Automated Software Prefetching for Irregular Access Patterns

Konstantinos Sotiropoulos

March 8, 2024

hej!

My name is Konstantinos!

I am a PhD candidate in Computer Architecture

For the past 1.5 years,
I ’ve been at Chalmers under the guidance of Per Stenström

Prior to that,
I was working as an embedded software engineer

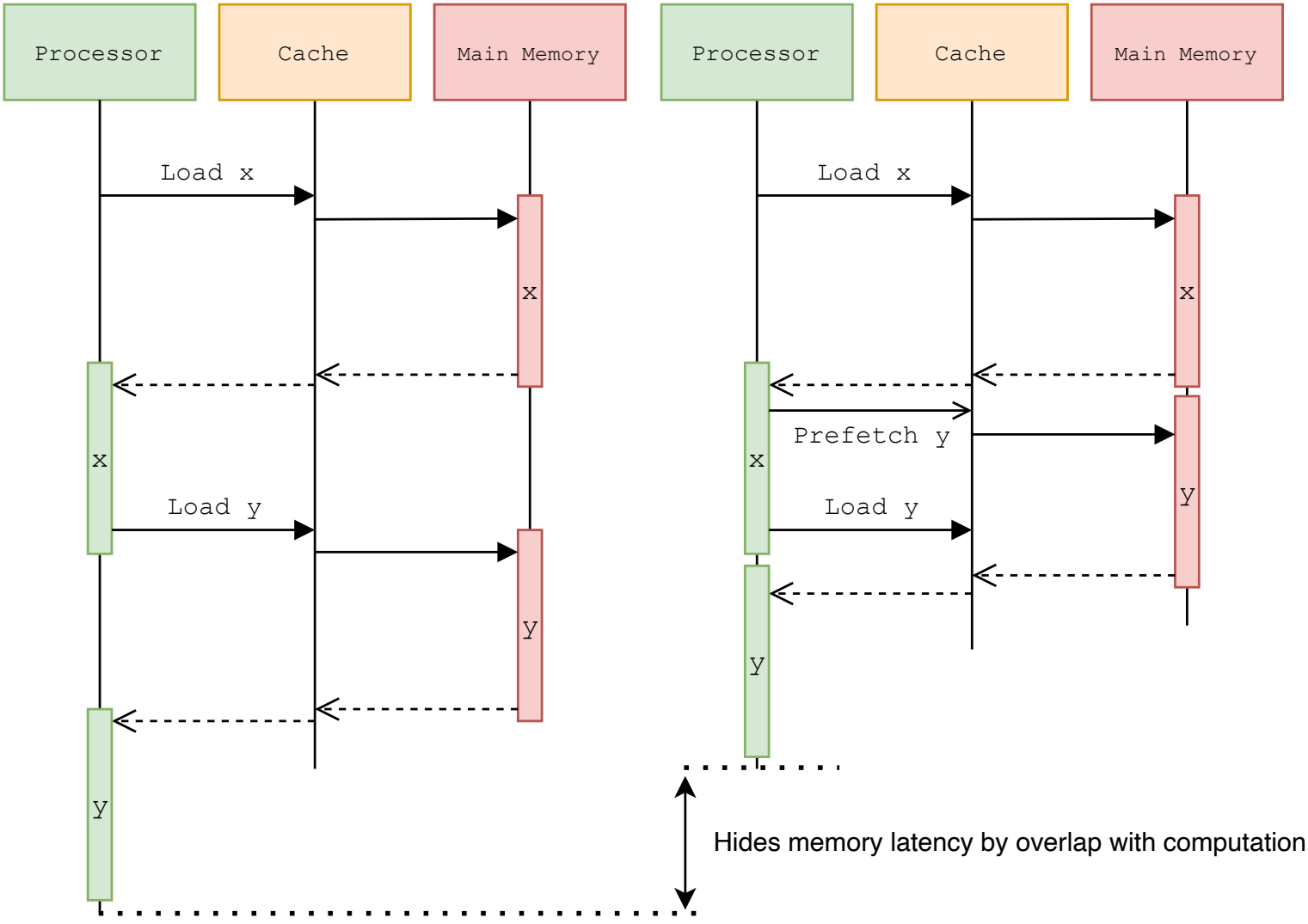
my focus: prefetching

Fetching data before it's actually requested

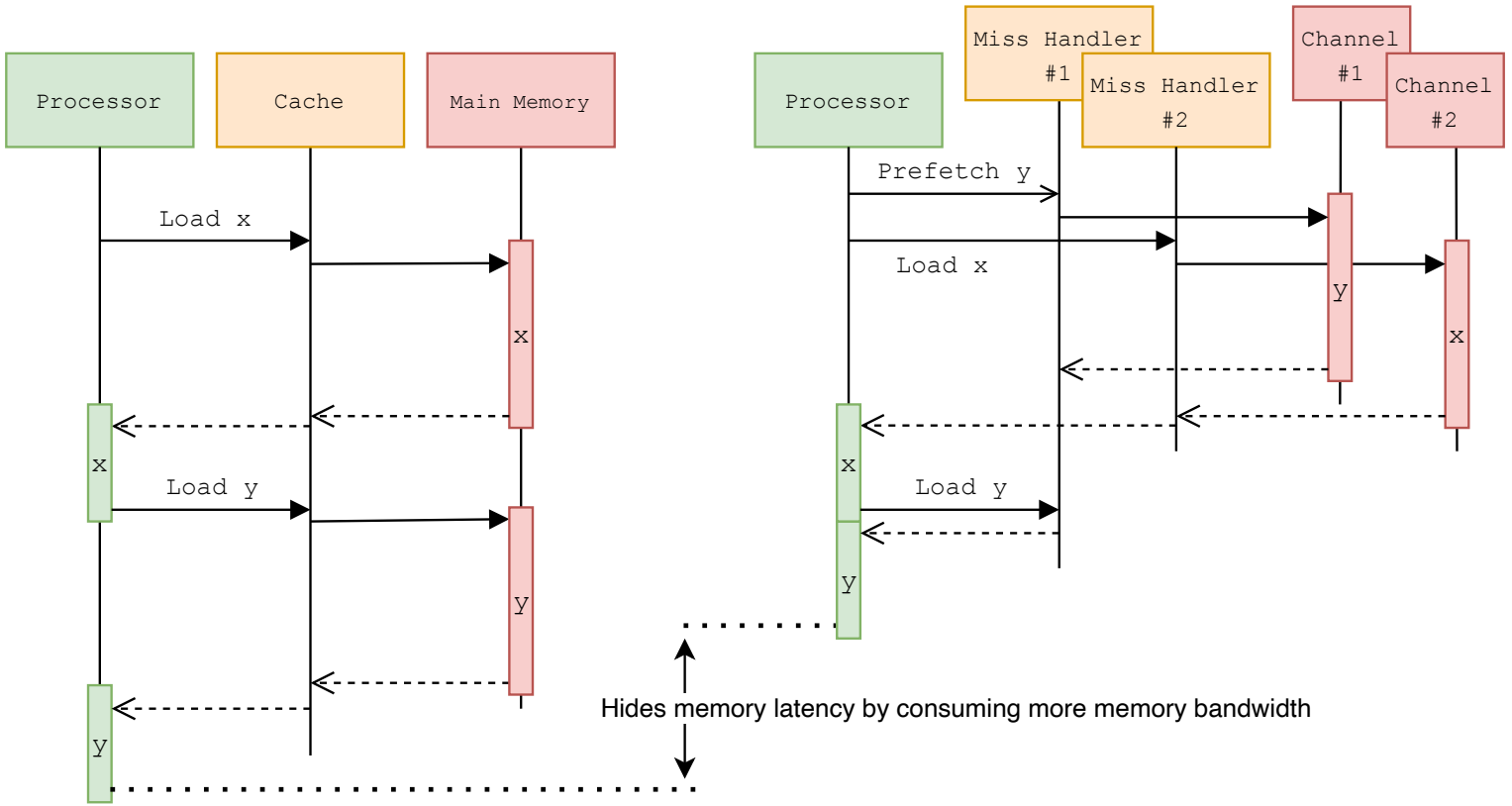
Prefetching is a memory latency hiding technique

A prefetch will not block the issuing party

prefetching: overlap with computation



prefetching: latency & bandwidth trade-off



prefetching: who can do it?

- hw prefetchers: non-programmable machinery inside the processor, observing and predicting future memory accesses
- sw prefetching: the program itself, using special prefetch instructions

  Prefetches can be inserted by:

  - manually, by the expert programmer for a performance library
  - automatically, by an optimizing compiler

my problem domain: irregular applications

Irregular Applications, not following predictable patterns in their memory accesses

Bottlenecked by main memory latency

A key source of irregularity, and my focus,
is indirection: `C[B[A[i]]]`

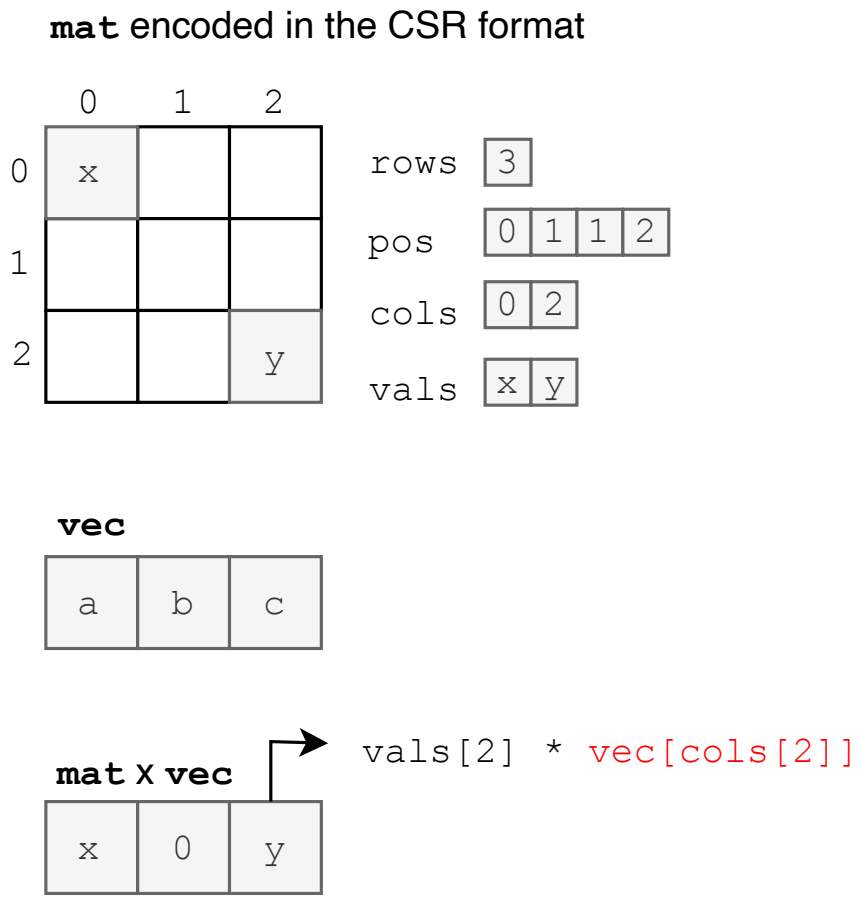
my application domain: sparse linear algebra

Efficient computation with tensors
predominantly filled with zeros

Store only non-zero values and their coordinates

A sparse tensor encoding describes the way these coordinates are stored

sparse tensor encodings: source of irregularity



recap: what have we seen so far?

- the focus: prefetching
- the problem domain: irregular applications
- the application domain: sparse linear algebra

prefetching for sparse linear algebra

- hw prefetching: not ideal for irregular
- sw prefetching: needs to be automated, by an optimizing compiler.

  For a given operation the number of combinations is exponential to the number of encodings

automatic sw prefetching: state-of-the-art

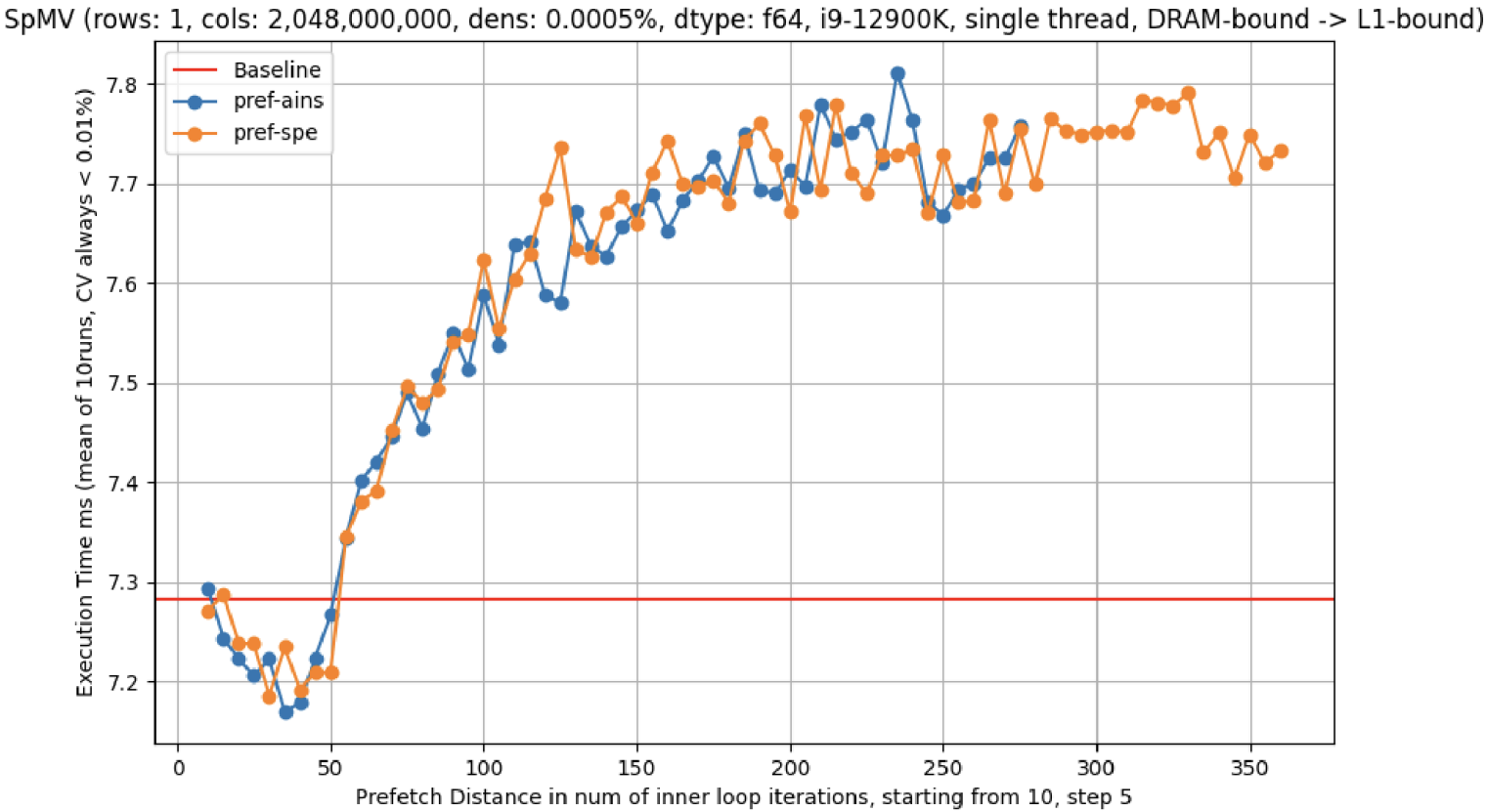
Evaluating Current Approaches:

1. Mehta et al., 2022 - *"Software Pre-execution for Irregular Memory Accesses in the HBM Era"*
2. Ainsworth & Jones, 2018 - *"Software Prefetching for Indirect Memory Accesses"*

Findings:

- Both approaches yield similar results, with marginal speedups
- Notably, these methods do not saturate memory bandwidth, indicating potential for more aggressive prefetching

automatic sw prefetching: state-of-the-art



automatic sw prefetching: challenge

The compiler struggles to infer the high-level semantics,
necessary for more aggressive prefetching,
from an assembly-like intermediate representation (IR)

There is a big gap between:

- what the application does: sparse linear algebra
- what the compiler sees: nests of non-affine loops

MLIR: bridging the semantic gap

Instead of optimizing by transforming an IR, why don't we generate optimized code

Start from a Domain Specific Language (DSL) like tensorflow and produce code that can run on the metal

Enter Multi-Level Intermediate Representation (MLIR)

MLIR: Sparse Matrix-Vector multiplication

Demo: from python to machine code, see the code generated at each abstraction layer

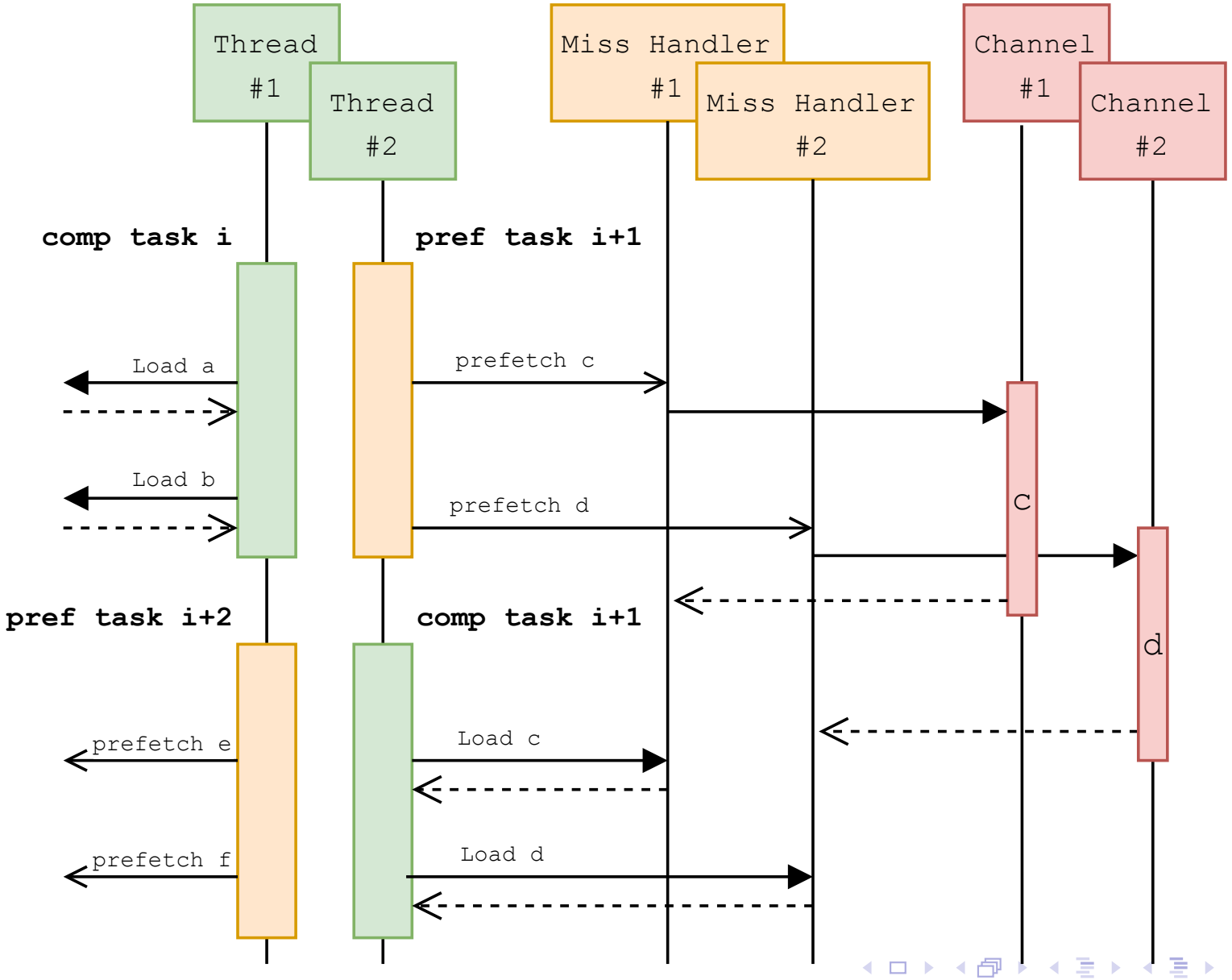
my approach: runahead prefetching

Multiple threads alternating between:

- prefetching data for a specific computation chunk
- executing that computation chunk

Described, but not automated in:
Kamruzzaman et al., 2011 - *"Inter-core prefetching for multi-core processors using migrating helper threads"*

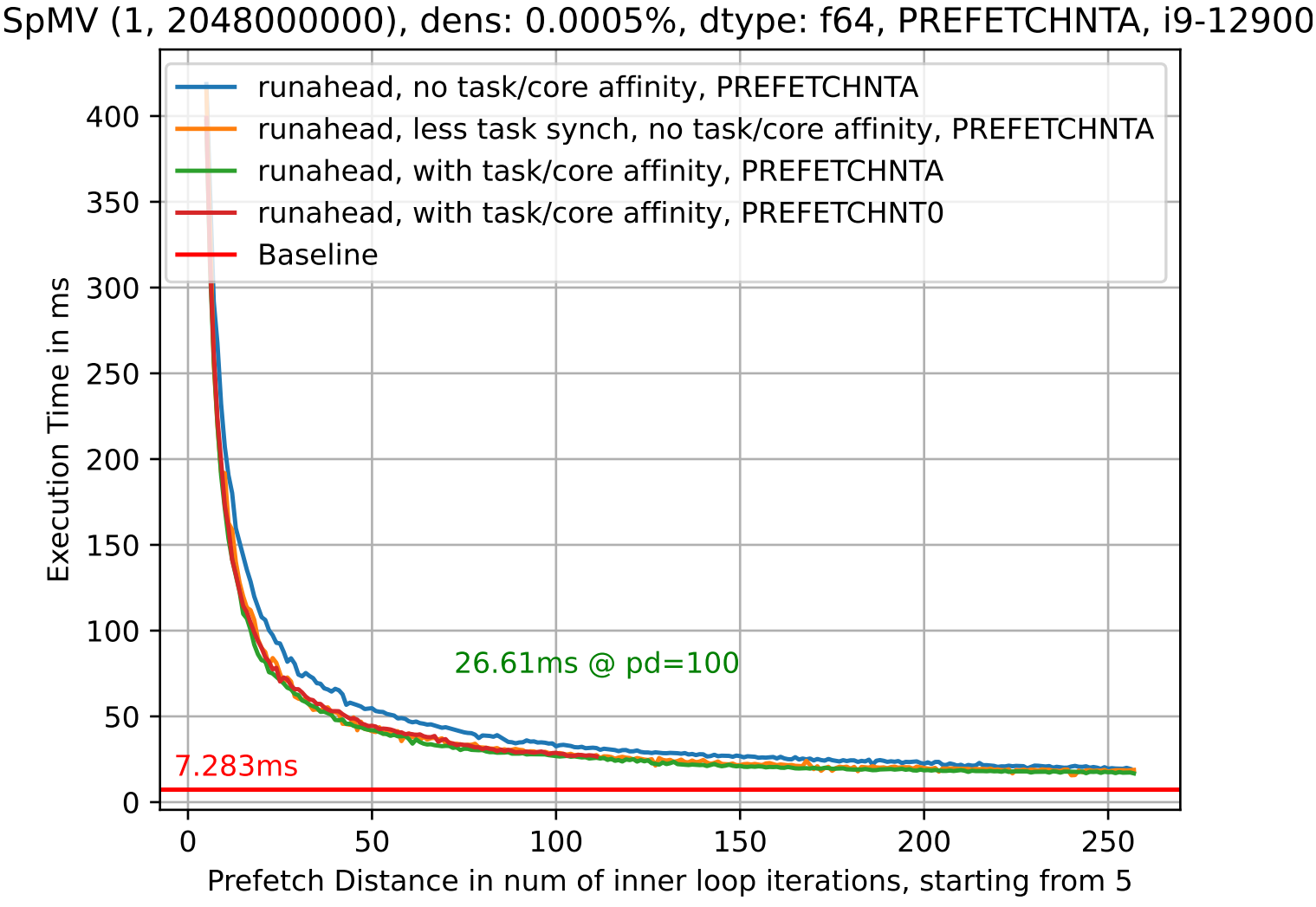
runahead prefetching: compute & prefetching tasks



my approach: automation using MLIR

Utilizing the OpenMP dialect within MLIR, we can generate the task-based parallel code described above

current status: prototyping



recap: what have we seen so far?

- focus: prefetching
- problem domain: irregular applications
- application domain: sparse linear algebra
- state-of-the-art: not automated and/or not aggressive enough
- hypothesis: not utilizing semantic info
- thesis: runahead prefetching, using MLIR, from DSL to OpenMP
- status: prototyping

thank you!

The github repo linked below houses our ongoing work