Friedrich-Alexander-Universität
Technische Fakultät

Computer Architecture Practical Exercise

2 Two Dimensional Jacobi

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

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

November 11, 2024

2D Jacobi

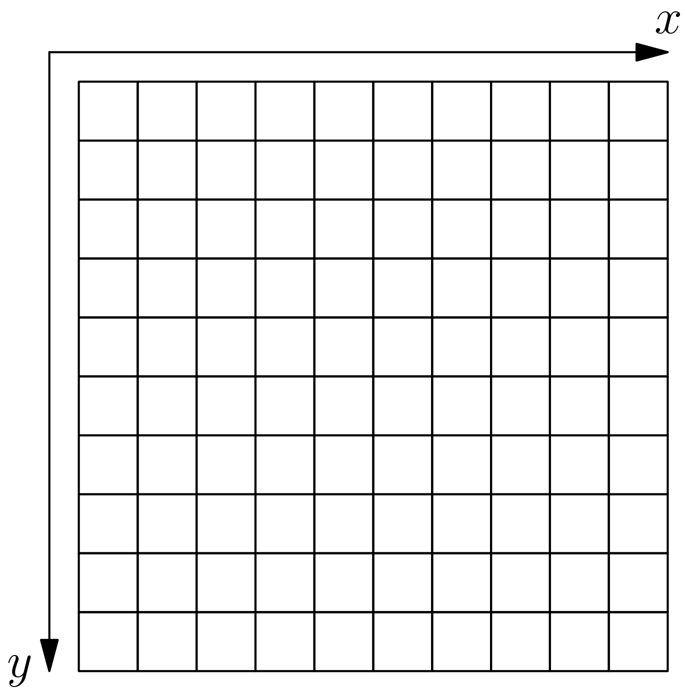
- Simple example for scientific computing
- Used as a numerical solver to process heat propagation
- Naive code is simple
- Code has a huge optimization potential
- Perfect example to illustrate modern CPU features like:
  - SIMD Vectorization
  - Cache Blocking
  - Multi-Core and Many-Core
  - NUMA

Fundamentals

- Jacobi-scheme is a numerical iteration based solver for linear equations of the type $Ax = b$, (e.g. needed to calculate the Δ Laplace operator).
- Laplace operator describes the physics of fields expressed as differential equations:
  - Electrostatic fields
  - Newtons dynamic in fluids and gases
  - Heat propagation
  - etc.

2D Jacobi Heat Propagation

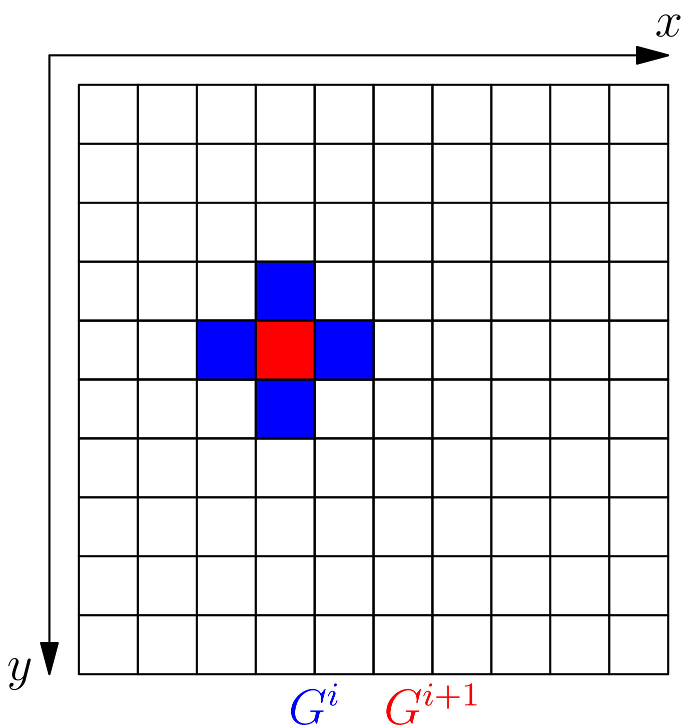
- 2-dimensional plane needs to be discretized.
- Each cell holds the current temperature as a `float` or `double` value.
- Grid is represented as an array of floating point values.



2D Jacobi Heat Propagation

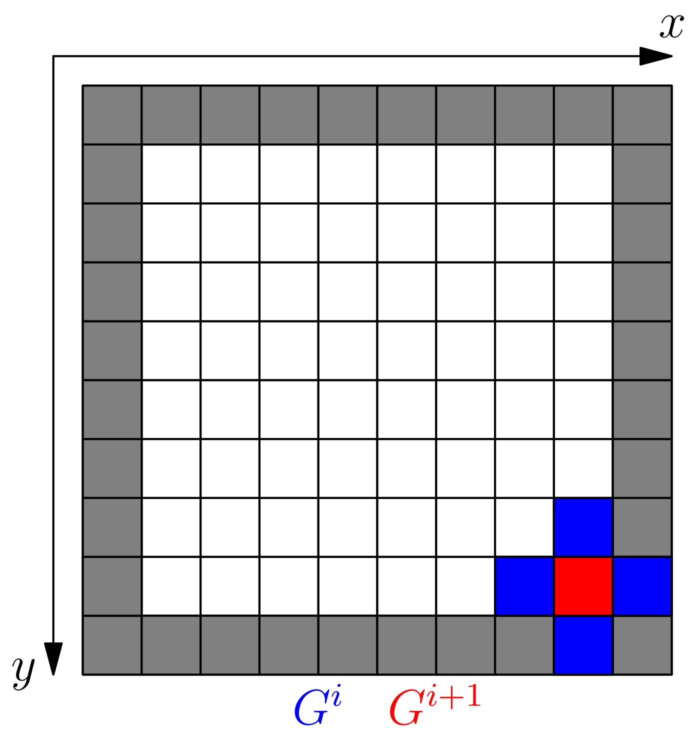
- Computation operates in discrete time steps and discrete spatial regions.
- Simulation of one time step i to $i+1$ for coordinates x and y:

$$G_{x,y}^{i+1} = \frac{G_{x,y+1}^{i} + G_{x,y-1}^{i} + G_{x+1,y}^{i} + G_{x-1,y}^{i}}{4}$$



2D Jacobi Update

- For each simulated time step the whole grid needs to be iterated.
- In our example the grid margins remain constant.
- After each iteration the two grids (source and target) can be swapped.

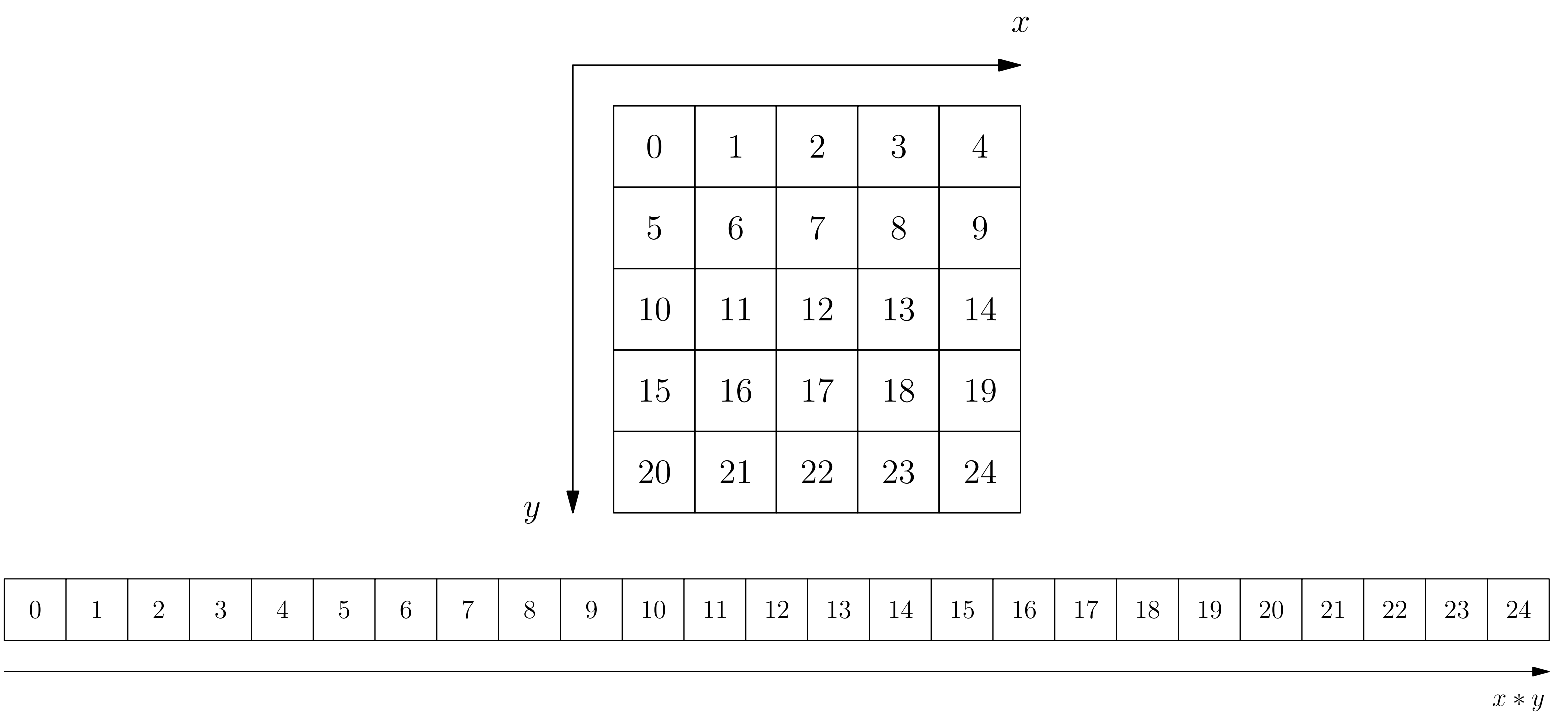


Array Linearization

Use a single memory allocation instead of many:

```
grid1D = malloc(dx * dy * sizeof(double));
```

- A two dimensional array can be accessed with `grid2D[x][y]`.
- A linearized array can be accessed with `grid1D[dx * y + x]`.
- Use the linearized access method!



2D Jacobi

Pseudocode

```
// Iterate over simulation time steps
for(t = 0 ; t < T ; t++) {
        // Update cells
        for(y = 1 ; y < Y-1 ; y++) {
                for(x = 1 ; x < X-1 ; x++) {
                        grid_trgt[y*X+x] = grid_src[(y-1)*X+x] + ...
                }
        }
        // Switch the pointers for next iteration
        swap(&grid_trgt, &grid_src);
}
```

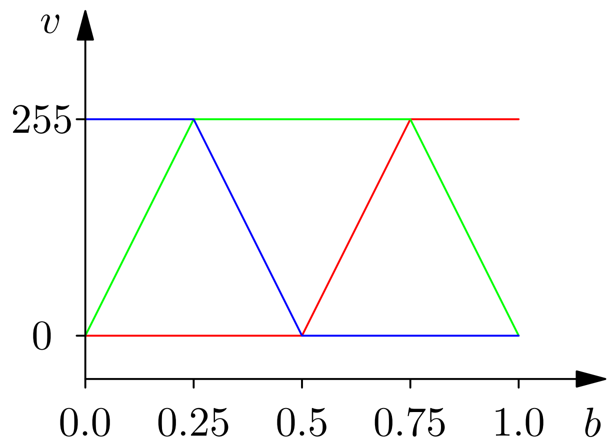
Task 2.1: 2D Jacobi

Implementation

- Download new sources and embed it in skeleton of exercise 0.
- Update makefile to build a new target.
- Update the main program to allocate two linearized arrays.
- Use `_mm_malloc()` with alignment on 64 B (*from* `<immintrin.h>`).
- The new program should receive the edge length (= grid width = grid height) as a parameter
- Initialize the left and top margin with *1.0*
- Set the remaining cells to *0.0*.
- Implement the *todos* in the new source code.

Visualization

- The function `draw_grid()` converts the gird to a **Portable Pixmap format** (.ppm) and saves it to a file.
- The header row of the file contains the meta information:
  `P<magic number> <width> <height> <max value>`
  e.g.: P3 128 128 256
- Use P3 or P6 as a format.
- Use the following coloration scheme to map the floating point range $[0.0; 1.0]$ to RGB colors.
- Use this function for debugging purposes during development



Task 2.2: 2D Jacobi

Result Validation

Expected Jacobi output produced by the `draw_grid()` function.

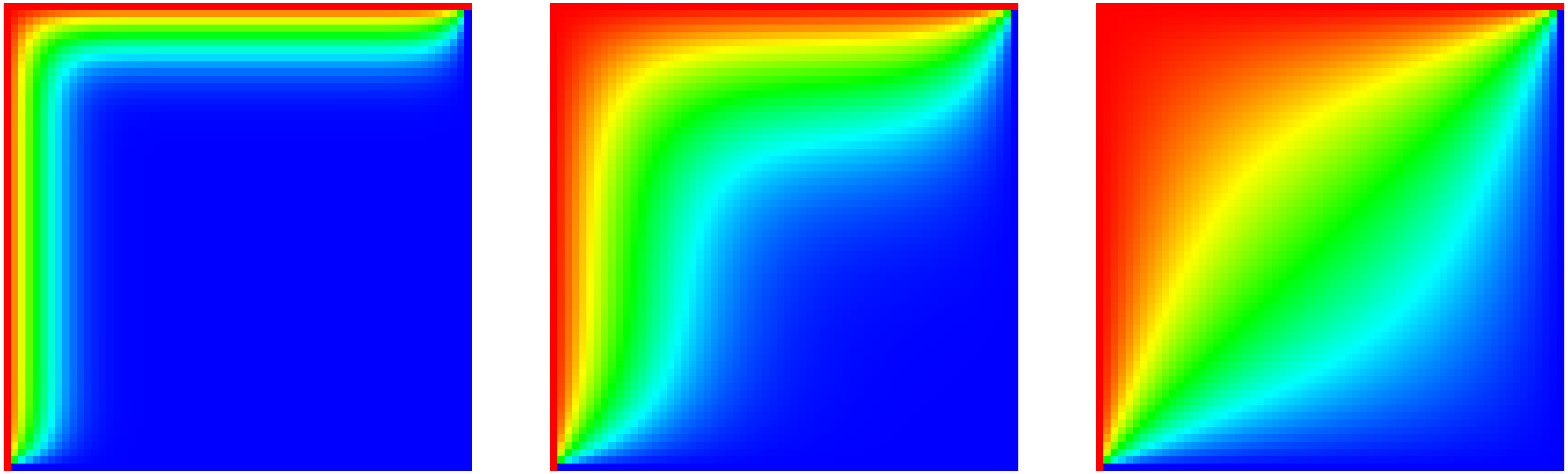


Figure: Three snapshots of a 64 by 64 Jacobi grid showing different states of the propagating heat from top-left to bottom-right.

Task 2.3: 2D Jacobi

Measurement

- For measurement the number of discrete time steps should be doubled until 1 second is reached.
- Log the same information as in the last exercise and
  - replace *AdditionsPerSecond* with *MUp/s* (for MegaUpdatesPerSecond)
  - add a new column *EdgeLength* showing the elements on the edge of the quadratic jacobi array
- An update is defined as a single grid cell update.
- $10^6 Up/s = 1 MUp/s$
- Measure from 1kiB to 128MiB (the total allocated memory).
- Use a quadratic array with edge length: $X = Y = \sqrt{\frac{n \cdot 1024B}{2 * 8B}}$

Task Overview

- E 2.1: Jacobi
  - Include new source code
  - Update main.c or create new one
  - Update Makefile
  - Implement jacobi.c
- E 2.2: Visualization
  - Implement draw.c
  - Call draw function from your main.c (for debugging and end result)
- E 2.3: Time Measurement
  - Update the `printf()` format
  - Adapt scripts for measurement
  - Run benchmark on cluster

Appendix: Checklist

Performance Optimization

During the timeline of this class new bullet points will be added.

- Compiling
  - Choice of the compiler (`icx`)
  - 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
- Measurement
  - Reasonable benchmark time
  - Reasonable benchmark workload
  - Reduce interference factors to a minimum