shows the cache hit
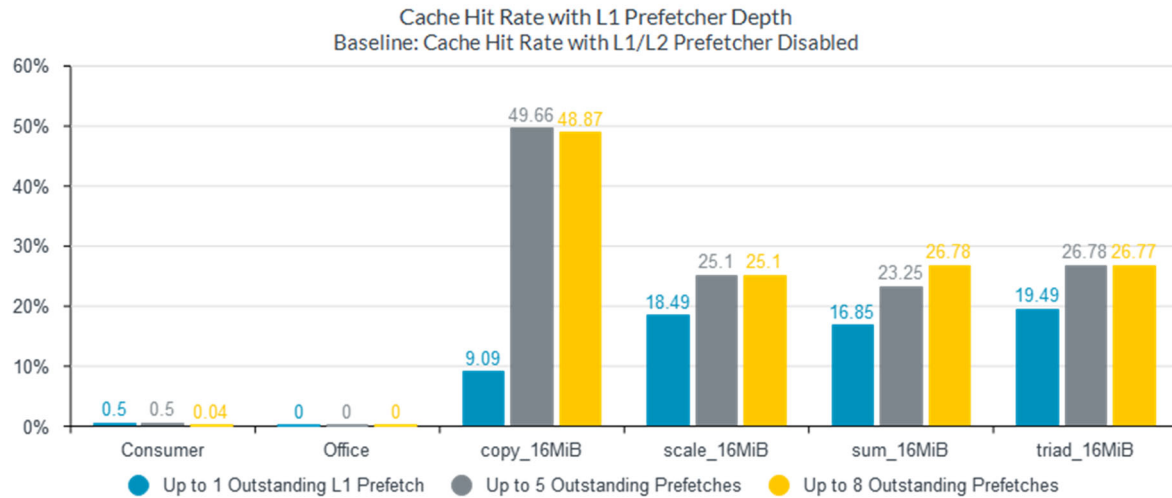rate study.

Figure 4 : Cache Hit Rate with L1 Prefetcher Depth

7.3.   L1 prefetcher enabled with maximum depth along with L2 prefetcher

This data represents the L1/L2 prefetcher performance in their maximum setting against the baseline where both L1/L2 prefetchers are turned off.

The EEMBC Suite consumer and office benchmarks does not show much impact when maximum capability of L2 prefetches is added along with maximum capability of L1 prefetches. While with JMC Stream benchmarks the performance is further enhanced if maximum level of L2 prefetches are added with 8 outstanding L1 prefetches.
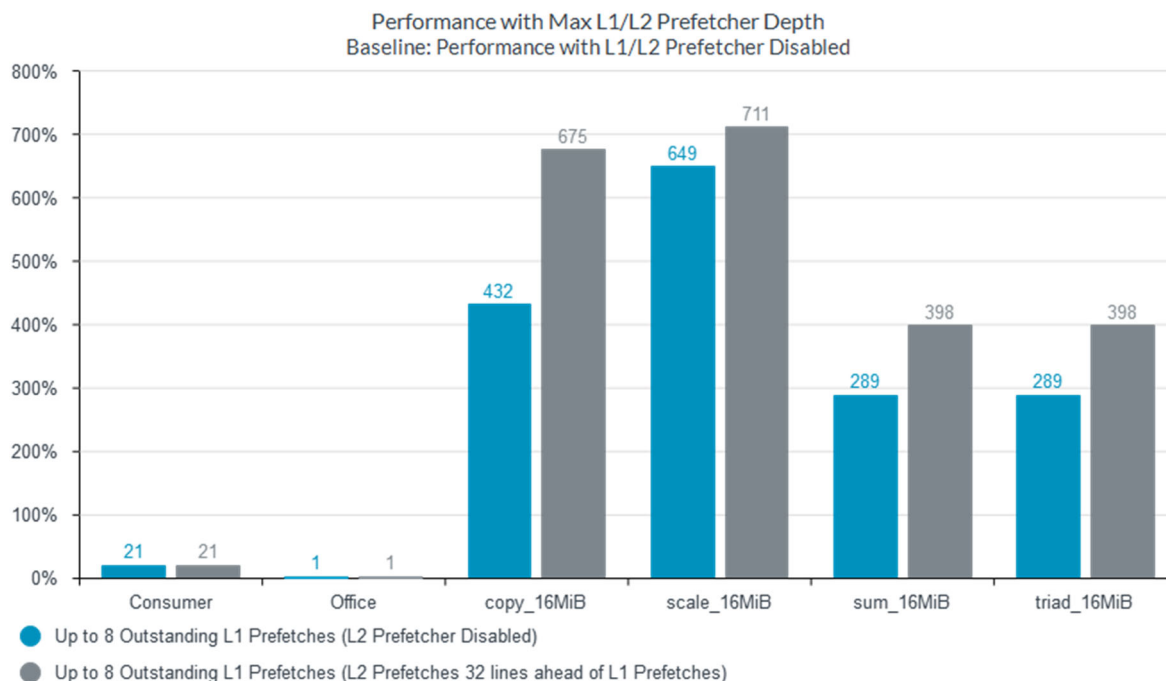
Figure 5 : Performance with maximum L1/L2 Prefetcher Depth

The cache hit rates are almost comparable in these two configurations, where maximum outstanding L1 prefetches are programmed with L2 prefetcher disabled and maximum outstanding L1 prefetches are programmed with L2 prefetches 32 lines ahead of L1 prefetches against the cache hit rates when both the prefetchers are turned off.
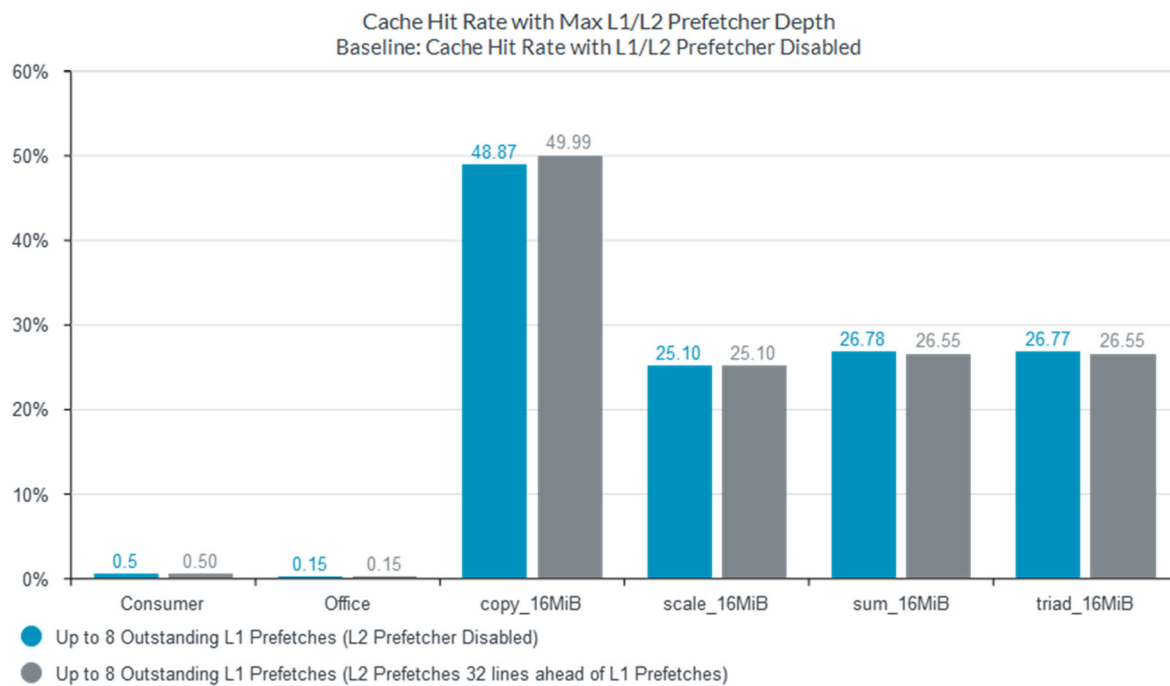
Figure 6 : Cache Hit Rate with maximum L1/L2 Prefetcher Depth

8. Constraints on prefetching

Prefetching can sometimes negatively impact the performance and energy consumption of a processor if the predicted memory addresses are not accurate. Such scenarios are caused because inaccurate prefetches result in unnecessary memory accesses.

Moreover, the prefetched data might displace cache lines that might be required later, by load/store instructions in the program. In such situation prefetching not only reduces performance but also wastes memory bandwidth by resulting in additional cache misses.

Apart from all the performance enhancements that we see, in some benchmarks, there could be significant reduction in performance as well. Therefore, increasing the aggressiveness of the hardware prefetcher irrationally can drastically reduce performance on several applications even though it might improve average performance of a processor.

9.   Conclusion

From the performance analysis, it can be concluded that with the appropriate selection of prefetcher configuration the performance can be improved with an expectation of stable power consumption.

The relative performance analysis data shows how powerfully Cortex-R82 prefetcher can scale the performance even with the minimum settings against the performance when prefetcher is turned off.

Our results indicate that in many scenarios, we need not use aggressive prefetching to achieve the highest performance. The default configuration usually meets the expectation, as shown in Table 2.

Appendix A. Revisions

This appendix describes the technical changes between released issues of this document.

Table A-1: Issue 1.0

Change	Location
First release	-