



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



# Advanced C++ Course

Linaro Forge

CSCS

# Introduction

## Linaro Forge tools

- MAP
  - "Linaro MAP is a high-performance (sampling) profiler designed to optimize the efficiency of software running on multicore processors."
  - Designed for 'hot-spot' analysis with stack traces and performance metrics
  - Supports C/C++/Fortran/Python
  - Nvidia & AMD GPU profiling (AMD not supported on CSCS)
  - Performance Reports
- DDT
  - "Linaro DDT is an advanced debugger designed to simplify the troubleshooting and optimization of complex, high-performance computing (HPC) applications."
  - Supports C/C++/Fortran/Python
  - Nvidia & AMD GPUs (AMD not supported on CSCS)
  - Supports containers (to be tested with CE)

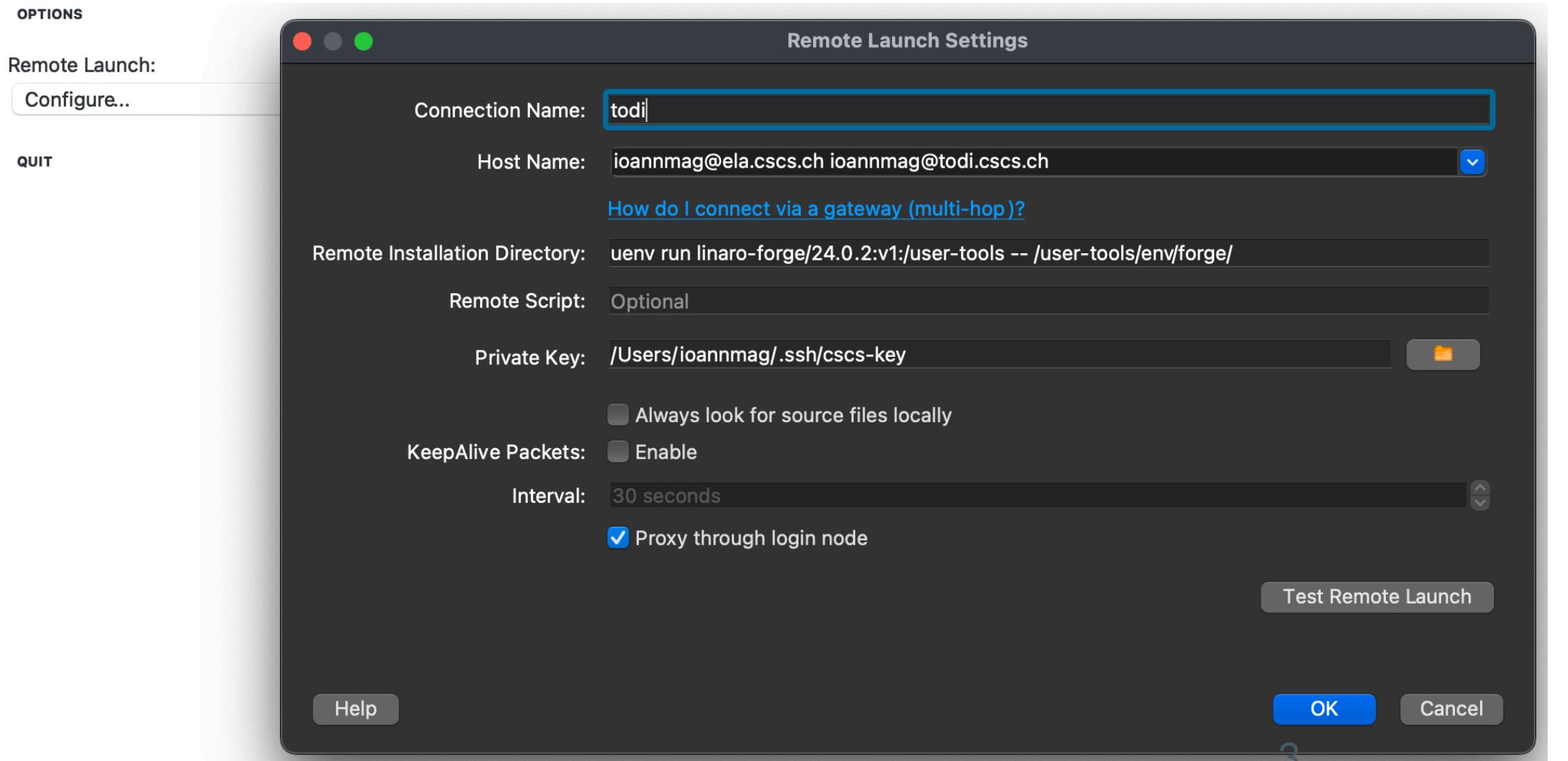
# Resources

- [\[Knowledge Base\] Debugging](#)
- [Linaro Forge UENV](#)
- [Client Download](#)

# Forge client configuration

- Forge client on local machine (Linux, MacOS, Windows)
- License
  - Max processes 272 / Max users 256
  - We pay for support so if any feature is missing RFE is possible

## TODI remote setup



# MAP

## Instructions

### Todi

```
uenv start --view=linaro:forge,prgen-gnu:default prgen-gnu/24.7 linaro-forge/24.0.2  
map -n 4 --mpi=slurm --mpiargs="-A csstaff -p debug -t 10" --profile <executable> <executable_arguments>
```

### Local machine

- Connect to todi
- LOAD PROFILE DATA FILE

## Instructions

### Local machine

- Connect to todi
- MANUAL LAUNCH (ADVANCED)
- Create correct listener

### Todi

```
uenv start --view=linaro:forge,prgenv-gnu:default prgenv-gnu/24.7 linaro-forge/24.0.2  
srun -n 4 -A csstaff -t30 -p debug ddt-client <executable> <executable_arguments>
```

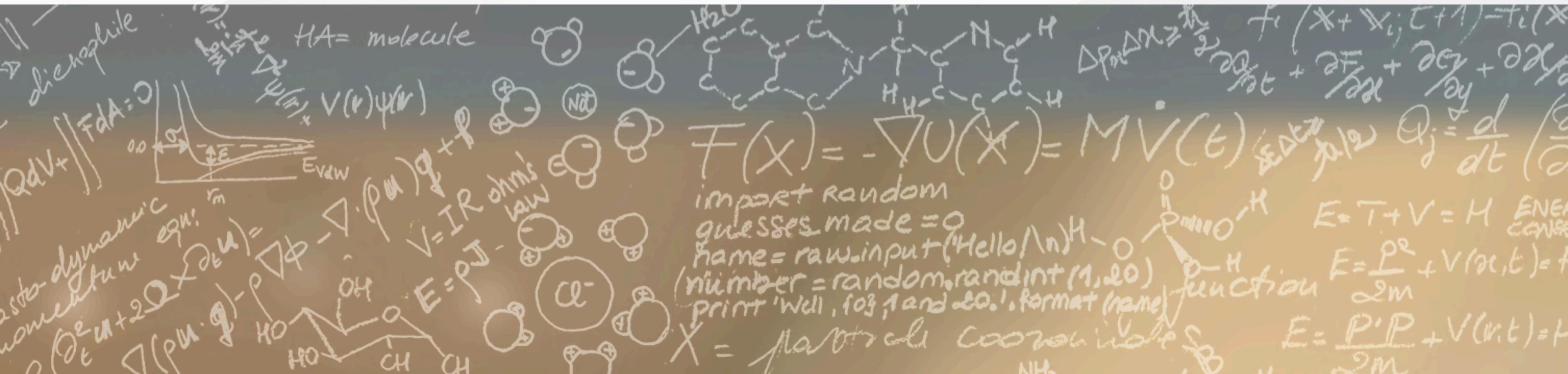
# Demo



CSCS

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

ETH zürich



# Questions?

# Resources

- [Linaro MAP website](#)
- [Linaro DDT website](#)
- [Ensuring Program Correctness with Linaro DDT Rudy Shand](#)

# Backup slides

# DDT

# linaro forge



[Get trial licence](#)

[Support](#)

[linaroforge.com](#)

license Serial: 17741\_2

#### RUN

Run and debug a program.

#### ATTACH

Attach to an already running program.

#### OPEN CORE

Open a core file from a previous run.

#### MANUAL LAUNCH (ADVANCED)

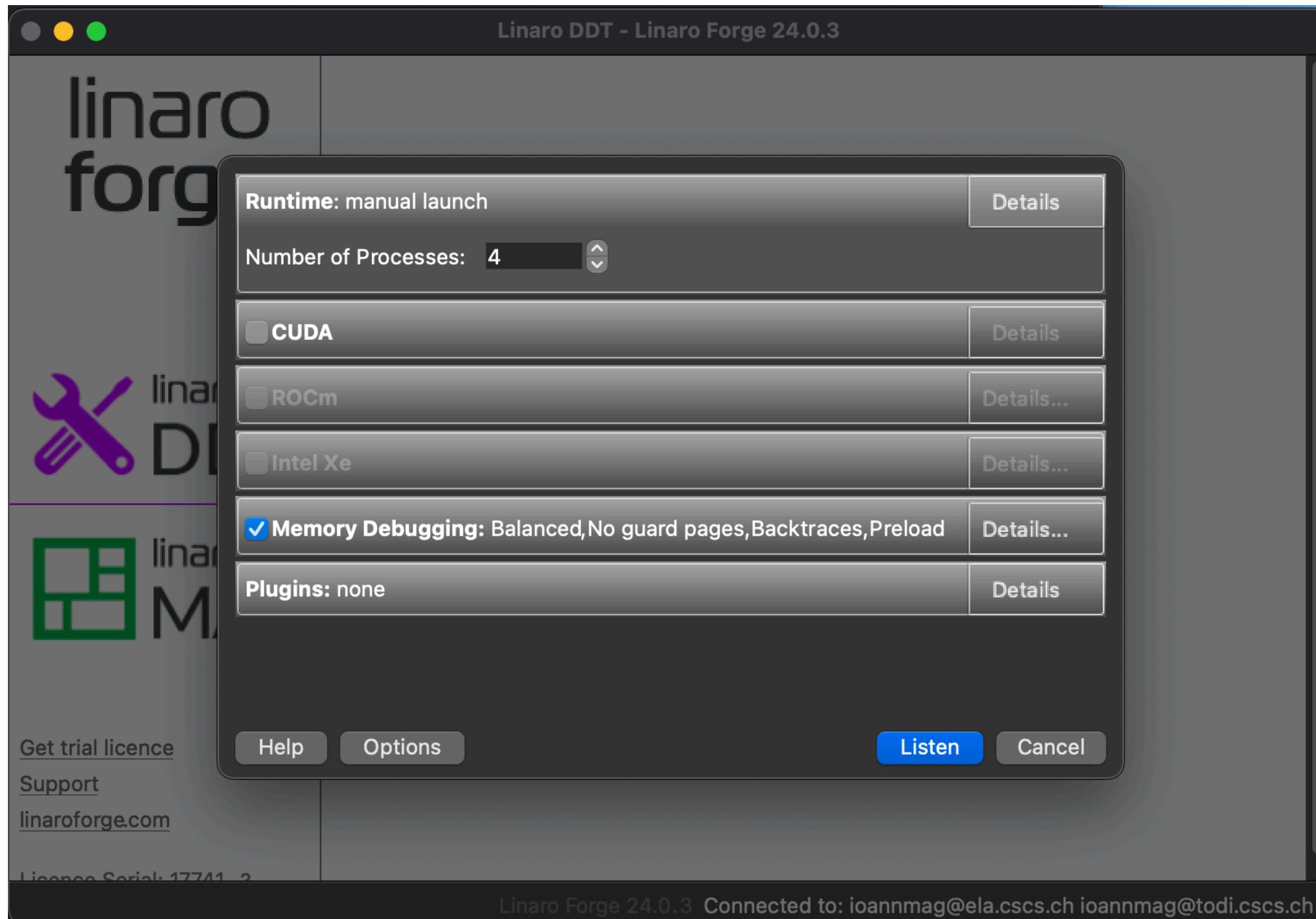
Manually launch the backend yourself.

#### OPTIONS

Remote Launch:

#### QUIT

Linaro Forge 24.0.3 Connected to: ioannmag@ela.cscs.ch ioannmag@todi.cscs.ch



Current Group: All Focus on current: Group Process Thread Step Threads Together

All 0 1 2 3

Create Group

Project Files matrix\_mult...

Search (⌘K)

Application Code

- /
- Sources
  - matrix\_mult...
  - broadcast()
  - calculate()
  - main(int)
  - MatrixD
  - myrand()
  - print\_matrix()
  - receive()
  - send()
  - usage()
- External Code

```

108
109 int main(int argc, char* argv[])
110 {
111     if(argc != 2)
112     {
113         usage();
114         return 1;
115     }
116
117     MPI_Init(&argc, &argv);
118
119     const int dim = atoi(argv[1]);
120     const int data_size = dim * dim;
121     int my_rank;
122     int num_procs;
123
124     MatrixData mat_a(data_size);
125     MatrixData mat_b(data_size);
126
127     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
128     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
129
130     // generate the numbers on root node
131     if(my_rank == 0)
132     {
133         generate(mat_a.begin(), mat_a.end(), myrand<MatrixData::value_type>);
134         generate(mat_b.begin(), mat_b.end(), myrand<MatrixData::value_type>);
135     }
136
137     // distribute the data
138     broadcast(MPI_COMM_WORLD, 0, mat_a);
139     broadcast(MPI_COMM_WORLD, 0, mat_b);
140
141     // integer division
142     // don't care about the remainder - root will do rest
143     const int partition_size = data_size / num_procs;
144
145     if(my_rank == 0)
146     {
147         MatrixData mat_c(data_size);
148         const int start = partition_size * (num_procs - 1);
149         const int end = data_size;
150
151         // calculate matrix section
152         calculate(start, end, dim, mat_a, mat_b, mat_c.begin() + start);
153
154         // only receive if we have data
155         if(partition_size != 0)
156         {
157             // receive from other processes
158         }
159     }
160
161     MPI_Finalize();
162
163     return 0;
164 }

```

Locals Current Line(s) Current Stack

Name	Value
argc	2

Input/Output Breakpoints Watchpoints Stacks (All) Tracepoints Tracepoint Output Logbook

Processes Function

4 main (matrix\_mult.cpp:111)

Name	Value
p	<No symbol "p" in current context.>

Ready Connected to: ioannmag@ela.csccs.ch ioannmag@todi.csccs.ch

The screenshot shows a debugger interface with a code editor and a floating 'Add Breakpoint' dialog.

**Code Editor:**

- Title:** matrix\_mult....
- File:** ple/matrix\_mult.cpp
- Line Number:** 111
- Content:** The code implements MPI-based matrix multiplication. It includes MPI operations like MPI\_Comm\_rank, MPI\_Comm\_size, MPI\_Broadcast, and MPI\_Recv. It also includes local operations like generate, calculate, and print\_matrix.

**Add Breakpoint Dialog:**

- Location:**
  - Line File: ple/matrix\_mult.cpp
  - Function
- Applies To:**
  - All
  - Process: All
  - Thread: All
- Hit Limits:**
  - Start on the n-th pass: 0
  - Trigger every n-th pass: 1
  - Stop after n hits: Forever
- Condition:** (empty)
- Language:** Auto
- Buttons:** Help, Add, Cancel

Current Group: All Focus on current: Group Process Thread Step Threads Together

All 0 1 2 3

Create Group

\* Project File... Search (...) Application Source mat External C

```

114         return 1;
115     }
116
117     MPI_Init(&argc, &argv);
118
119     const int dim = atoi(argv[1]);
120     const int data_size = dim * dim;
121     int my_rank;
122     int num_procs;
123
124     MatrixData mat_a(data_size);
125     MatrixData mat_b(data_size);
126
127     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
128     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
129
130     // generate the numbers on root node
131     if(my_rank == 0)
132     {
133         generate(mat_a.begin(), mat_a.end(), myrand<int>());
134         generate(mat_b.begin(), mat_b.end(), myrand<int>());
135     }
136
137     // distribute the data
138     broadcast(MPI_COMM_WORLD, 0, mat_a);
139     broadcast(MPI_COMM_WORLD, 0, mat_b);
140
141     // integer division
142     // don't care about the remainder - root will do rest
143     const int partition_size = data_size / num_procs;
144
145     if(my_rank == 0)
146     {
147         MatrixData mat_c(data_size);
148         const int start = partition_size * (num_procs - 1);
149         const int end = data_size;
150
151         // calculate matrix section
152         calculate(start, end, dim, mat_a, mat_b, mat_c.begin() + start);
153
154         // only receive if we have data
155         if(partition_size != 0)
156         {
157             // receive from other processes
158             for(int p = 1; p < num_procs; ++p)
159             {
160                 const int offset = partition_size * (p - 1);
161                 receive(MPI_COMM_WORLD, p, mat_c, offset, partition_size);
162             }
163         }
164     }

```

Processes 0-3:  
Process stopped at breakpoint in main (matrix\_mult.cpp:138).

Always show this window for user-defined breakpoints

Continue | Pause



Input/Output Breakpoints Watchpoints Stacks (All) Tracepoints Tracepoint Output Logbook  
Processes | Function ^ |  
4 main (matrix\_mult.cpp:138)

Locals Current Line(s) Current Stack	
Current Line(s)	
Name	Value
> mat_a	std::vector of length 16, capacity ...
Evaluate	
Name	Value
p	<No symbol "p" in current context.>

All      0 1 2 3

Create Group

Project File... matrix\_mult...

Search (...

Application / Source matrix\_mult.c

External C

```

121     int my_rank;
122     int num_procs;
123
124     MatrixData mat_a(data_size);
125     MatrixData mat_b(data_size);
126
127     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
128     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
129
130     // generate the numbers on root node
131     if(my_rank == 0)
132     {
133         generate(mat_a.begin(), mat_a.end(), myrand<MatrixData::value_type>);
134         generate(mat_b.begin(), mat_b.end(), myrand<MatrixData::value_type>);
135     }
136
137     // distribute the data
138     broadcast(MPI_COMM_WORLD, 0, mat_a);
139     broadcast(MPI_COMM_WORLD, 0, mat_b);
140
141     // integer division
142     // don't care about the remainder - root will do rest
143     const int partition_size = data_size / num_procs;
144
145     if(my_rank == 0)
146     {
147         MatrixData mat_c(data_size);
148         const int start = partition_size * (num_procs - 1);
149         const int end = data_size;
150
151         // calculate matrix section
152         calculate(start, end, dim, mat_a, mat_b, mat_c.begin() + start);
153
154         // only receive if we have data
155         if(partition_size != 0)
156         {
157             // receive from other processes
158             for(int p = 1; p < num_procs; ++p)
159             {
160                 const int offset = partition_size * (p - 1);
161                 receive(MPI_COMM_WORLD, p, mat_c, offset, partition_size);
162             }
163         }
164
165         print_matrix("Matrix A", mat_a, dim);
166         print_matrix("Matrix B", mat_b, dim);
167         print_matrix("Matrix C", mat_c, dim);
168     }
169     else
170     {

```

Locals Current Line(s) Current Stack

Name	Value
argc	2
> argv	0xffffffffc058
dim	4
data_size	16
my_rank	3
num_procs	4
mat_a	std::vector of length 16, capacity 16
[0]	3
[1]	6
[2]	7
[3]	5
[4]	3
[5]	5
[6]	6
[7]	2
[8]	9
[9]	1
[10]	2
[11]	7
[12]	0
[13]	9
[14]	3
[15]	6
mat_b	std::vector of length 16, capacity 16
[0]	0
[1]	6
[2]	2
[3]	6
[4]	1
[5]	8
[6]	7
[7]	9
[8]	2
[9]	0
[10]	2

```
149     const int end = data_size;
150
151     // calculate matrix section
152     calculate(start, end, dim, mat_a, mat_b, mat_c.begin() + start);
153
154     // only receive if we have data
155     if(partition_size != 0)
156     {
157         // receive from other processes
158         for(int p = 1; p < num_procs; ++p)
159         {
160             const int offset = partition_size * (p - 1);
161             receive(MPI_COMM_WORLD, p, mat_c, offset, partition_size);
162         }
163     }
164
165     print_matrix("Matrix A", mat_a, dim);
166     print_matrix("Matrix B", mat_b, dim);
167     print_matrix("Matrix C", mat_c, dim);
168
169 } else
170 {
171     // check we have values to calculate
172     if(partition_size != 0)
173     {
174         MatrixData mat_c_section(partition_size);
175         const int start = partition_size * (my_rank - 1);
176         const int end = start + partition_size;
177
178         // calculate matrix section
179         calculate(start, end, dim, mat_a, mat_b, mat_c_section);
180     }
}

```

partition\_size

mat\_c

- Add breakpoint for All
- Add tracepoint for All (MPI\_COMM\_WORLD, p, mat\_c, offset, partition\_size)
- Run to here

Undo  
Redo

Cut  
Copy  
Paste  
Delete

Select All

Open in external editor  
Close

Type is: MatrixData  
Value is: std::vector of length 16, capacity 16

Input/Output | Breakpoints | Watchpoints | Stacks (All) | Tracepoints | Tracepoint Output | Logbook

Tracepoint Output

Processes | Values loaded

Evaluate

Name	Value
p	<No symbol "p" in current

Current Group: All Focus on current: Group Process Thread Step Threads Together

All 0 1 2 3

Create Group

Project File... matrix\_mult....

Search (...

Application / Source matrix\_mult.cpp

External C

```

114         return 1;
115     }
116
117     MPI_Init(&argc, &argv);
118
119     const int dim = atoi(argv[1]);
120     const int data_size = dim * dim;
121     int my_rank;
122     int num_procs;
123
124     MatrixData mat_a(data_size);
125     MatrixData mat_b(data_size);
126
127     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
128     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
129
130     // generate the numbers on root node
131     if(my_rank == 0)
132     {
133         generate(mat_a.begin(), mat_a.end(), myrand<int>());
134         generate(mat_b.begin(), mat_b.end(), myrand<int>());
135     }
136
137     // distribute the data
138     broadcast(MPI_COMM_WORLD, 0, mat_a);
139     broadcast(MPI_COMM_WORLD, 0, mat_b);
140
141     // integer division
142     // don't care about the remainder - root will do rest
143     const int partition_size = data_size / num_procs;
144
145     if(my_rank == 0)
146     {
147         MatrixData mat_c(data_size);
148         const int start = partition_size * (num_procs - 1);
149         const int end = data_size;
150
151         // calculate matrix section
152         calculate(start, end, dim, mat_a, mat_b, mat_c.begin() + start);
153
154         // only receive if we have data
155         if(partition_size != 0)
156         {
157             // receive from other processes
158             for(int p = 1; p < num_procs; ++p)
159             {
160                 const int offset = partition_size * (p - 1);
161                 receive(MPI_COMM_WORLD, p, mat_c, offset, partition_size);
162             }
163         }
164     }

```

Locals Current Line(s) Current Stack

Name Value

> mat\_a std::vector of length 16, capacity ...

Processes 0-3:

Process stopped at breakpoint in main (matrix\_mult.cpp:138).

Always show this window for user-defined breakpoints

Continue Pause

Input/Output Breakpoints Watchpoints Stacks (All) Tracepoints Tracepoint Output Logbook

Stacks (All)

Processes Function

4 main (matrix\_mult.cpp:138)

Evaluate

Name Value

p <No symbol "p" in current context.>

Current Group: All Focus on current: Group Process Thread Step Threads Together

All 0 1 2 3

Create Group

Project File... matrix\_mult....

Search (...

Application / Source matrix\_mult.cpp

External C

```

114         return 1;
115     }
116
117     MPI_Init(&argc, &argv);
118
119     const int dim = atoi(argv[1]);
120     const int data_size = dim * dim;
121     int my_rank;
122     int num_procs;
123
124     MatrixData mat_a(data_size);
125     MatrixData mat_b(data_size);
126
127     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
128     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
129
130     // generate the numbers on root node
131     if(my_rank == 0)
132     {
133         generate(mat_a.begin(), mat_a.end(), myrand<int>());
134         generate(mat_b.begin(), mat_b.end(), myrand<int>());
135     }
136
137     // distribute the data
138     broadcast(MPI_COMM_WORLD, 0, mat_a);
139     broadcast(MPI_COMM_WORLD, 0, mat_b);
140
141     // integer division
142     // don't care about the remainder - root will do rest
143     const int partition_size = data_size / num_procs;
144
145     if(my_rank == 0)
146     {
147         MatrixData mat_c(data_size);
148         const int start = partition_size * (num_procs - 1);
149         const int end = data_size;
150
151         // calculate matrix section
152         calculate(start, end, dim, mat_a, mat_b, mat_c.begin() + start);
153
154         // only receive if we have data
155         if(partition_size != 0)
156         {
157             // receive from other processes
158             for(int p = 1; p < num_procs; ++p)
159             {
160                 const int offset = partition_size * (p - 1);
161                 receive(MPI_COMM_WORLD, p, mat_c, offset, partition_size);
162             }
163         }
164     }

```

Locals Current Line(s) Current Stack

Name Value

> mat\_a std::vector of length 16, capacity ...

Processes 0-3:

Process stopped at breakpoint in main (matrix\_mult.cpp:138).

Always show this window for user-defined breakpoints

Continue Pause

Input/Output Breakpoints Watchpoints Stacks (All) Tracepoints Tracepoint Output Logbook

Stacks (All)

Processes Function

4 main (matrix\_mult.cpp:138)

Evaluate

Name Value

p <No symbol "p" in current context.>

Control

Tools

Window

Help

## Multi-Dimensional Array Viewer

Process Details

Message Queues

Current Memory Usage

Overall Memory Stats

Cross-Process Comparison

Cross-Thread Comparison

Open Logbook

Disassemble

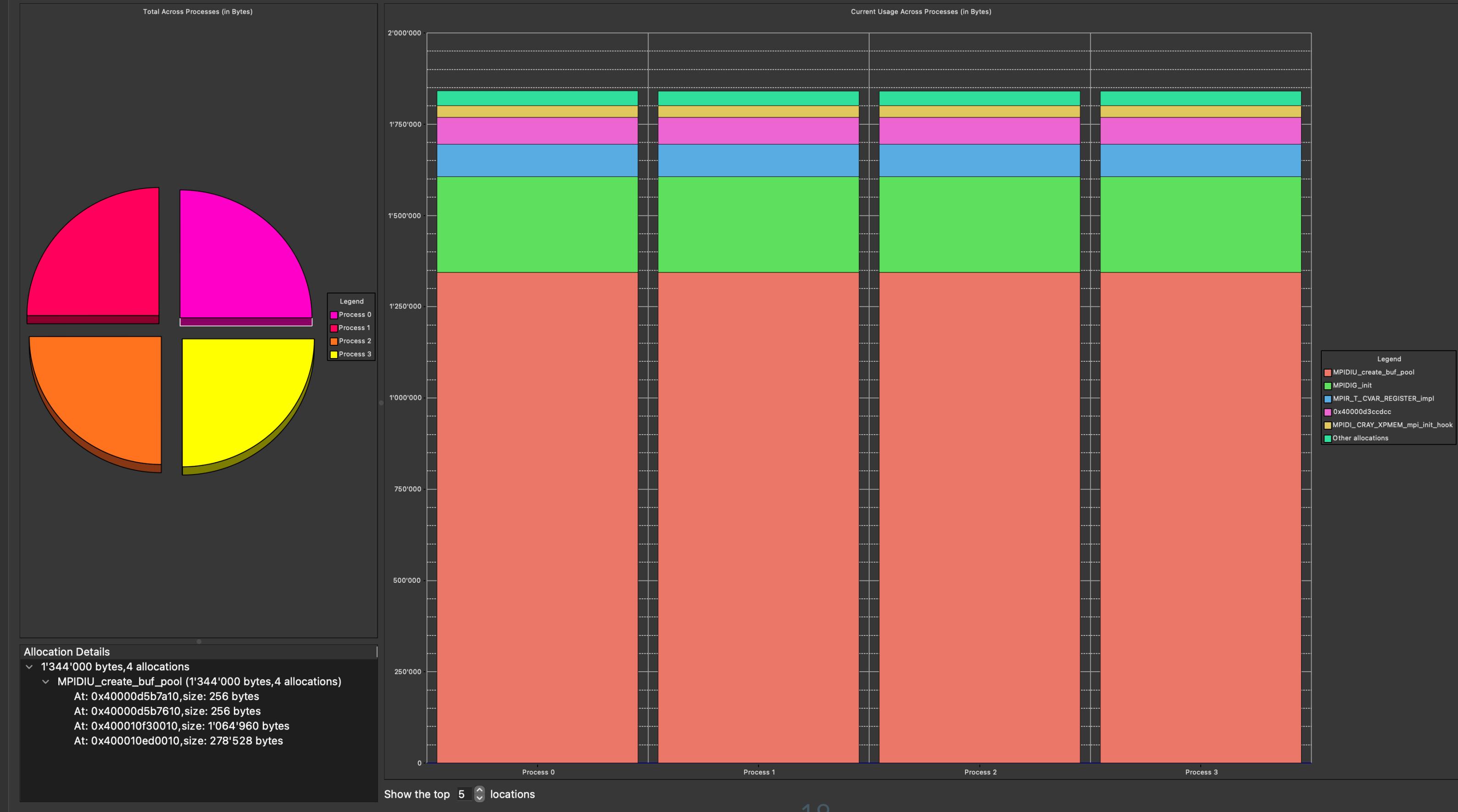
Threads To

Group

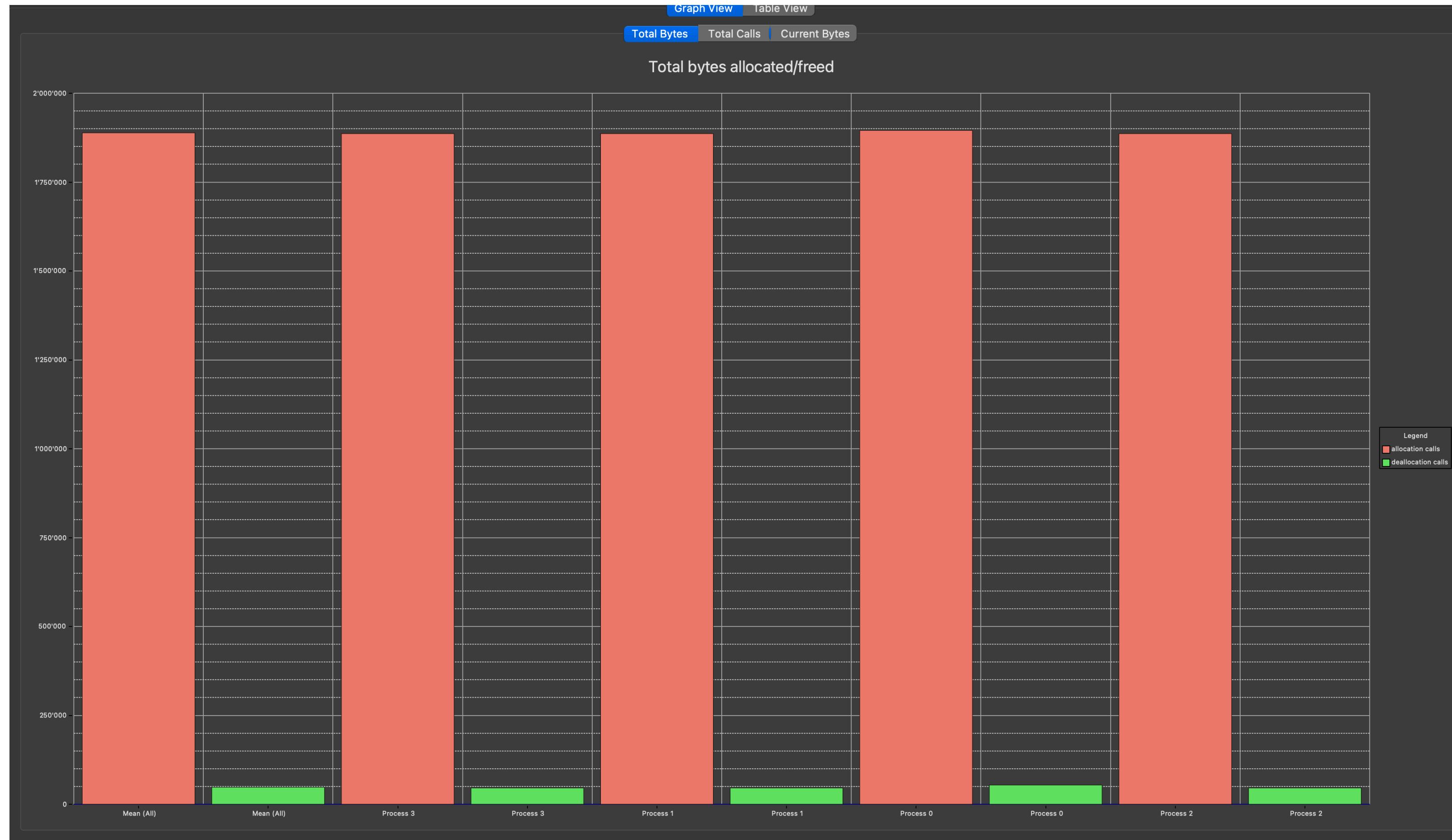
Restrict to the top 4 processes Refresh

Allocations from: Host

Memory Usage Allocation Table



Show the top 5 locations



# MAP

## linaro forge



[Get trial licence](#)

[Support](#)

[linaroforge.com](#)

Licence Serial: 17741 ?

**PROFILE**  
[Profile a program.](#)

**LOAD PROFILE DATA FILE**  
[Load a profile data file from a previous run.](#)

**OPTIONS**

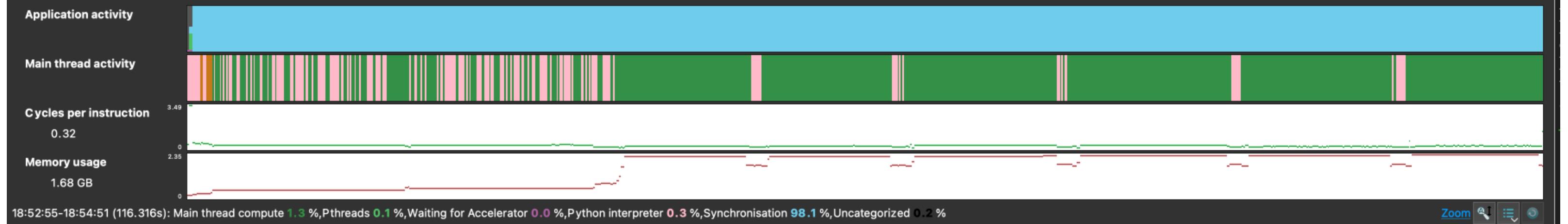
Remote Launch:

todi



**QUIT**

Linaro Forge 24.0.3



run\_filtered\_torus\_grid... nb\_func... nabla4\_unstructured...

```

864     torus_grid.num_vertices,
865     torus_grid.num_edges,
866     torus_grid.num_levels,
867     torus_grid.size[E2C2VDim],
868     grid_cartesian_dimensions[0],
869     grid_cartesian_dimensions[1],
870     halo,
871     repetitions,
872     dry_runs,
873     )
874   )
875
876   print_median_runtimes(runtimes)
877
878   with open(args.output.split(".")[0] + ".json", "w") as file:
879     dump(runtimes, file)
880
881
882 if name == "__main__":
883   run_benchmarks()
884

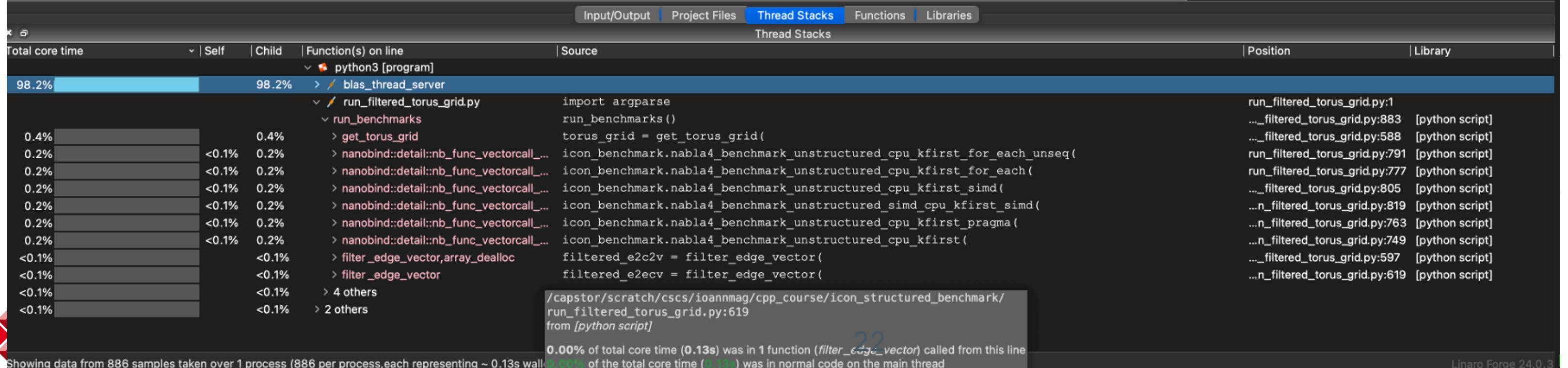
```

1.5%

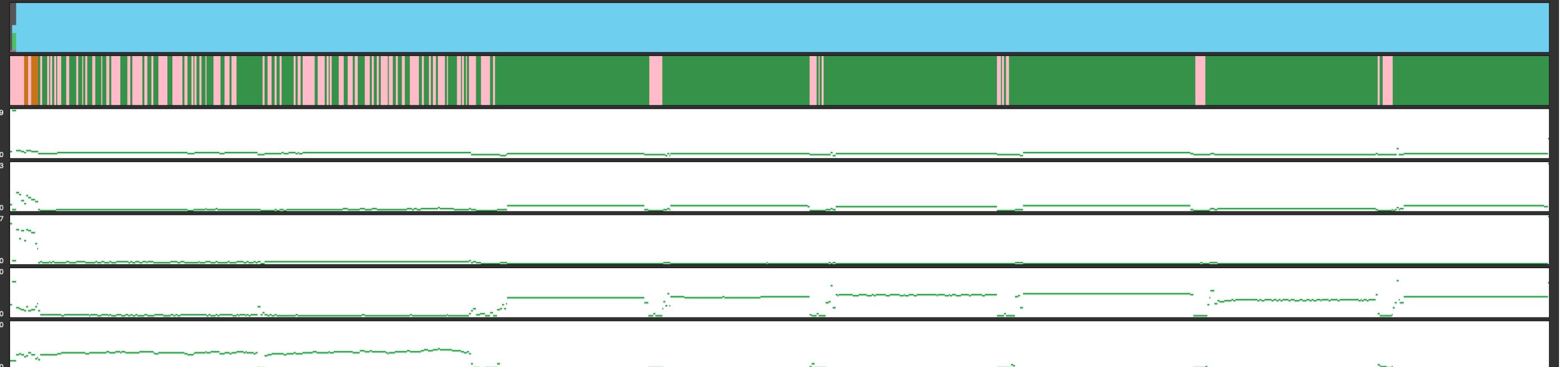
Time spent on line 883

Breakdown of the 1.5% time spent on this line:

- Executing instructions 0.0%
- Calling functions 100.0%



## Application activity



18:52:55-18:54:51 (116.316s): Main thread compute 1.3 %, Pthreads 0.1 %, Waiting for Accelerator 0.0 %, Python interpreter 0.3 %, Synchronisation 98.1 %, Uncategorized 0.2 %

[Zoom](#)

File	Function	Time spent
run_filtered_torus_grid...	nb_func....	0.1%
nb_func....	nabla4_unstructured....	
nabla4_unstructured....	benchmark....	
benchmark....	nabla4_unstructured_simd...	
nabla4_unstructured_simd...		

```

144
145
146
147
148
149 v
150 v
151
152 >
153 >
154 >
155 >
156 >
157 >
    };
}

/// Compute function timed for benchmarking
template <backend_impl I>
void run() {
    if constexpr (I == backend_impl::cpu_kfirst_simd) {
        run_cpu_kfirst_simd();
    } else {
        ...
    };
}

```

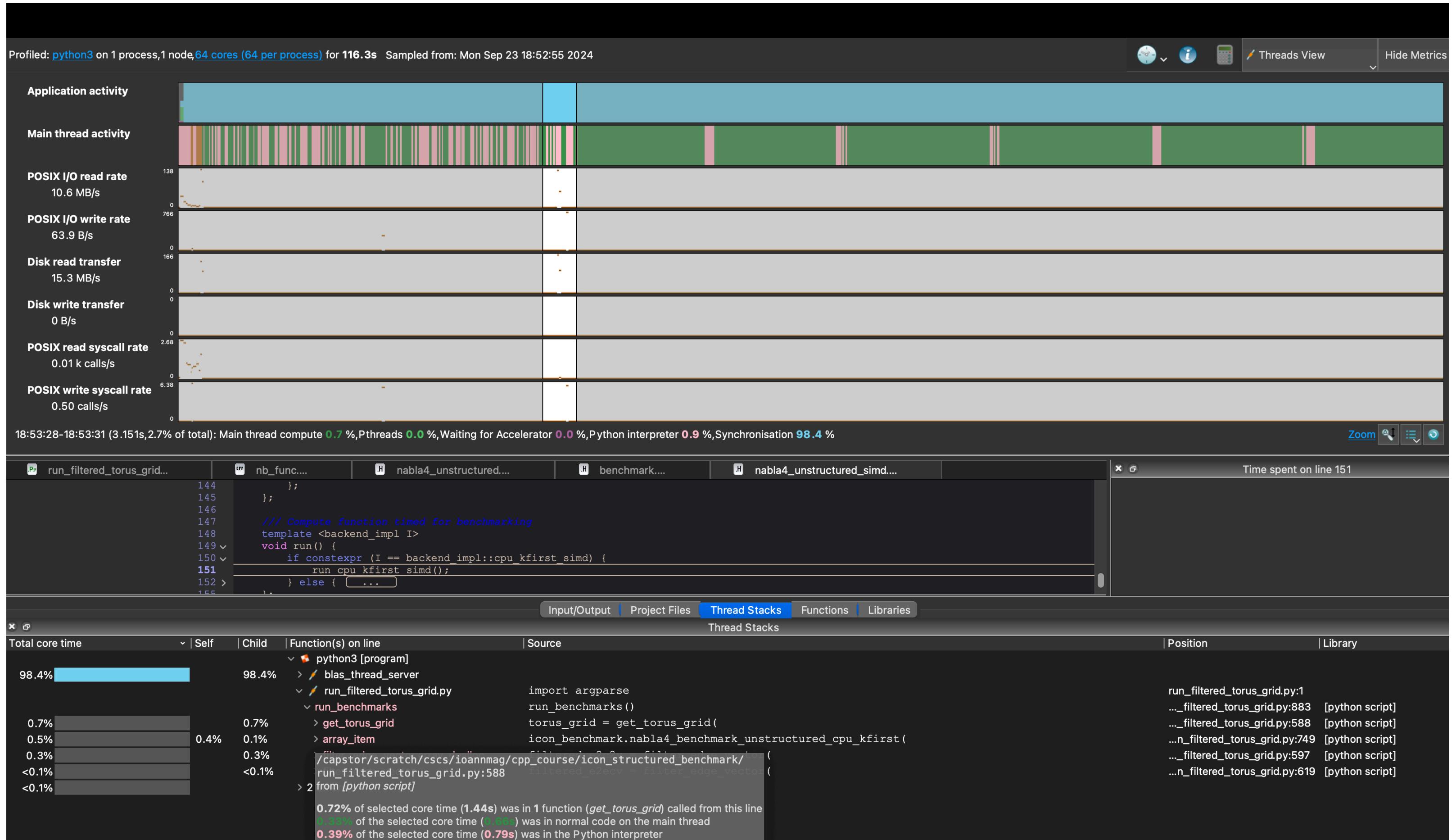
Time spent on line 151

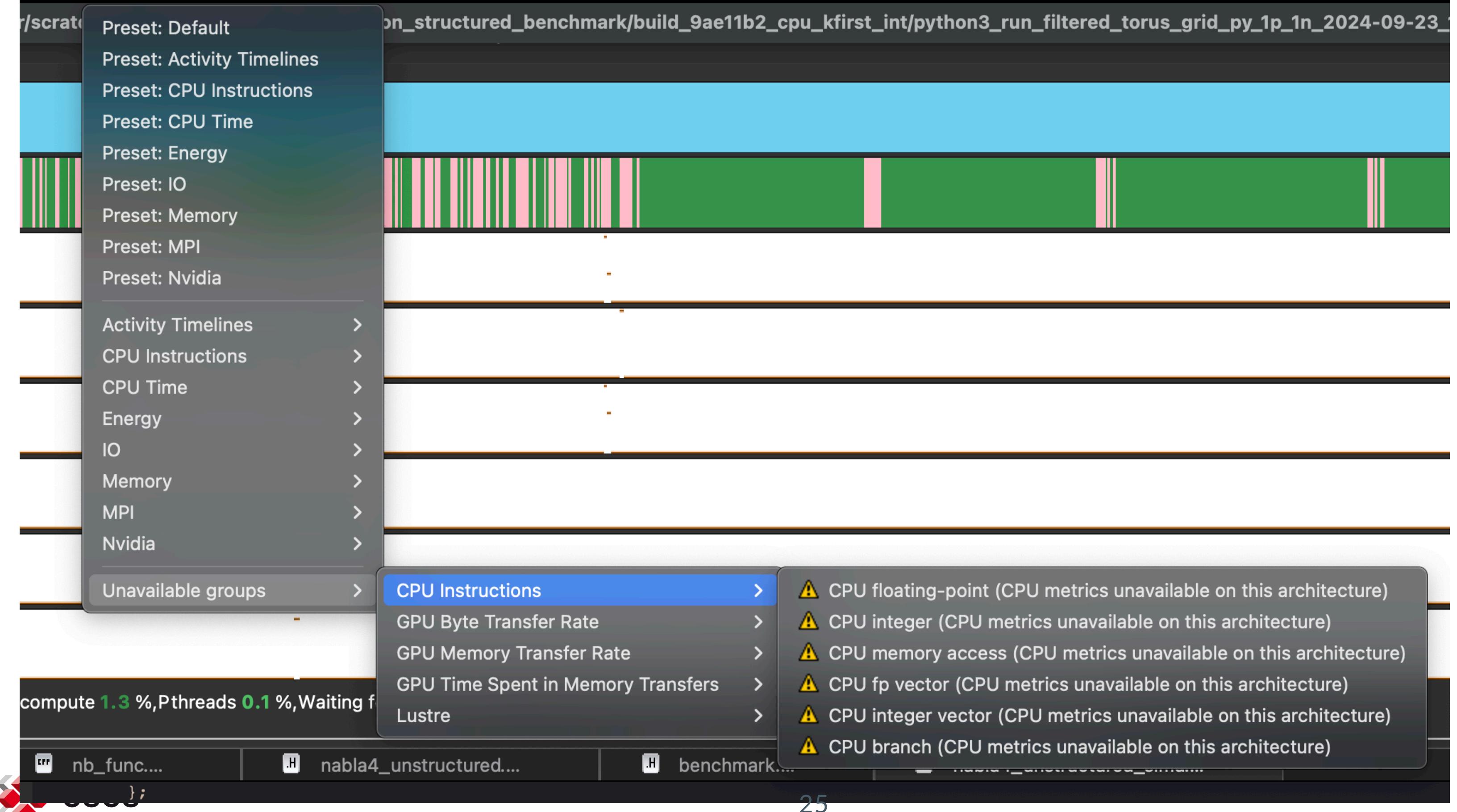
Breakdown of the 0.1% time spent on this line:

Executing instructions 0.0%

Calling functions 100.0%

Total core time	Self	Child	Function(s) on line	Source	Position	Library
0.2%	<0.1%	0.2%	> nanobind::detail::nb_func_vectorcall...	icon_benchmark.nabla4_benchmark_unstructured_cpu_kfirst_for_each()	run_filtered_torus_grid.py:777	[python script]
0.2%	<0.1%	0.2%	> nanobind::detail::nb_func_vectorcall...	icon_benchmark.nabla4_benchmark_unstructured_cpu_kfirst_simd()	..._filtered_torus_grid.py:805	[python script]
			> nanobind::detail::nb_func_vectorcall...	icon_benchmark.nabla4_benchmark_unstructured_simd_cpu_kfirst_simd()	...n_filtered_torus_grid.py:819	[python script]
			> nanobind::detail::func_create<false,t...	result = f->impl((void *) f->capture, args, args_flags,	nb_func.cpp:725	icon_benchmark.cpyth...
			> nanobind::detail::func_create<false,t...	f.impl = [](void *p, PyObject **args, uint8_t *args_flags, rv_policy policy,	nb_func.h:177	icon_benchmark.cpyth...
			> nabla4_benchmark_unstructured...	NB_TYPE_CASTER(List, io_name(NB_TYPING_SEQUENCE, NB_TYPING_LIST) +	nb_list.h:18	icon_benchmark.cpyth...
			> nabla4_benchmark_vector<(ba...	std::make_tuple(e2c2v, e2ecv, CellDim, VertexDim, EdgeDim, KDim, ECVDim), repetitions, dry...	wrapper.cpp:235	icon_benchmark.cpyth...
			> run_benchmark<nabla4_unstr...	return run_benchmark<T<Data::kfirst>, I>(std::forward<decltype(args)>(args), repetitions, ...	wrapper.cpp:37	icon_benchmark.cpyth...
			> nabla4_unstructured_simd<...	benchmark_object.template run<I>();	benchmark.hpp:46	icon_benchmark.cpyth...
			> nabla4_unstructured_simd...	run_cpu_kfirst_simd();	..._unstructured_simd.hpp:151	icon_benchmark.cpyth...
<0.1%	<0.1%		> std::experimental::paralle...	u_vert[E2C2V_1][k_index] * primal_normal_vert_v1_e2ecv[1] +	..._unstructured_simd.hpp:119	icon_benchmark.cpyth...
<0.1%				z_nabla4_e2_wp[edge_index][k_index] =	...unstructured_simd.hpp:125	icon_benchmark.cpyth...
<0.1%			> std::experimental::paralle...	v_vert[E2C2V_0][k_index] * primal_normal_vert_v2_e2ecv[0] +	..._unstructured_simd.hpp:118	icon_benchmark.cpyth...
<0.1%			> std::experimental::paralle...	u_vert[E2C2V_3][k_index] * primal_normal_vert_v1_e2ecv[3] +	...unstructured_simd.hpp:123	icon_benchmark.cpyth...
<0.1%			> std::experimental::paralle...	const auto nabv_norm_wp = u_vert[E2C2V_2][k_index] * primal_normal_vert_v1_e2ecv[2] +	..._unstructured_simd.hpp:121	icon_benchmark.cpyth...
<0.1%			> std::experimental::paralle...	v_vert[E2C2V_2][k_index] * primal_normal_vert_v2_e2ecv[2] +	...unstructured_simd.hpp:122	icon_benchmark.cpyth...
<0.1%			> std::experimental::paralle...	4.0 * ((nabv_norm_wp - 2.0 * z_nabla2_e2e[edge_index][k_index]) * inv_vert_vert_length_sqr +	...unstructured_simd.hpp:126	icon_benchmark.cpyth...





Metrics

Reports

Compiler Remarks

Window

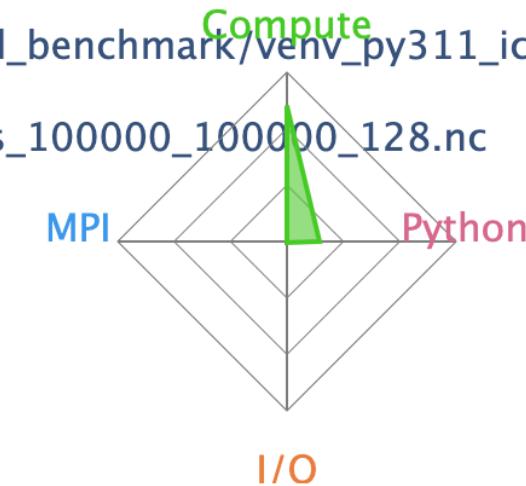
Help

View HTML Performance Report in browser

View Text Performance Report in dialog

Export Performance Report...

Command: /capstor/scratch/cscs/ioannmag/cpp\_course/icon\_structured\_benchmark/venv\_py311\_icon4py/bin/python3  
 .../run\_filtered\_torus\_grid.py  
 /capstor/scratch/cscs/ioannmag/torus\_files\_strassman/torus\_100000\_100000\_128.nc  
 --backend cpu\_kfirst --dry-run  
 Resources: 1 node (288 physical, 288 logical cores per node)  
 4 GPUs per node available  
 Memory: 857 GiB per node  
 Tasks: 1 process  
 Machine: nid005037  
 Architecture: aarch64  
 CPU Family: neoverse-v2  
 Start time: Mon Sep 23 18:52:55 2024  
 Total time: 117 seconds (about 2 minutes)  
 Full path: .



## Summary: run\_filtered\_torus\_grid.py is **Compute-bound** in this configuration

Compute	80.0%	93.1s	<div style="width: 80%; background-color: #2e7131;"></div>	Time spent running application code. High values are usually good. This is <b>high</b> ; check the CPU and accelerators sections for advice
MPI	0.0%	0.0s	<div style="width: 0%; background-color: #3498db;"></div>	Time spent in MPI calls. High values are usually bad. This is <b>very low</b> ; this code may benefit from a higher process count
I/O	0.7%	0.8s	<div style="width: 7%; background-color: #e67e22;"></div>	Time spent in filesystem I/O. High values are usually bad. This is <b>very low</b> ; however single-process I/O may cause MPI wait times
Python Interpreter	19.3%	22.4s	<div style="width: 19.3%; background-color: #e74c3c;"></div>	Time spent in the Python interpreter. Calling out to precompiled libraries may be more efficient. This is <b>very low</b> ; focus on improving MPI or I/O performance first

This application run was **Compute-bound** (based on main thread activity). A breakdown of this time and advice for investigating further is in the [CPU Metrics](#) and [Accelerators](#) sections below.

## CPU Metrics

Linux perf event metrics:

Cycles per instruction	0.32	
L2D cache miss ratio	1.03	
Stalled backend cycles	28.3%	
Stalled frontend cycles	11.2%	

**Cycles per instruction** is low, which is good. Vectorization allows multiple instructions per clock cycle.

## I/O

A breakdown of the 0.7% (0.8s) I/O time:

Time in reads	50.0%	0.4s	
Time in writes	50.0%	0.4s	
Effective process read rate	256 MB/s		
Effective process write rate	22.3 kB/s		

Most of the time is spent in **write operations** with a very low effective transfer rate. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected.

## Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	1.56 GiB	
Peak process memory usage	2.19 GiB	
Peak node memory usage	2.0%	

The peak node memory usage is very low. Larger problem sets

## MPI

A breakdown of the 0.0% (0.0s) MPI time:

Time in collective calls	0.0%	0.0s	
Time in point-to-point calls	0.0%	0.0s	
Effective process collective rate	0.00	bytes/s	
Effective process point-to-point rate	0.00	bytes/s	

No time is spent in **MPI** operations. There's nothing to optimize here!

## Threads

A breakdown of how multiple threads were used:

Computation	<0.1%	9.6s	
Synchronization	99.9%	11622.0s	
Physical core utilization	22.2%		
System load	0.5%		

Significant time is spent **synchronizing threads**. Check which locks cause the most overhead with a profiler.

This may be a sign of overly fine-grained parallelism or of workload imbalance between threads.

## Accelerators

A breakdown of how CUDA accelerators were used:

GPU utilization	0.0%	
Mean GPU memory usage	0.8%	
Peak GPU memory usage	0.8%	

GPUs are available but are not used. Identify suitable hot loops