Friedrich-Alexander-Universität
Technische Fakultät

Computer Architecture Practical Exercise

6 Cache Blocking 2D

Kenan Gündogan[1] **Philipp Gündisch**[1]

[1]Friedrich-Alexander Universität Erlangen-Nürnberg, Chair of Computer Science 3 (Computer Architecture)

December 13, 2023

L1 Cache Blocking

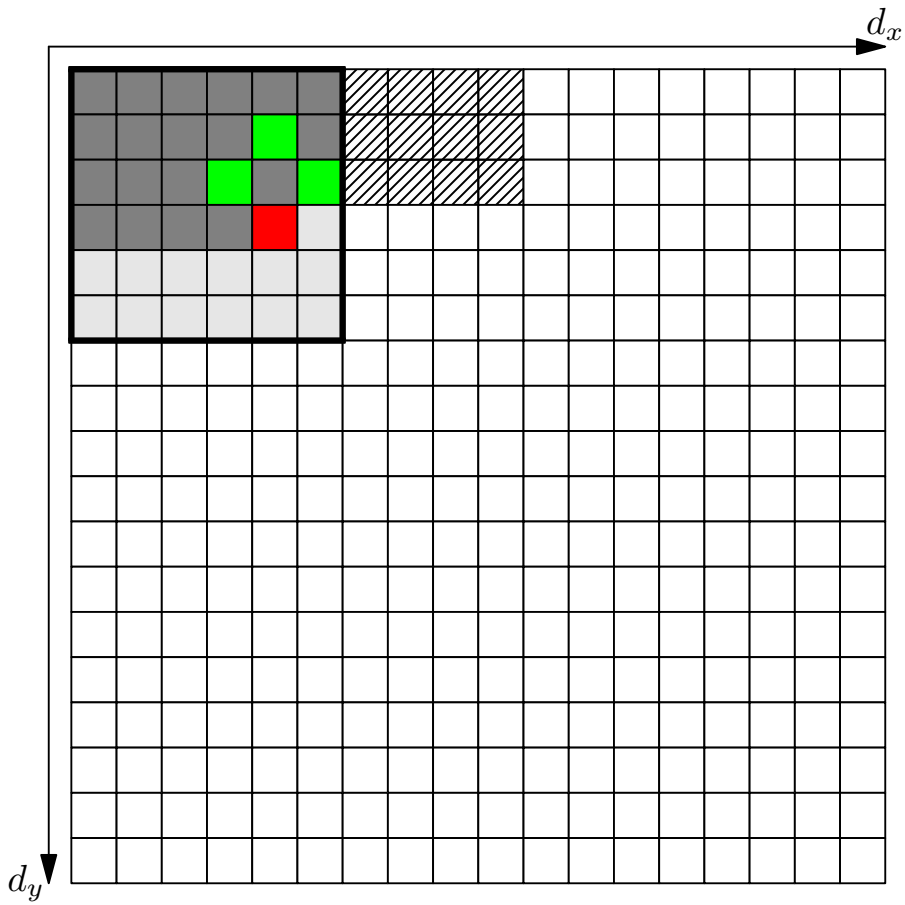
Motivation

- L1 Cache enables the fastest memory access
- Usually very small cache sizes
- *Cache Prefetching* cannot be ignored for these very small cache sizes

Advanced L1 Blocking

Cache Prefetching

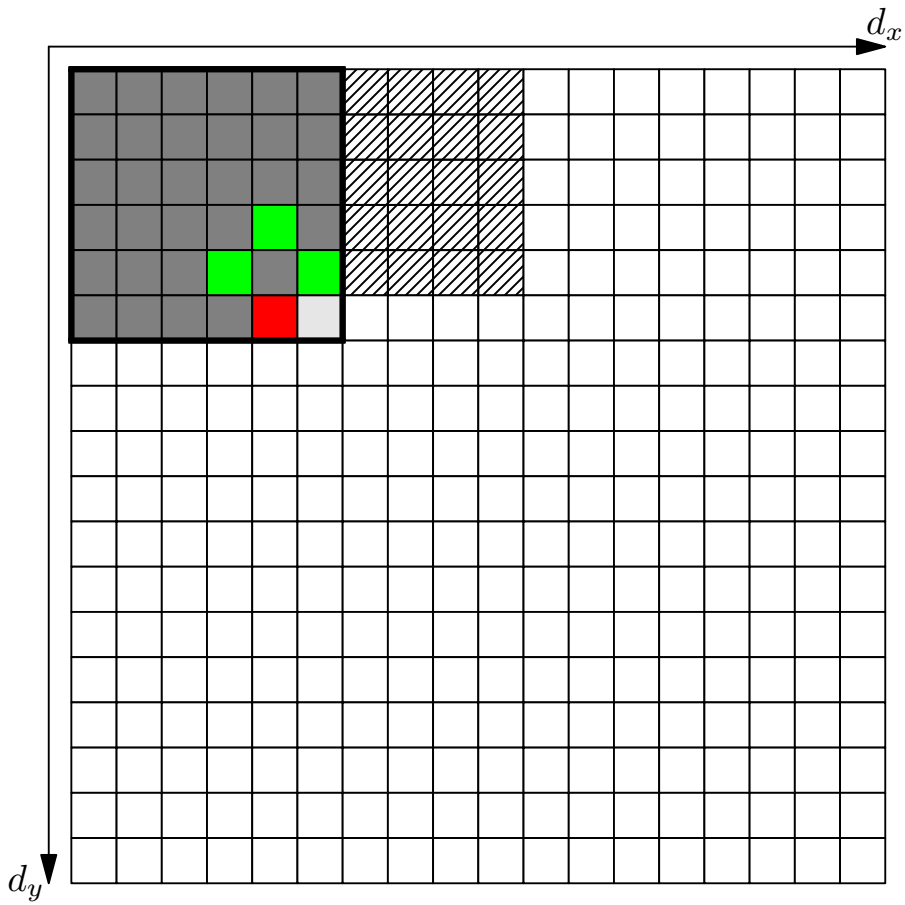
- Hardware prefetcher tries to guess upcoming memory addresses to *prefetch* data from slow memory
- Works similar to branch prediction but for memory access patterns
- Issue: Additional data is filling up the cache but is not accessed before displacement



Advanced L1 Blocking

Solution

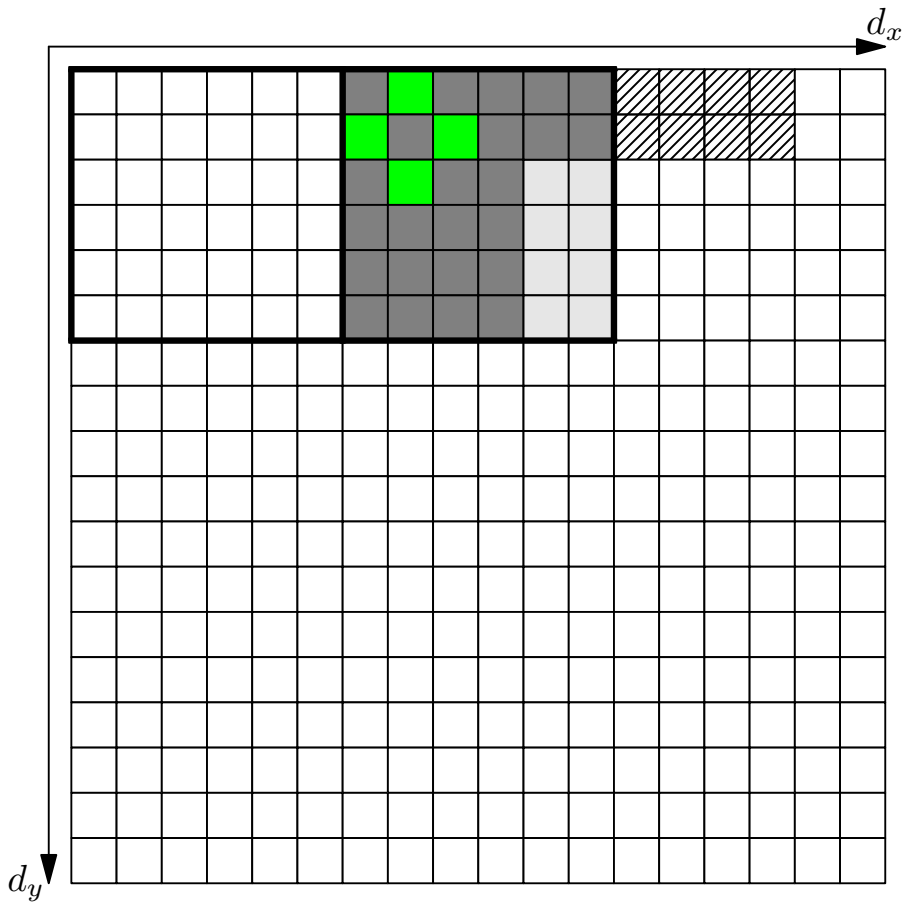
- Stop block iteration before prefetched data gets displaced
- Block in d_x and d_y direction
- The block of size $b_x \cdot b_y$ is called a ***tile***
- For L1 blocking, keep the **whole *tile*** and the prefetched data within the cache



Advanced L1 Blocking

Grid Iteration

- When the tile is processed proceed with the tile to the east
- Once a row of tiles is completed proceed to the south



Task 6.1: Cache Blocking

Blocking Parameters

Due to the unknown prefetching size it is difficult to estimate the blocking factor in b_x and b_y direction. Additionally, a modification of b_x does not have the same impact as a modification of b_y but both influence each other. Therefore, we will determine these blocking factors empirically.

- Determine a reasonable theoretical maximum and minimum for b_x and b_y
- Implement L1 cache blocking by extending the solution of the last exercise
- Make b_x and b_y compile time constants
- Choose a significantly large GridSize of about 512 MiB
- Determine (roughly) the best b_x, b_y combination while benchmarking for a **reasonable time**
- **OPTIONAL** Visualize your results as a heatmap with b_x and b_y as the axis and MUp/s as the coloration (similar to the jacobi ppm file)

Task 6.2: Cache Blocking

Benchmark

- Benchmark your implementation with the best b_x, b_y combination from 6.1
- Benchmark from 1 MiB to 16 GiB
- Create the performance plots as usual with MUp/s and ArraySize as axis
- Compare the new version against the blocked version from exercise 5

Task Overview

- E 6.1: Blocking Parameters
  - Determine the optimal blocking parameter combination for b_x and b_y
- E 6.2: L1 Cache Blocking
  - Benchmark the updated implementation from 1MiB - 16GiB

Appendix: Checklist

Performance Optimization (1/2)

During the timeline of this class new bullet points will be added. Recently added entries are bold.

- Compiling
  - Choice of the compiler (`icc`)
  - Compiler flag to optimize aggressively (e.g. `-O3`)
  - Compiler flag to adapt for specific hardware (e.g. `-xHost`)
- Programming Techniques (if applicable)
  - Use `#define` and `const` instead of variables
  - Data type aware programming
  - Use aligned memory (e.g. with `_mm_malloc()` or `posix_memalign()`)
  - Consecutive address iteration
  - Variable declarations outside of loops
  - Reduce function calls
  - Use intrinsics (to utilize SIMD)
  - Cache aware programming (Spatial Blocking)
  - **Prefetcher aware programming (L1 Cache Blocking)**

Appendix: Checklist

Performance Optimization (2/2)

During the timeline of this class new bullet points will be added. Recently added entries are bold.

- Measurement
  - Reasonable benchmark time
  - Reasonable benchmark workload
  - Reduce interference factors to a minimum
- Optimization Process
  - Check assembler code while optimizing
  - Check performance gains while optimizing
  - Use profiling tools
  - Ensure correctness of code
  - Optimize iteratively