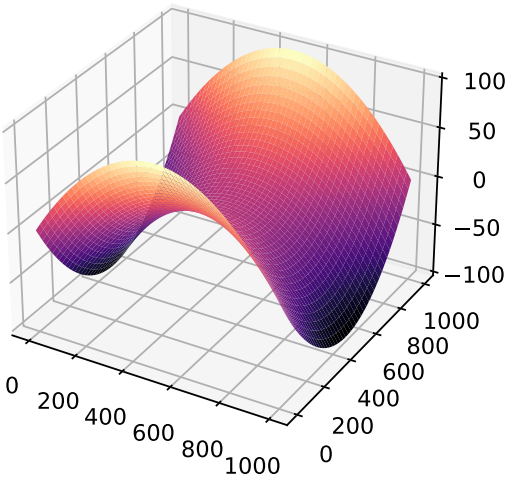
Jakob Hördt, Philipp Müller

Numerical Solution of Laplace’s Equation

Implementing a high performance solver and conducting performance analysis

Table of contents

The Laplace equation



$$f : \mathbb{R}^n \to \mathbb{R}$$

$$\Delta f = 0$$

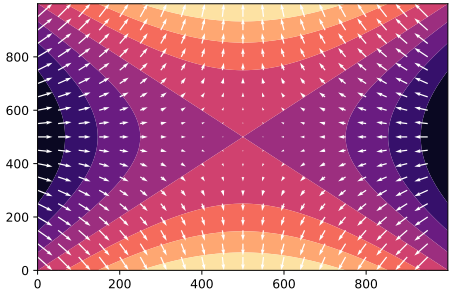
$$\Delta f = \nabla^2 f = \text{div grad } f$$

$$\nabla = \left(\frac{\partial}{\partial x_1}, \dots \frac{\partial}{\partial x_n} \right)$$

left: $f(x, y) \approx x^2 - y^2$

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 2 - 2 = 0 \text{X}$$

Laplace Operator



$\nabla f = \text{grad } f = \left(\frac{\partial f}{\partial x_1}, \ldots \frac{\partial f}{\partial x_n}\right) = \text{first order derivative of } f$

$\nabla^2 f = \frac{\partial^2 f}{\partial x_1^2} + \ldots + \frac{\partial^2 f}{\partial x_n^2} = \text{second order derivative of } f$

Intuition

$\Delta f = 0$

No max or min

In 2D: $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0 \rightarrow \frac{\partial^2 f}{\partial x^2} = -\frac{\partial^2 f}{\partial y^2}$

- Curvature in one dimension cancels curvature in other direction

Equilibrium equation

Influx equals efflux at every point

Applications

Gauß's law for **gravity**:Gauß's law for **gravity**:

$$\nabla g = -4\pi G\rho$$

$$g = -\nabla\phi$$

Poisson's equation for gravitational fields:

$$\Delta\phi = 4\pi G\rho$$

$$\rho = 0 \rightarrow \Delta\phi = 0$$

Laplace's equation for

Heat equation:

$$\frac{\partial}{\partial t}u(x,t) = -a\Delta_x u(x,t)$$

$\frac{\partial}{\partial t}u(x,t) = 0 \rightarrow$ Laplace equation

Electrostatics:

$$\mathbf{E} = -\nabla V_{\mathbf{E}}$$

$$\nabla\mathbf{E} = \nabla(-\nabla V_{\mathbf{E}})$$

$$= -\Delta V_{\mathbf{E}} = \rho\varepsilon_0$$

$V_{\mathbf{E}}$ satisfies Laplace equation where $\rho = 0$

Boundary Conditions

Laplace equation

- infinitely many solutions

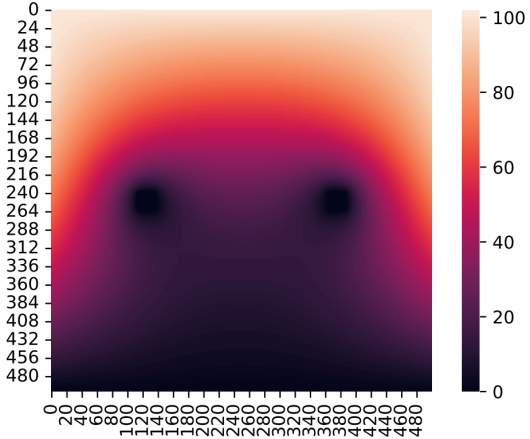
Dirichlet boundary conditions

- fixed continuous values at some boundary

Dirichlet boundary conditions

- unique solution

Solutions



Analytic solution infeasible

Goal

We created numeric solver

Scales to large problem size on a cluster

Outline

Finite element method

Discretize domain into grid cells with spacing h:

$$u_{i,j+1} = f(x, y+h)$$

$$u_{i-1,j} = f(x-h, y) \qquad u_{i,j} = f(x,y) \qquad u_{i+1,j} = f(x+h, y)$$

$$u_{i,j-1} = f(x, y-h)$$

Finite difference method

Approximate Laplace operator at grid cells.
By Taylor expansion in x-dimension:

$$u_{i+1,j} \approx u_{i,j} + h\frac{\partial f}{\partial x} + \frac{1}{2}h^2\frac{\partial^2 f}{\partial x^2}$$

$$u_{i-1,j} \approx u_{i,j} - h\frac{\partial f}{\partial x} + \frac{1}{2}h^2\frac{\partial^2 f}{\partial x^2}$$

Then:

$$u_{i+1,j} + u_{i-1,j} \approx 2u_{i,j} + h^2\frac{\partial^2 f}{\partial x^2}$$

Finite difference method

Similarly for y-dimension:

$$u_{i,j+1} + u_{i,j-1} \approx 2u_{i,j} + h^2 \frac{\partial^2 f}{\partial y^2}$$

Adding both together leads to:

$$\Delta f \approx (u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j} - 4u_{i,j})/h^2$$

Since $\Delta f = 0$:

$$u_{i,j} \approx (u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j})/4$$

Each grid cell is roughly the average of its neighbors!

Relaxation

$n \times n$ grid

- n^2 linear equations
- Solving directly infeasible

Relaxation method

- Iteratively apply approximation to grid cells until convergence
  - Jacobi: out-of-place, pleasingly parallel
  - Gauß-Seidel: in-place, twice as fast, sequential, allows overrelaxation
- $\mathcal{O}(n^2)$ iterations, $\mathcal{O}(n^4)$ runtime.

Overrelaxation

Idea: overcorrecting with $w \in [1, 2)$

$$\text{next} = u + w * (\text{average} - u)$$

$\mathcal{O}(n)$ iterations, $\mathcal{O}(n^3)$ runtime.

Outline

Function: data struct

```
1 struct Data {
2     std::ptrdiff_t width;
3     std::ptrdiff_t height;
4     std::vector<scalar_t> data;
5
6     scalar_t& idx(std::ptrdiff_t x, std::ptrdiff_t y) {
7         return data[x + width * y];
8     }
9 };
```

Function: get_input 1

```
1  Data get_input() {
2      const int heat = 100;
3
4      Data input;
5      input.width = 1000;
6      input.height = 1000;
7      input.data.resize(input.width*input.height, std::numeric_limits<scalar_t
       ↪ >::quiet_NaN());
8      //see next slide
9  }
```

Function: get_input 2

```
for (auto y = 0z; y < input.height; ++y) {
    for (auto x = 0z; x < input.width; ++x) {
        if (y == 0) {
            input.idx(x,y) = heat;
        } else if (x == 0 || x == input.width-1 || y == input.height-1) {
            input.idx(x,y) = 0;
        }
    }
}

return input;
```

Function: get_variable_coordinates

```
std::vector<Coordinate> variable_coordinates;

for (auto y = 0z; y < data.height; ++y) {
    for (auto x = 0z; x < data.width; ++x) {
        if (std::isnan(data.idx(x,y))) {
            if (is_border(x,y)) {
                throw std::runtime_error{"Error"};
            }
            variable_coordinates.push_back({x,y});
        }
    }
}
return variable_coordinates;
```

Function: make_first_guess

```
1 void make_first_guess(Data& data, const std::vector<Coordinate>&
      ↪ variable_coordinates) {
2     for (auto [x, y] : variable_coordinates) {
3         data.idx(x,y) = 0;
4     }
5 }
```

Function: zeitschritt

```
const auto max_iterations = 10000;
    auto i = 0;
    for (; i < max_iterations; ++i) {
        scalar_t residual = 0;
        for (auto [x,y] : variable_coordinates) {
            const auto old_value = data.idx(x,y);
            const auto average = (data.idx(x,y-1) + data.idx(x-1,y) + data.idx
    ↪ (x+1,y) + data.idx(x,y+1)) / 4;
            data.idx(x,y) = data.idx(x,y) + relaxation_factor * (average -
    ↪ data.idx(x,y));
            residual += std::pow(data.idx(x,y) - old_value, 2);
        }
        if (residual < precision) {
            break;
        }
    }
```

Problems with sequential version

Higher accuracy $\rightarrow$ Bigger Grid

- Needs more Performance

Bigger Grid $\rightarrow$ More iterations to converge

- Also needs more Performance

$\Rightarrow$ Parallelization needed!

Outline

Parallelization approach

Split up iterations or grid

- Iterations can't be split
  - Every iteration depends on previous one

Parallelization approach

Split up iterations or grid

- Iterations can't be split
  - Every iteration depends on previous one

Grid needs to be split!

- Means that boundarys have to be communicated

Parallelization difficulties

Find a way to split the grid

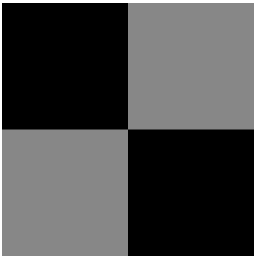


Figure: Split in 2 dimensions

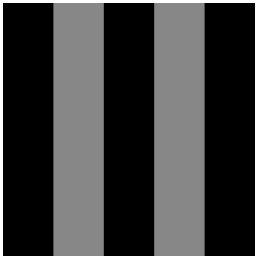


Figure: Split in 1 dimension

Parallelization difficulties

Find a way to split the grid

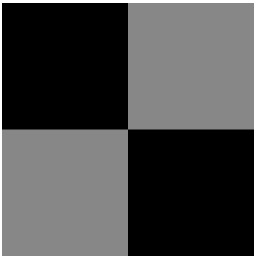


Figure: Split in 2 dimensions

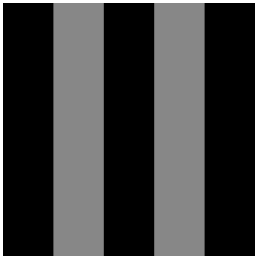


Figure: Split in 1 dimension

one dimension is easier to initialize!

Divide the grid

```
auto smaller_tasks_size = input.global_width / world.size();
auto smaller_task_amount = world.size() - input.global_width % world.size();

if(world.rank() < smaller_task_amount){
    input.width = lower_local_width
}
else{
    input.width = lower_local_width + 1
}

input.height = size;
```

Function: get_input

Uses data from grid division

Function: get_input

Uses data from grid division

Every thread gets it’s local borders...

Function: get_input

Uses data from grid division

Every thread gets it’s local borders...

...and storage space for it’s and it’s neighbours data

Function: get_input

Uses data from grid division

Every thread gets it’s local borders...

...and storage space for it’s and it’s neighbours data

Otherwise works as in the sequential

Parallelization difficulties

Communicating the borders

- After every iteration
- Using sendrecieve

```
if (const auto result = MPI_Sendrecv(
      &data.idx(0, 0), data.height, mpi_type_scalar, left_neighbor, 0,
      &data.idx(data.width, 0), data.height, mpi_type_scalar,
      right_neighbor, 0, cart, MPI_STATUS_IGNORE
);

result != MPI_SUCCESS) {
throw mpi::exception{"MPI_Sendrecv", result};
}
```

Parallelization difficulties

Residual needs to be communicated

- Only happens every 100 iterations

```
1  if constexpr (with_residual) {
2    residual = mpi::all_reduce(cart, residual, std::plus<>{});
3    return residual;
4  }
```

Outline

Strong scaling

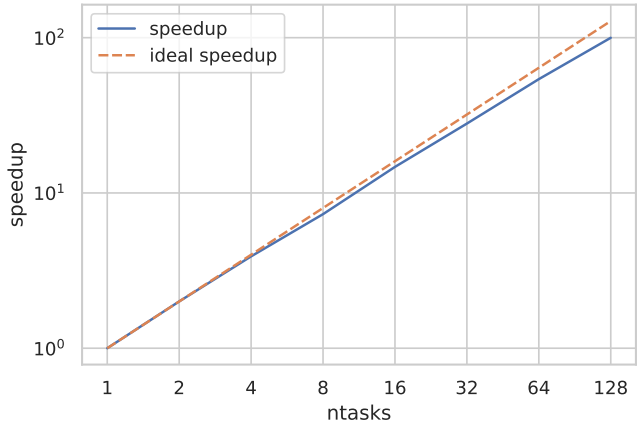
Strong scaling tests made with 2500x2500 grid

1, 2, 4, 8, 16, 32, 64 and 128 threads

128 threads on more than one node

Strong scaling graph

128 Tasks: $99/128 = 0.778$



Weak scaling

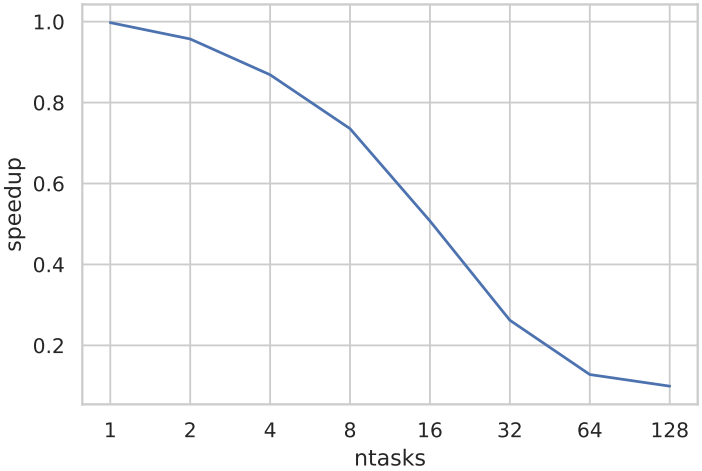
Also 1, 2, 4, 8, 16, 32, 64 and 128 tasks

Iterations are fixed at 20000

$n^2 = 1000^2 \cdot \text{tasks}$

$n = 1000 \cdot \sqrt{\text{tasks}}$

Weak scaling graph



Interpretation

$2 \cdot \text{test}_{16} \approx \text{test}_{32}$

Suboptimal grid division

- per task communication $\propto n$

Scorep

32 task experiment

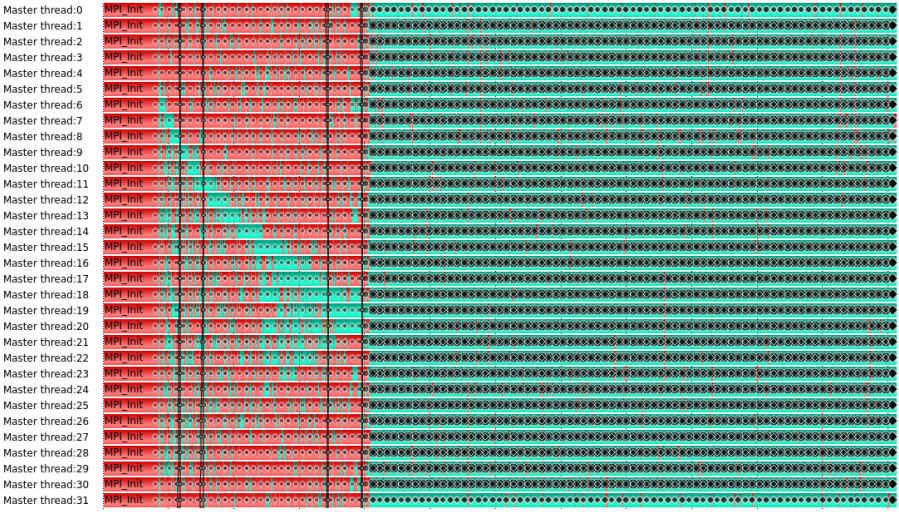
Scorep instrumentation

Profile

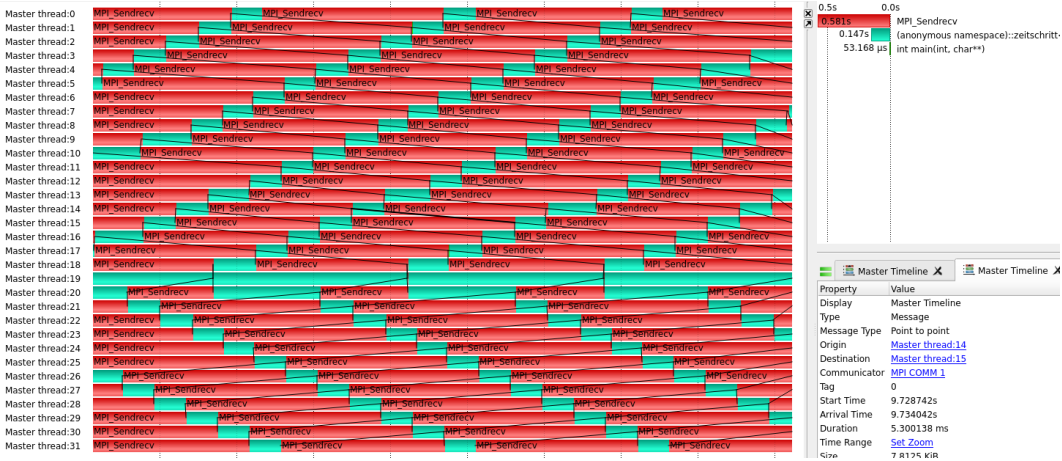
- 70.2% in zeitschritt without residual
- 20.5% in MPI_Sendrecv
- 0.7% in zeitschritt with residual

Run with tracing

Vampir


Master thread:0
Master thread:1
Master thread:2
Master thread:3
Master thread:4
Master thread:5
Master thread:6
Master thread:7
Master thread:8
Master thread:9
Master thread:10
Master thread:11
Master thread:12
Master thread:13
Master thread:14
Master thread:15
Master thread:16
Master thread:17
Master thread:18
Master thread:19
Master thread:20
Master thread:21
Master thread:22
Master thread:23
Master thread:24
Master thread:25
Master thread:26
Master thread:27
Master thread:28
Master thread:29
Master thread:30
Master thread:31
MPI_Init

Vampir


Master thread:0
Master thread:1
Master thread:2
Master thread:3
Master thread:4
Master thread:5
Master thread:6
Master thread:7
Master thread:8
Master thread:9
Master thread:10
Master thread:11
Master thread:12
Master thread:13
Master thread:14
Master thread:15
Master thread:16
Master thread:17
Master thread:18
Master thread:19
Master thread:20
Master thread:21
Master thread:22
Master thread:23
Master thread:24
Master thread:25
Master thread:26
Master thread:27
Master thread:28
Master thread:29
Master thread:30
Master thread:31
MPI_Sendrecv
0.5s
0.0s
0.581s
MPI_Sendrecv
0.147s
(anonymous namespace)::zeitschritt
53.168 µs
int main(int, char**)
Master Timeline
Master Timeline
Property
Value
Display
Master Timeline
Type
Message
Message Type
Point to point
Origin
Master thread:14
Destination
Master thread:15
Communicator
MPI COMM 1
Tag
0
Start Time
9.728742s
Arrival Time
9.734042s
Duration
5.300138 ms
Time Range
Set Zoom
Size
7.8125 KiB

Denormalized Floating Point

```
auto grid_point_step(float a, float b, float c, float d, float prev_result) {
  const auto average = (a + b + c + d) / 4;
  const auto result = prev_result + 1.7 * (average - prev_result);
  return result;
}
static void normal(benchmark::State& state) {
  volatile float a = 1;
  volatile float b = 1;
  volatile float c = 1;
  volatile float d = 1;
  volatile float prev_result = 1;
  for (auto _ : state) {
    const auto result = grid_point_step(a, b, c, d, prev_result);
    benchmark::DoNotOptimize(result);
  }
}
```

Denormalized Floating Point

```cpp
static void neighbors_denormal(benchmark::State& state) {
  volatile float a = std::numeric_limits<float>::denorm_min();
  volatile float b = std::numeric_limits<float>::denorm_min();
  volatile float c = std::numeric_limits<float>::denorm_min();
  volatile float d = std::numeric_limits<float>::denorm_min();
  volatile float prev_result = 1;
  for (auto _ : state) {
    const auto result = grid_point_step(a, b, c, d, prev_result);
    benchmark::DoNotOptimize(result);
  }
}
```

Denormalized Floating Point

normal: 1.95 ns

neighbors_denormal: 50.9 ns

simple fix: $\text{initial guess} = 1$

28s$\rightarrow$21s

Improved Vampir profile

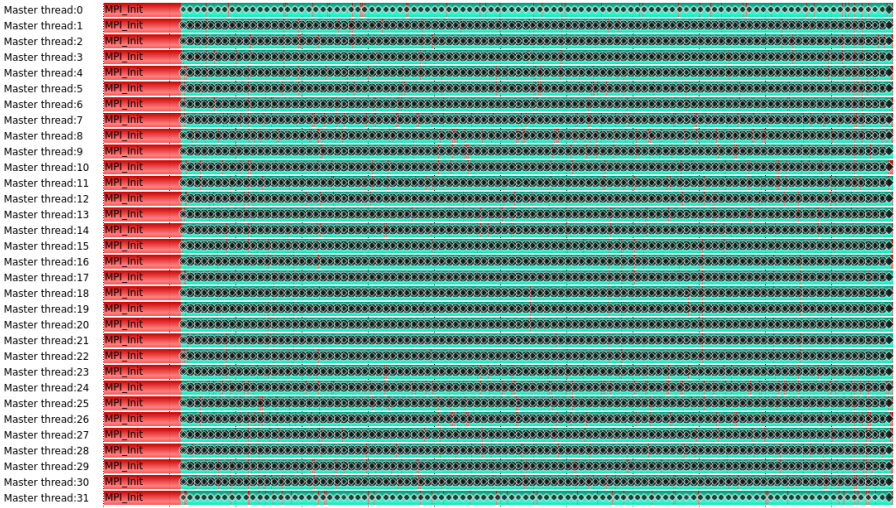
85.7% in zeitschritt without residual

9.8% in MPI_Init

3.0% in MPI_Sendrecv

0.9% in zeitschritt with residual

Improved Vampir profile



Conclusion

Goals achived?

Conclusion

Goals achived?

- Yes!

Conclusion

Goals achived?

- Yes!

Lots of Performance improvements

Conclusion

Goals achived?

- Yes!

Lots of Performance improvements

n-Dimensional

Conclusion

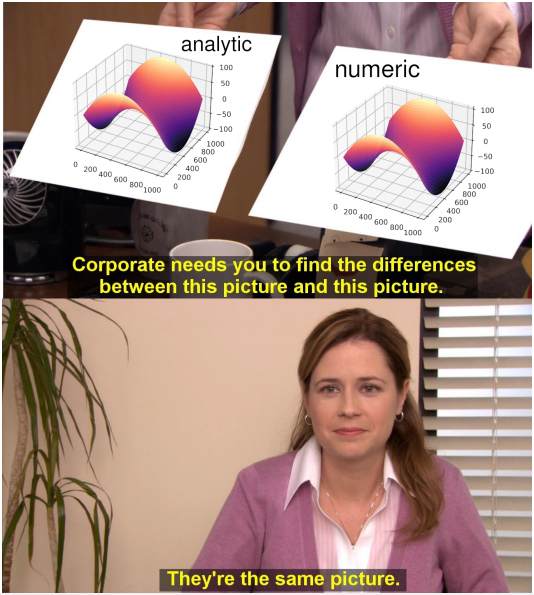
Goals achived?

- Yes!

Lots of Performance improvements

n-Dimensional

Input-file


analytic
numeric
Corporate needs you to find the differences between this picture and this picture.
They're the same picture.