# Adaptive Regular Pattern Prefetcher for L1 Data Cache for Microprocessors with RISC-V ISA

Bogdana Tishuk[0000−0002−4599−3136] and Sergei Bykovskii[0000−0003−4163−9743]

ITMO University, 49 Kronverksky Pr., St. Petersburg, Russia
tishuk52@gmail.com, sergei_bykovskii@itmo.ru

**Abstract.** Due to high percent of data is represented as N - dimensional arrays or other array - based structures, especially in scientific applications, access to this data will have regular pattern, consequently regular pattern prefetchers work effectively for a wide range of tasks. Main advantages of regular pattern prefetchers are low off - chip traffic, hardware simplicity, consequently low power consumption, compact area on the chip and low incremental memory bus load, caused by prefetching. This paper introduces AR2P - adaptive regular pattern prefetcher. The key ideas of AR2P are adaptive prefetch depth that depends on spatial locality of data, late prefetch detection, and decreasing power consumption as compared with RPT prefetcher due to flip - flop switch reduce. We evaluate AR2P using cycle accurate single core MARSS - RISCV simulator and TACLE benchmark set. AR2P shows 95% coverage on the average on benchmarks with regular access pattern and 37% for complex patterns.

**Keywords:** RISC-V · prefetching · cache

## 1 Introduction

Instruction and Data prefetchers are critical modules of modern computing systems, due to DRAM-memory is a bottleneck and one memory access costs nearly 100 machine cycles, consequently cache misses significantly influence on task execution time. Prefetching data and instructions into cache before demand is designed to reduce cache misses and increase performance.

Now microprocessors usually contain several prefetchers, for caches of different levels and different memory access patterns. For example, Intel microarchitectures: Ivy Bridge, Broadwell and other have two prefetchers for L2 cache and two prefetchers for L1 data cache. For L1 data cache there is an adjacent line prefetcher and PC - localized prefetcher. ARM cores have similar prefetching subsystem.

Prefetchers can take place in memory subsystem as shown on Figure 1 as a variant.

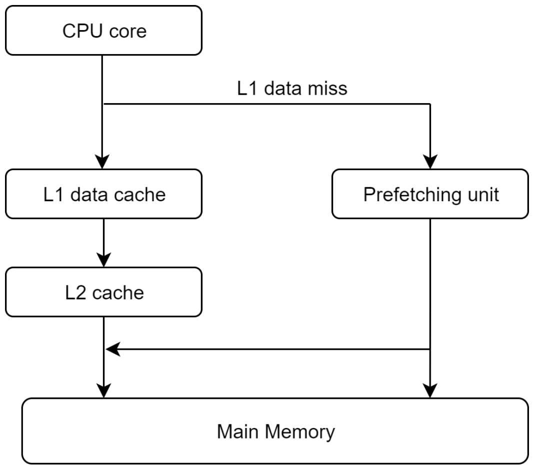


Figure 1 - Prefetchers in memory subsystem

Main questions, that arise in design process of L1 prefetching subsystem are:

- Is it possible to adapt prefetcher for various access patterns dynamically?
- What is optimal for designed system, large L1 cache and simple prefetcher, or small L1 cache and complex accurate prefetcher?
- Where is effective to keep prefetched data, in cache or in on-chip prefetch buffers?
- Which event triggers prefetching?
  - Cache misses
  - LLC access
  - Prefetced data hits
- How do conflict misses in not fully - associative caches depend on prefetching?

There is a trade off between accuracy, coverage and hardware complexity, although recent researches[5] show that high percentage of effective prefetches are accurately detected regular pattern prefetches, so high performance of prefetching subsystem can be reached by rather simple hardware.

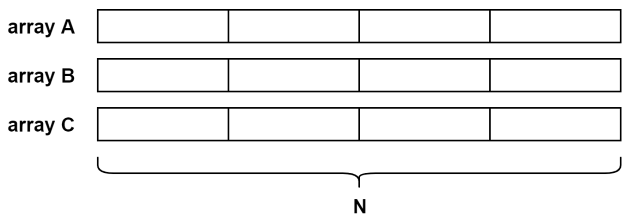
## 2 Related works

This section introduces analysis of various regular pattern prefetchers and correlated prefetchers for irregular patterns, that are used in today computing systems.

### 2.1 Regular pattern prefetchers

The simplest regular pattern prefetcher is next N line prefetcher. On triggering event on address X, prefetcher puts into cache memory N lines next to accessed line. If memory virtualization is used, prefetcher should stop on physical page bounds. On sequential access patterns like matrix operations with primitive data types or for instruction cache next N line prefetchers are rather effective. Jamison D. shows in his research that next N line prefetcher can have accuracy and coverage more 90 % on hydro and about 75% on mgrid benchmarks [1].

Other simple prefetcher is a stride prefetcher. On triggering event on address X, prefetcher puts into cache memory words from X + S to X + (S + N) addresses in common case, where S is the stride. Prefetcher detects if stride repeats necessary times, it predicts that next stride will be same and prefetches data. In common case, stride prefetchers are effective on operations with arrays of structures and matrix rows and columns. Now microprocessors usually use RPT prefetcher. To introduce RPT prefetcher consider situation, presented on Figure 2.



```
for ( int i = 0; i < N; i++)
    C[i] = A[i] + B[i]
```

Figure 2 - Accesses arrays in cycle

There is a cycle, where each element of array C should be summed up with appropriate elements of arrays A and B. On each step of the cycle A[i], B[i], C[i] are fetched:
Stride 1 : address A[i] - address C[i-1]
Stride 2 : address B[i] - address A[i]
Stride 3 : address C[i] - address B[i]

Due to this stride is various, consequently clearly discernible stride does not exist and stride prefetcher is ineffective here. The solution to the problem is an on chip table, where prefetcher keeps last memory access address , detected stride and confidence counter. On triggering event for defined PC current access address is compared with last address. If stride is the same, confidence counter updates, else stride field of table updates with new value. If confidence counter value is more than reference value, prefetch occurs on current address with added stride. Main metrics: accuracy - percent of prefetch hit and coverage - percent of cache misses, prevented by prefetcher, are 90% for art benchmark, and 30 % and 23 % for gcc [3].

## 2.2 Correlated prefetchers

Correlated prefetchers are effective on non-stride or irregular memory access patterns as processing linked list or graph elements. Common principle of correlated prefetcher is memorizing sequential streams of memory accesses, and on prefetching next stream part on triggering event.

**Markov Table.** Markov table is one of early solutions to organize temporary correlated access pairs spatially, by table, indexed by physical address[2]. Each table entry is fix - sized, but correlated streams can variate from two memory words to several MBytes. Consequently Markov Table is prone to inner fragmentation and loss of useful metadata due to fact that short streams waste space in Markov table, long streams are truncated.

**Domino** Domino prefetcher[6] is accurate correlated prefetcher for L1 data cache. Domino keeps metadata about correlated L1 data cache misses in two on - chip miss history tables MHT1 and MHT2. Also Domino keeps information about two previous L1 data cache misses in runtime.

| | | |
|---|---|---|
| tag | valid | prediction |

Figure 3 - MHT1 entry

MHT2 holds tags of two previous misses and is indexed as by exclusive or of current and previous L1 data cache miss. On L1 data cache miss Domino looks up entry with the same tags of misses in MHT2 and prefetches data on prediction address. If entry in MHT2 does not exist, Domino updates MHT2 with tags of two previous misses and current miss as prediction, and looks up entry in MHT1. If MHT1 entry exists, Domino prefetcher data on prediction address, if entry is not available, Domino updates MHT1.

**MISB** ARM introduced Managed Irregular Stream Buffer (MISB) prefetcher in 2019. The main idea of MISB is to create a new structural address space in which correlated physical addresses are kept in sequence. The key point is that in this structural address space, streams of correlated memory addresses are both temporally ordered and spatially ordered. The mapping from physical to structural addresses and vise versa is performed at a cache line granularity by two spatially indexed on-chip address metadata caches whose contents can be easily synchronized with that of the TLB[2].Unlike ISB, introduced in 2013 , MISB has inner stride prefetcher for metadata caches, Bloom filter, to filter requests to off - chip mapping tables when mappings do not exist in mapping tables, and finer mechanism of interaction with TLB. MISB shows significant speedup on Libquantum and SoPlex - 95 % and 65% , on gcc about 20% [2]. MISB is an effective algorithm to process irregular access patterns, but hardware complexity of MISB is significantly higher in comparison to other prefetchers.

## 3 Proposal

Today's microprocessors usually have separate L1 caches for instructions and data. If there are no branches, instruction are fetched sequentially from memory, consequently for L1 instruction cache next N line prefetcher can be effectively used with branch predictor interaction.

For L1 data cache there is a solution to modify localized PC data prefetcher by improving prefetch stride scale and prefetch depth prediction mechanism for regular patterns. The second modification is unlike RPT prefetcher updates Last address field for all entries of it's on-chip metadata table, our solution updates Last address for specified PC entry. This reduces power consumption due to field last address flip-flop switch decrease.

In common case for stride prefetcher if stride is close to the current demand, next demand may occur before necessary data be prefetched, that defeats purpose of prefetching. If stride is large, prefetching occurs too early, prefetched data can be evicted from cache. RPT prefetcher potentially suffers of late prefetches. This problem can be solved by detecting that demand was occurred before data was prefetched. If this repeats several times sequentially, prefetcher decides to increment stride . If stride doesn't change for several times, prefetcher assumes that it is a vector or matrix, calculates prefetch depth. Organization of prefetcher is shown on Figure 4.

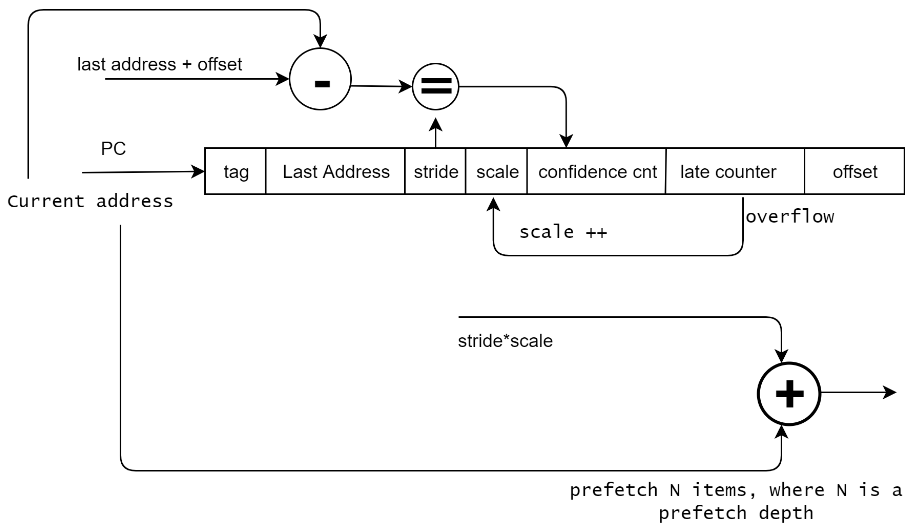
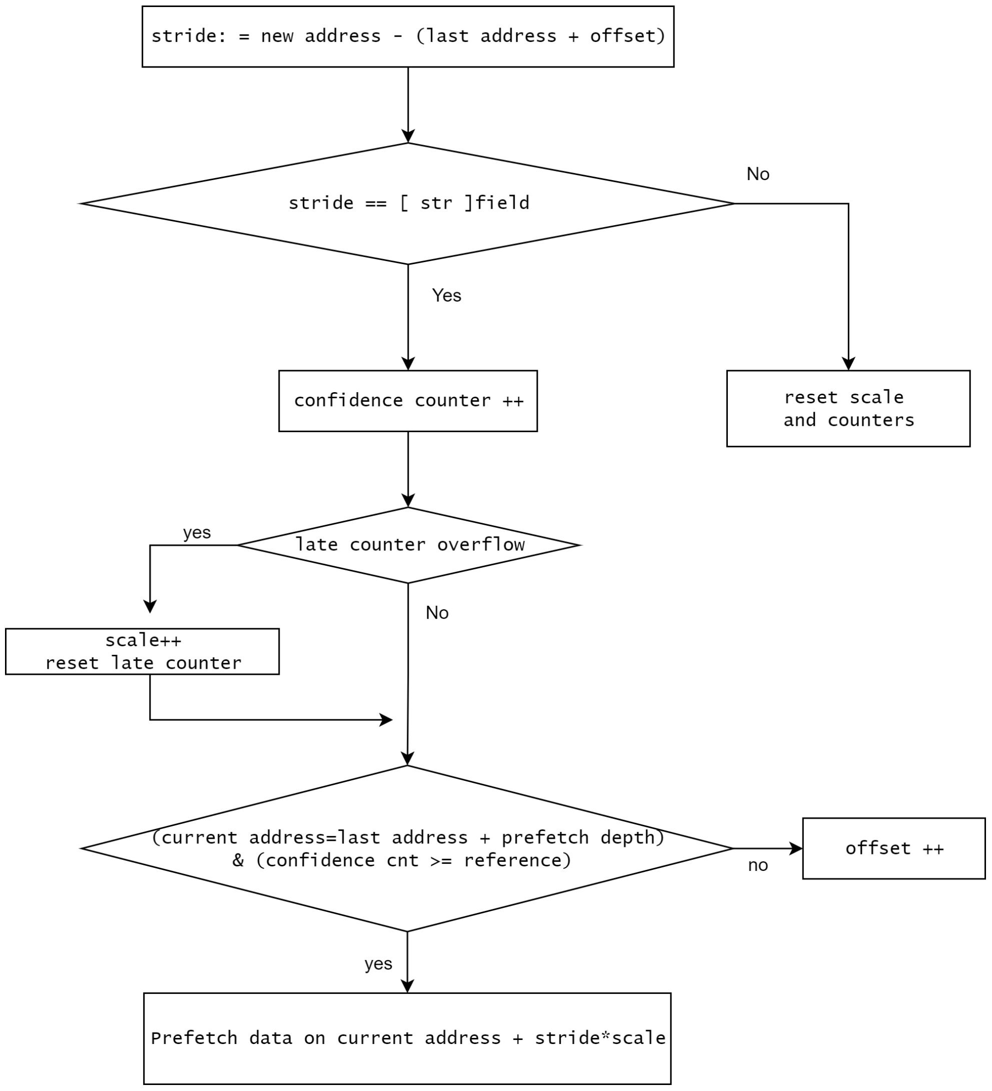


Figure 4 - Functional scheme of proposed prefetcher

At the first moment stride is detected as difference between last address base + offset and current address. Offset indicates position in prefetched vector, when prefetch depth more than 1. Further if difference is equal stride, confidence counter increments. Prefetch depth can be calculated as difference between confidence counter and reference repeat value. When current address become equal last address base + prefetch depth, last address base changed to current address and new data is prefetched. If stride changes, scale, confidence counter, late counter are reset. On late prefetch event late counter increments. If late counter overflows, stride is increased by itself, scale increments. Algorithm of proposed prefetcher is shown on Figure 5


stride: = new address - (last address + offset)
stride == [ str ]field
No
Yes
confidence counter ++
reset scale and counters
late counter overflow
yes
No
scale++ reset late counter
(current address=last address + prefetch depth) & (confidence cnt >= reference)
no
offset ++
yes
Prefetch data on current address + stride*scale
Figure 5 - Proposed algorithm

## 4 Evaluation

For value evaluation of proposed method was used cycle accurate simulator MARSS - RISCV with following configuration:

- core : out of order, single core
- L1 data cache
  - size : 32 KB
  - ways : 8
  - eviction policy : LRU
- L2 shared cache
  - size : 512 KB .
  - ways : 8
  - eviction policy : LRU

The evaluation uses TACLE benchmarks to explore prefetcher efficiency on regular access patterns and Linux launch for complex pattern exploration. To make sure that results given by MARSS - RISCV simulator are adequate, benchmarks were simulated on MARSS-RISCV without prefetcher and results of simulation were compared to RTL simulation of RISC-V core with the same configuration. The difference in measurements was not more than 0,9%.

Models of AR2P, stride prefetcher, adjacent line prefetcher and Domino prefetcher were simulated. Single core results are presented on the Figure 6.

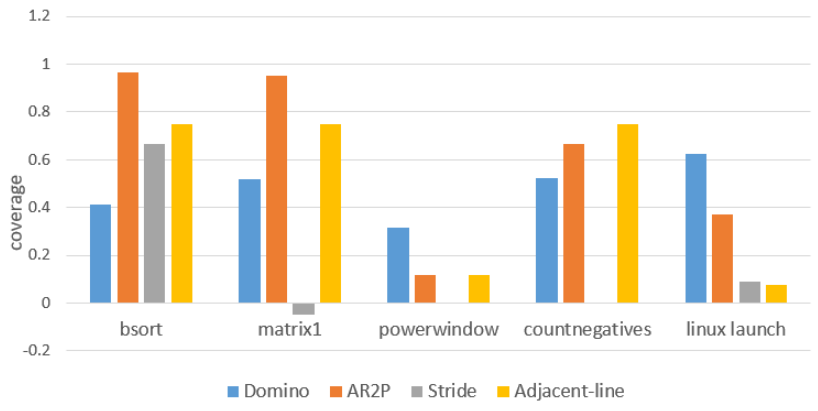


Figure 6 - Single core coverage

Coverage is a metric that shows what portion of cache misses were prevented by prefetcher.

$$c = 1 - \frac{p}{m}, \tag{1}$$

where c - coverage, p - misses with prefetcher, m - misses without prefetcher.

On diagram Tacle benchmarks bsort and matrix1 have mainly regular access patterns, Tacle powerwindow and countnegatives have complex regular and irregular patterns. Linux launch has more irregular than regular memory access patterns.

AR2P shows 95% coverage on the average on Tacle benchmark set with mainly regular patterns, that is 30% more effective than stride prefetcher and 20% more effective than adjacent line prefetcher. On complex patterns as Linux launch AR2P shows 37% coverage, that is 25% less effective than Domino because of conflict evictions from reference prefetch table and non clear stride.

At the moment vector of attention is directed on:

– The ability to dynamically adapt to irregular and complex patterns. The main idea is to detect, if clear stride does not exist and conflict evictions occur frequently, prefetcher assumes, that memory access pattern is irregular and uses reference prefetch table as miss history table.
– The dependence of efficiency of prefetcher on associativity of reference prefetch table/miss history table.

## 5 Conclusion

Researches[5] show that most effective prefetching is done by detecting and processing regular access patterns. In this paper, we presented an analysis of the most common data prefetching algorithms and presented an adaptive algorithm for regular patterns that tracks and prevents late prefetching. Also, due to the adaptive prefetch depth, the memory bus bandwidth consumption is reduced. As expected, AR2P shows inferior results for irregular access patterns and current works are directed on the ability to dynamically adapt to irregular and complex patterns

## References

1. Jamison D. Collins, Dean M. Tullsen: Runtime Identification of Cache Conflict Misses: The Adaptive Miss Buffer. ACM Transactions on Computer Systems, **4**(5), 414–439 (2001)
2. Jain A., Lin C.: Linearizing Irregular Memory Accesses for Improved Correlated Prefetching. In:Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture-MICRO-46, pp. 247–259. ACM , Davis, CA, USA (2013)
3. Al-Sukhni H., Holt J., Connors D. : Improved Stride Prefetching using Extrinsic Stream Characteristics. In:Performance Analysis of Systems and Software, 2006 IEEE International Symposium, pp. 166–176. IEEE Xplore, April (2006)
4. Wu, H., Nathella, K., Sunwoo, D., Jain, A., Lin, C: Efficient Metadata Management for Irregular Data Prefetching. In:ISCA '19: ACM International Symposium on Computer Architecture, pp. 1–13. June 22–26, 2019, Phoenix, Arizona, USA
5. Kondguli S., Huang M.: T2: A Highly Accurate and Energy Efficient Stride Prefetcher. In: 2017 IEEE International Conference on Computer Design (ICCD)., pp. 373–376.
6. Bakhshalipour M.,Lotfi-Kamran P., Sarbazi-Azad. H: Domino Temporal Data Prefetcher. In: 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)., pp. 131–142