



Jakob Hördt, Philipp Müller

Numerical Solution of Laplace's Equation

Implementing a high performance solver and conducting performance analysis

Performance Analysis

Conclusion

Table of contents

- 1 The Problem
- 2 Our Solution
- **3** Sequential Solution
- 4 Parallelized Solution
- 5 Performance Analysis



Our Solution

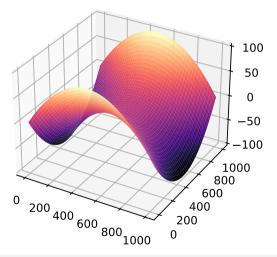
Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

The Laplace equation



$f:\mathbb{R}^n ightarrow\mathbb{R}$
$\Delta f=0$
$\Delta f = abla^2 f = \operatorname{div} \operatorname{grad} f$
$\nabla = \left(\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n}\right)$
left: $f(x,y) \approx x^2 - y^2$
$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 2 - 2 = 0 \checkmark$

Our Solution

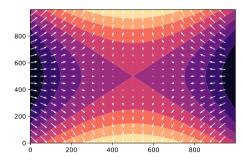
Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Laplace Operator



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

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

derivative of f

Our Solution

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Intuition

 $\Delta f = 0$

No max or min

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

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:

$$abla g = -4\pi G
ho$$
 $g = -
abla \phi$

Poisson's equation for gravitational fields:

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

$$ho = \mathbf{0}
ightarrow \Delta \phi = \mathbf{0}$$

Laplace's equation for

Heat equation:

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

 $rac{\partial}{\partial t} u(x,t) = 0
ightarrow ext{Laplace}$ equation

Electrostatics:

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

 $abla \mathbf{E} =
abla (abla V_{\mathbf{E}})$

 $= -\Delta V_{\rm E} =
ho arepsilon_{
m 0}$

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

Boundary Conditions

Laplace equation

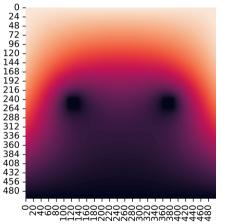
- infinitely many solutions
- Dirichlet boundary conditions
 - fixed continuous values at some boundary
- Dirichlet boundary conditions
 - unique solution

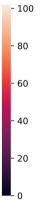
The Problem Our Solution Sequential Solution 0000000 000000 00000000 000000000000000000000000000000000000	
---	--

Performance Analysis

Conclusion

Solutions





Analytic solution infeasible

Our Solution

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Goal

We created numeric solver

Scales to large problem size on a cluster

Our Solution

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Outline

1 The Problem

2 Our Solution

3 Sequential Solution

- 4 Parallelized Solution
- 5 Performance Analysis



Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Finite element method

Our Solution

00000

Discretize domain into grid cells with spacing *h*:

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

Performance Analysis

Conclusion

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:

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

Jakob Hördt, Philipp Müller

Performance Analysis

Conclusion

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

\blacksquare *n* × *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.

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Overrelaxation

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

next = u + w * (average - u)

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

Our Solution

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Outline

1 The Problem

2 Our Solution

3 Sequential SolutionFunctions

Problems with Sequential

4 Parallelized Solution

5 Performance Analysis



Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Function: data struct

```
struct Data {
   std::ptrdiff_t width;
   std::ptrdiff_t height;
   std::vector<scalar_t> data;
   scalar_t& idx(std::ptrdiff_t x, std::ptrdiff_t y) {
       return data[x + width * y];
   }
  };
```

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Function: get_input 1

 Our Solution
 Sequential Solution
 Parallelized Solution

 0000000
 0000000
 0000000
 0000000

Performance Analysis

Function: get_input 2

```
1 for (auto y = 0z; y < input.height; ++y) {
      for (auto x = 0z; x < input.width; ++x) {
2
          if (v == 0) \{
3
               input.idx(x,y) = heat;
4
          } else if (x == 0 || x == input.width-1 || y == input.height-1) {
               input.idx(x,y) = 0:
7
      }
8
9 }
11 return input;
```

The Problem Our Solution

Parallelized Solution

Performance Analysis

Conclusion

Function: get_variable_coordinates

```
std::vector<Coordinate> variable_coordinates;
  for (auto y = 0z; y < data.height; ++y) {
3
      for (auto x = 0z: x < data.width: ++x) {
4
          if (std::isnan(data.idx(x,y))) {
5
              if (is_border(x,v)) {
                   throw std::runtime_error{"Error"}:
              variable_coordinates.push_back({x,v});
9
13 return variable_coordinates;
```

The ProblemOur Solution0000000000000

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Function: make_first_guess

```
The Problem
```

Our Solution Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Function: zeitschritt

```
1 const auto max_iterations = 10000:
       auto i = 0;
       for (: i < max_iterations; ++i) {</pre>
 3
           scalar_t residual = 0:
 4
           for (auto [x,y] : variable_coordinates) {
 5
                const auto old_value = data.idx(x.v):
                const auto average = (data.idx(x,y-1) + data.idx(x-1,y) + data.idx
       \hookrightarrow (x+1,v) + data.idx(x,v+1)) / 4;
                data.idx(x.v) = data.idx(x,y) + relaxation_factor * (average -
       \hookrightarrow data.idx(x,y)):
                residual += std::pow(data.idx(x,y) - old_value, 2);
 9
           if (residual < precision) {</pre>
11
                break:
13
       }
14
```

Performance Analysis

Conclusion

Problems with sequential version

$\blacksquare Higher accuracy \rightarrow Bigger Grid$

- Needs more Performance
- $\blacksquare \ Bigger \ Grid \rightarrow More \ iterations \ to \ converge$
 - Also needs more Performance

 \Rightarrow Parallelization needed!

Our Solution

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Outline

- 1 The Problem
- 2 Our Solution
- **3** Sequential Solution
- 4 Parallelized Solution
 Parallelization Approach
 Parallelization Difficulties

5 Performance Analysis



Performance Analysis

Conclusion

Parallelization approach

Our Solution

Split up iterations or grid

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

Parallelized Solution ●●●●●●● Performance Analysis

Conclusion

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

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Parallelization difficulties

Our Solution

Find a way to split the grid



Figure: Split in 2 dimensions



Figure: Split in 1 dimension

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Parallelization difficulties

Our Solution

Find a way to split the grid



Figure: Split in 2 dimensions



Figure: Split in 1 dimension

one dimension is easier to initialize!

Jakob Hördt, Philipp Müller

Divide the grid

```
all auto smaller_tasks_size = input.global_width / world.size();
2 auto smaller_task_amount = world.size() - input.global_width % world.size();
3
4 if(world.rank() < smaller_task_amount){</pre>
      input.width = lower_local_width
5
6 }
7 else{
      input.width = lower_local_width + 1
8
}
9
 input.height = size;
```

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Function: get_input

Our Solution

Uses data from grid division

Jakob Hördt, Philipp Müller

Performance Analysis

Conclusion

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};
}
```

Performance Analysis

Conclusion

Parallelization difficulties

Our Solution

Residual needs to be communicated

Only happens every 100 iterations

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

Our Solution

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Outline

- 1 The Problem
- 2 Our Solution
- **3** Sequential Solution
- 4 Parallelized Solution
- 5 Performance Analysis
 - Strong Scaling
 - Weak Scaling
 - Vampir

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Strong scaling

Our Solution

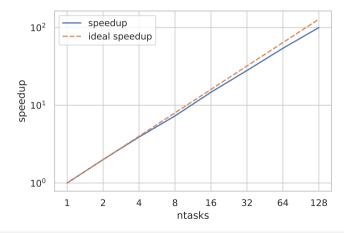
Strong scaling tests made with 2500x2500 grid

- 1, 2, 4, 8, 16, 32, 64 and 128 threads
- 128 threads on more than one node

 The Problem
 Our Solution
 Sequential Solution
 Parallelized Solution
 Performance Analysis
 Conclusion

 Strong scaling graph

■ 128 Tasks: 99/128 = 0.778



Parallelized Solution

Performance Analysis

Conclusion

Weak scaling

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

- Iterations are fixed at 20000
- $\blacksquare n^2 = 1000^2 \cdot \text{tasks}$

Our Solution

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

The Problem Our Solution

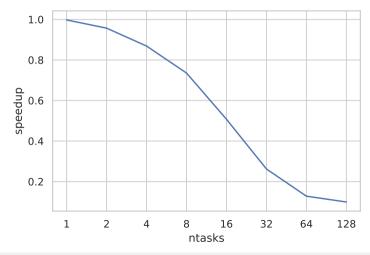
Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Weak scaling graph



Jakob Hördt, Philipp Müller

Our Solution

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Interpretation

- $\blacksquare \ 2 \cdot test_{16} \approx test_{32}$
- Suboptimal grid division
 - ▶ per task communication $\propto n$

Our Solution

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

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

Our Solution

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

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

		Addbbb db call a boo de		
Master thread:0		Contract of the state of the st	11.1	
Master thread:1				***************************************
Master thread:2		ၜၛၣၜ႞ၣၜၜၜၜၜၜၜၜၜၜ႞ၜၣ႞ႜႜႜႜႜႜ		•
Master thread:3			0000	***************************************
Master thread:4	MPI_Init_opoo	o do bo bo do	00 <mark>00</mark>	•
Master thread:5	MPI_Init opooo			
Master thread:6	MPI_Init_o <mark>o</mark> ooooo		pp of	•
Master thread:7	MPI_Inito <mark>ooo</mark> oooo	••••••••••••••	0 0 0	
Master thread:8	MPI_Initoo_ooooo		0000	**************************************
Master thread:9	MPI_Initoo_eeooo	ၜၣၣၜၜၣၜၜၣၜၣၜၜၜၜၜၜၜၜၜၜၜၜ	0000	**************************************
Master thread:10	MPI_Init opposition	စာ စုရ ရ စု ရ စု စု ရ ရ စု ရ ရ စု ရ ရ စု ရ ရ စု ရ	0000	
Master thread:11	MPI_Init ooooooo	a o po	0000	***************************************
Master thread:12	MPI_Init oo oo oo oo	o contro do polo picto :	60 D C	***************************************
Master thread:13	MPI_Init oo oo	Apido oci o pido a dojajo po celi	000	
Master thread:14	MPI_Init_ooo	64 00 0000 0000 0000 0000		
Master thread:15	MPI Init operation		0000	
Master thread:16	MPI_Init_ooooo	a codo doc o coo cido o co	0000	
Master thread:17	MPI_Init_ood_oodo		opoc	
Master thread:18	MPI Init on one		0000	
Master thread:19	MPI Init one end			***************************************
Master thread:20	MPI Init oppopol			
Master thread:21	MPI Init oceano		0000	
Master thread:22	MPI Init of designed			
Master thread:23	MPI Init ocidente		00.00	
Master thread:24			0000	
Master thread:24				
Master thread:26	MPI Init			
Master thread:20	MPI Init •0 00000	01-01-01-01-00-00-01-01-01-01-01-01-01-0	0000	
Master thread:28	MPI Init opposition	1000 YOF	1	
Master thread:29				
Master thread:30	MPI_Init_ooddoodd			
Master thread:31	MPI_Init_opoo	မ္ရာ စုံစုရင္ စစ္ ေစာ့ရာစု စုံစုရြက္ရေး	oope	•••••••••••••••••••••••••••••••••••••••

Our Solution

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Vampir

Mast	er	thre	ad:	0
Mast	er	thre	ad:	1
Mast	er	thre	ad:	2
Mast	er	thre	ad:	3
Mast	er	thre	ad:	4
Mast	er	thre	ad:	5
Mast	er	thre	ad:	6
Mast	er	thre	ad:	7
Mast	er	thre	ad:	8
Mast	er	thre	ad:	9
Mast	er	thre	ad:	10
Mast	er	thre	ad:	11
Mast	er	thre	ad:	12
Mast	er	thre	ad:	13
Mast	er	thre	ad:	14
Mast	er	thre	ad:	15
Mast	er	thre	ad:	16
Mast	er	thre	ad:	17
Mast	er	thre	ad:	18
Mast	er	thre	ad:	19
Mast	er	thre	ad:	20
Mast	er	thre	ad:	21
Mast	er	thre	ad:	27
Mast	er	thre	ad:	23
Mast	er	thre	ad:	24
Mast	er	thre	ad:	25
Mast	er	thre	ad:	26
Mast	er	thre	ad:	27
Mast				
Mast				
Mast				
Mast	er	thre	ad:	31

ster thread:0	MPI_Sendrecv	MPI_Sendrecv	MPL Sen	drecv	MPI Sendrecv	X	0.0s
ster thread:1	MPI_Sendrecv	MPI_Sendrecv	Irecy MPI Sendrecy		MPL Sendrecv	0.581s 0.147s	MPI_Sendrecv
ster thread:2	MPI_Sendrecv MPI_S	endrecv	MPI Sendrecv	MP	Sendrecv	53.168 µs	(anonymous namespace)::zeitschritt- s int main(int. char**)
ster thread:3	MPI_Sendrecv		MPI Sendrecv	MPL Sendr	ecv	55.106 µs	int main(int, char**)
ster thread:4	MPI_Sendrecv	MPLS	endrecv	MPI Sendrecv			
ster thread:5	MPI_Sendrecv	MPI Sendrecy	M	PI Sendrecv	MPI_Sendrecv		
ster thread:6	MPI_Sendrecv	MPI_Sendrecv	MPI Send	irecv	MPI_Sendrecv		
ster thread:7	MPI_Sendrecv	MPI_Sendrecv	MPI_Sendrecv		MPI_Sendrecv		
ster thread:8	MPI_Sendrecv	IPI Sendrecv	MPI_Sendrecv	MPL	Sendrecv		
ster thread:9	MPI_Ser	ndrecv	MPI Sendrecv	MPI_Sendred	v v		
ster thread:10	MPI_Sendrecv	MPL Se	endrecv	MPI_Sendrecv	MPI_Sendrecv		
ster thread:11	MPI_Sendrecv	MPI Sendrecv	MPI	Sendrecv	MPI Sendrecv		
ster thread:12	MPI_Sendrecv	MPI_Sendrecv	MPI_Sendre	ecv.	MPI Sendrecv		
ster thread:13	MPI_Sendrecv	MPI_Sendrecv	MPI_Sendrecv		MPI Sendrecv		
ster thread:14	MPI_Sendrecv MPI	Sendrecv	MPI Sendrecv	MPI	Sendrecv		
ster thread:15	MPI_Sendre	CV MPL:	Sendrecv	MPI_Sendrecv			
ster thread:16	MPI_Sendrecv	MPI_Sendred	v	MPI Sendrecv	MPI_Sendrecv		
ster thread:17	MPI_Sendrecv	MPI Sendrecv	MPI Se	endrecv	MPI Sendrecv		
ster thread:18	MPI_Sendrecv	MPI_Sendrecv	MPI_Sendrecv		MPI_Sendrecv	- III Masta	r Timeline 🗙 🗮 Master Timeline 🎗
ster thread:19							
ster thread:20	MP1_Sendrecv	MPT_Sendre		MPI_Sendrecv	MPT_Sendrecv	Property	Value
ster thread:21	MPI_Sendrecv		Sendrecv	MPI_Sendrecv		Display	Master Timeline
ster thread:22	MPI_Sendrecv MPI_Se		MPI_Sendrecv	MPI_Se		Type	Message
ster thread:23	MPI_Sendrecv	MPI_Sendrecv	MPI_Sendrecv		MPI_Sendrecv	Message Type	
ster thread:24	MPI_Sendrecv	MPI_Sendrecv	MPI_Sendrecv		MPI_Sendrecv	Origin	Master thread:14
ster thread:25	MPI_Sendrecv	MPI_Sendrecv	MPI_Ser		MPI_Sendrecv	Destination	Master thread:15
ster thread:26	MPI_Sendrecv	MPI_Sendrec		MPI_Sendrecv	MPI_Sendrecv	Communicator	
ster thread:27	MPI_Sendrecv		endrecv	MPI_Sendrecv		Tag	0
ster thread:28	MPI_Senc		MPI_Sendrecv	MPI_Send		Start Time	9.728742s
ster thread:29		Pl_Sendrecv	MPI_Sendrecv	M	PI_Sendrecv	Arrival Time	9.734042s
ster thread:30	MPI_Sendrecv	MPI_Sendrecv	MPI_Sendrecv		MPI_Sendrecv	Duration	5.300138 ms
ster thread:31	MPI_Sendrecv	MPI_Sendrecv	MPI_Sen	drecv	MPt_Sendrecv	Time Range	Set Zoom
		1				Size	7 8175 KIR

 Dur Solution
 Sequential Solution
 Parallelized Solution
 Performance Analysis
 Conclusion

 0000000
 0000000
 0000000
 0000000
 0000000
 0000000

Denormalized Floating Point

```
1 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:
4
5 }
6 static void normal(benchmark::State& state) {
    volatile float a = 1:
7
    volatile float b = 1:
8
    volatile float c = 1:
9
    volatile float d = 1:
    volatile float prev_result = 1:
11
    for (auto _ : state) {
      const auto result = grid_point_step(a, b, c, d, prev_result);
13
      benchmark::DoNotOptimize(result):
14
15
16 }
```

Parallelized Solution

Denormalized Floating Point

Our Solution

```
static void neighbors_denormal(benchmark::State& state) {
    volatile float a = std::numeric limits<float>::denorm min():
2
    volatile float b = std::numeric_limits<float>::denorm_min();
3
    volatile float c = std::numeric_limits<float>::denorm_min():
4
    volatile float d = std::numeric limits<float>::denorm min():
5
    volatile float prev_result = 1;
    for (auto _ : state) {
7
      const auto result = grid_point_step(a, b, c, d, prev_result);
8
      benchmark::DoNotOptimize(result);
g
11
 1
```

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Denormalized Floating Point

Our Solution

normal: 1.95 ns

- neighbors_denormal: 50.9 ns
- **simple fix:** initial guess = 1

■ 28s→21s

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Improved Vampir profile

Our Solution

85.7% in zeitschritt without residual

■ 9.8% in MPI_Init

■ 3.0% in MPI_Sendrecv

0.9% in zeitschritt with residual

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion

Improved Vampir profile

Master thread:0	MPI_Init	
Master thread:1	MPI_Init	***
Master thread:2	MPI_Init	***
Master thread:3	MPI_Init	
Master thread:4	MPI_Init	
Master thread:5	MPI_Init	***
Master thread:6	MPI_Init	***
Master thread:7	MPI_Init	
Master thread:8	MPI_Init	
Master thread:9	MPI_Init	***
Master thread:10	MPI_Init	* 33
Master thread:11	MPI_Init	***
Master thread:12	MPI_Init	***
Master thread:13	MPI_Init	***
Master thread:14	MPI_Init	***
Master thread:15	MPI_Init	***
Master thread:16	MPI_Init	***
Master thread:17	MPI_Init	***
Master thread:18	MPI_Init	***
Master thread:19	MPI_Init	
Master thread:20	MPI_Init	***
Master thread:21	MPI_Init	***
Master thread:22	MPI_Init	
Master thread:23	MPI_Init	
Master thread:24	MPI_Init	***
Master thread:25	MPI_Init	***
Master thread:26	MPI_Init	***
Master thread:27	MPI_Init	
Master thread:28	MPI_Init	***
Master thread:29	MPI_Init	***
Master thread:30	MPI_Init	***
Master thread:31	MPI_Init	000

0	MPI_Init
1	MPI_Init
2	MPI Init
3	MPI_Init ************************************
4	MPI_Init
5	MPI_Init
б	MPI_Init
7	MPI_Init
в	MPI_Init
9	MPI_Init
10	MPI_Init
11	MPI_Init
12	MPI_Init
13	MP_Init
14	MPI_Init
15	MPI_Init
16	MPI_Init
17	MPI_Init
18	MPI_Init
19	MPI_Init
20	MPI_Init
21	MPI_Init
22	MPI_Init
23	MPI_Init
24	MPI_Init
25	MPI_Init
26	MPI_Init
27	MPI_Init
28	MPI_Init********************************
29	MPI Init
30	MPI_Init
31	MP_Init

Sec

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion ●○

Conclusion

Goals achived?

Our Solution

000000

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion ●○

Conclusion

Goals achived?

Our Solution

Yes!

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion ●○

Conclusion

Goals achived?

Our Solution

Yes!

Lots of Performance improvements

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion ●○

Conclusion

Goals achived?

Our Solution

Yes!

Lots of Performance improvements

n-Dimensional

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion ●○

Conclusion

Goals achived?

Our Solution

Yes!

Lots of Performance improvements

n-Dimensional

Input-file

 The Problem
 Our Solution

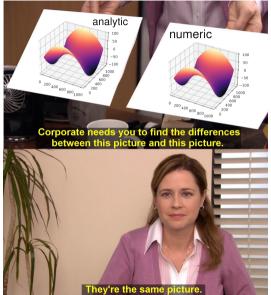
 0000000
 0000000

Sequential Solution

Parallelized Solution

Performance Analysis

Conclusion ○●



Jakob Hördt, Philipp Müller

PC HPC 2022