Friedrich-Alexander-Universität
Technische Fakultät

Computer Architecture Practical Exercise

5 Cache Blocking

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

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

December 5, 2023

Caches & 2D Jacobi

Motivation

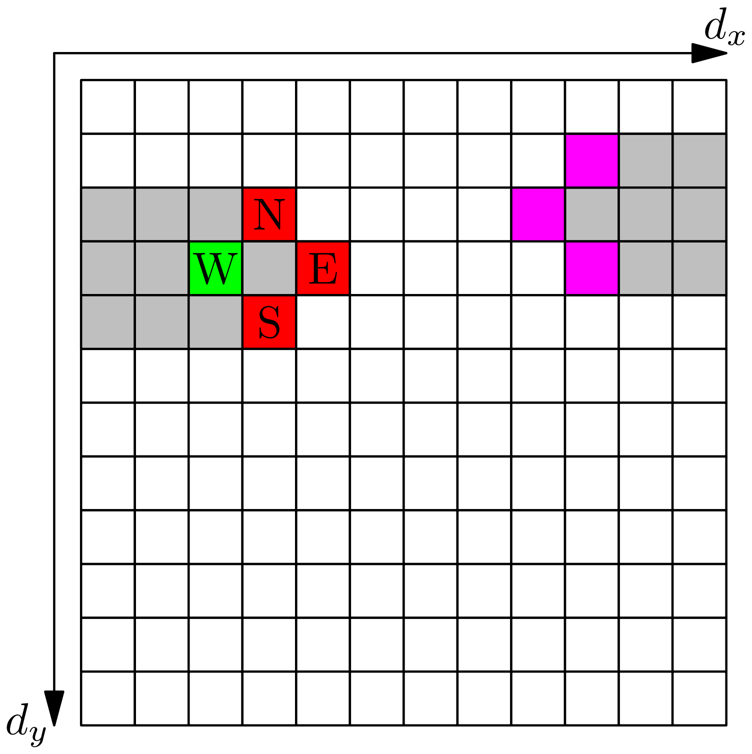
- Physical simulations like Jacobi are time and space discrete
- The higher the grid resolution the more accurate is the simulation
- But: A High resolution leads to an increased memory consumption exceeding the CPU cache capacity significantly
- Solution:
  1. Distributed computing across cluster nodes (not part of this module, see PACL)
  2. Grid size per node can still be in the GBs, vastly exceeding the cache sizes
- Are caches of any help at all in this scenario?

Caches & 2D Jacobi

Are relatively small caches helpful?

Yes! Even if the grid is too large to fit into the L3 cache, caches can still be helpful.

1. The cell to the west (**W**) can be loaded from the cache since it has been used two iterations ago
2. Ideally, also the cells to the north (**N**) and east (**E**) can be loaded from cache since they have been recently used
3. Whether or not the cells are still available in the cache depends on the grid width d_x, the cache size and the cache replacement strategy
4. Only the cell to the south (**S**) needs to be loaded from RAM

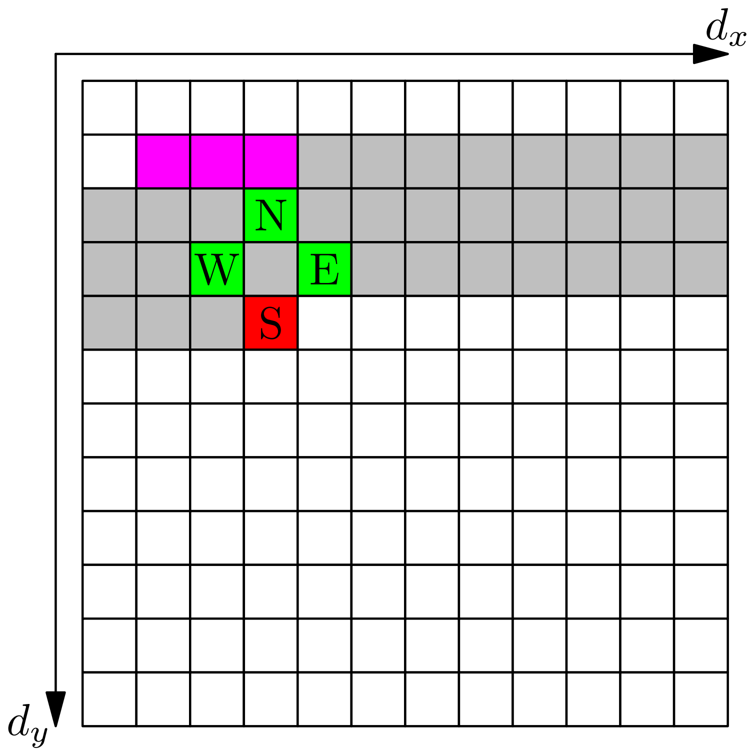


Caches & 2D Jacobi

Layer Condition

To ensure the cells can be read from the cache we formulate a so called **layer condition**.

1. The north (**N**) and east (**E**) neighbor cells can be loaded from cache if the cache is big enough to keep three full grid lines (+1)
2. If c_i is the cache capacity of the L_i-Cache in bytes then the two cells can be read from the cache if $c_i > (3 \cdot d_x + 1) * 8B + OtherMem$
3. What about the second grid?
4. Hardware integrated prefetchers also have negative effects on the layer condition (next week)



Example showing a satisfied layer condition.

Task 5.1: Cache Sizes

- Note down the cache sizes (in kB) for each level
- What maximum grid width d_x (in cells) can utilize cache blocking according to the layer condition
- What maximum jacobi grid size (in cells) can utilize cache effects of each level
- Fill the following table

	Cache Size	d_x	Max Jacobi Grid Size
L1	32 kiB		
L2	256 kiB		
L3	25 MiB		

Task 5.2: Cache Effects

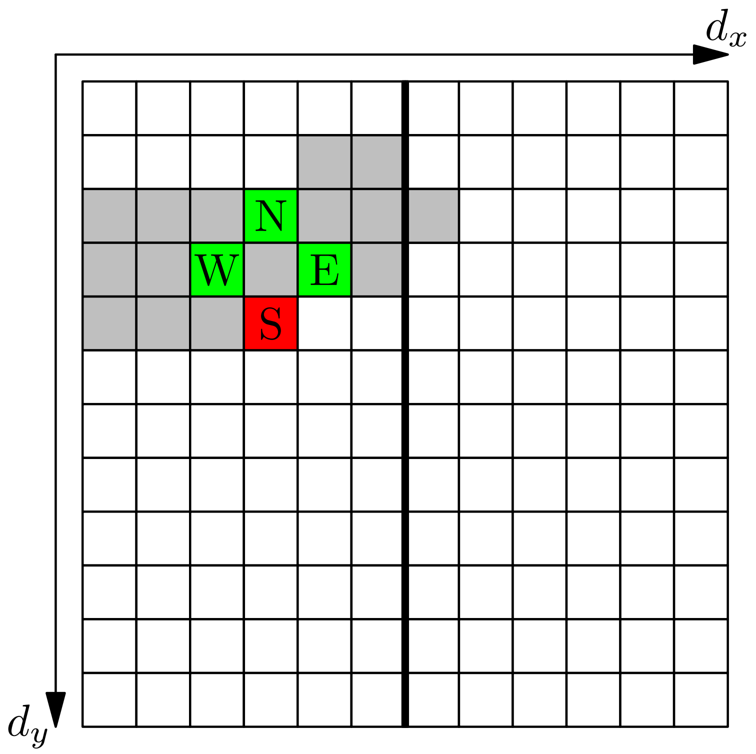
Show that it is very crucial for the performance what data is loaded from which cache.

- Benchmark the AVX vectorized jacobi implementation
- This time, benchmark from 1MiB to 16GiB
- Create the performance plots as usual with MUp/s and ArraySize as axis
- Confirm the plots with the theoretical grid sizes of task 5.1

Cache Blocking

To improve the AVX vectorized jacobi implementation even further we use a cache blocking technique known as **Spatial Blocking**.

1. Loop iterations are artificially limited (blocked) such that cache data can be reused
2. The inner most loop will be split into two loops
3. The outer loop runs over i_b blocks, each of width b_x (except the last block)
4. The inner loop is limited to run from $i_b \cdot b_x$ to $(i_b + 1) \cdot b_x$
5. Hint: That makes three nested loops in total (y, i_b, b_x)
6. Hint: Ensure checking the results for correctness (e.g. with `diff`)



Task 5.3: Jacobi Spatial Blocking

- Determine a blocking factor b_x which satisfies the L2 layer condition
- Make the blocking factor a `#define` constant
- Update the AVX jacobi implementation with this blocking factor
- Ensure checking the results for correctness (e.g. with `diff`)
- Extend the plot from task 5.2 with the results of this version

Task Overview

- E 5.1: Cache Blocking
  - Use the layer condition to fill the table
- E 5.2: Cache Levels and Performance `gprof`
  - Benchmark the existing implementation from 1MiB - 16GiB
- E 5.3: Cache Blocking
  - Extend the jacobi AVX implementation with spatial blocking
  - Plot the results to show the difference to the not blocked variant
  - Interpret the results

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)**

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