Durham University

A brief introduction to profiling with the Intel toolchain

Mark Turner and Mark Dixon
Advanced Research Computing

July 7th, 2022

Welcome and icebreaker

Thank you for joining!

- Meet the team

- 1 hour course (12pm-1pm)
- Please do post in the chat or speak up when questions arise

- Feel free to post something in the chat about where in the university you come from and something about your research interests

Durham University

Course structure

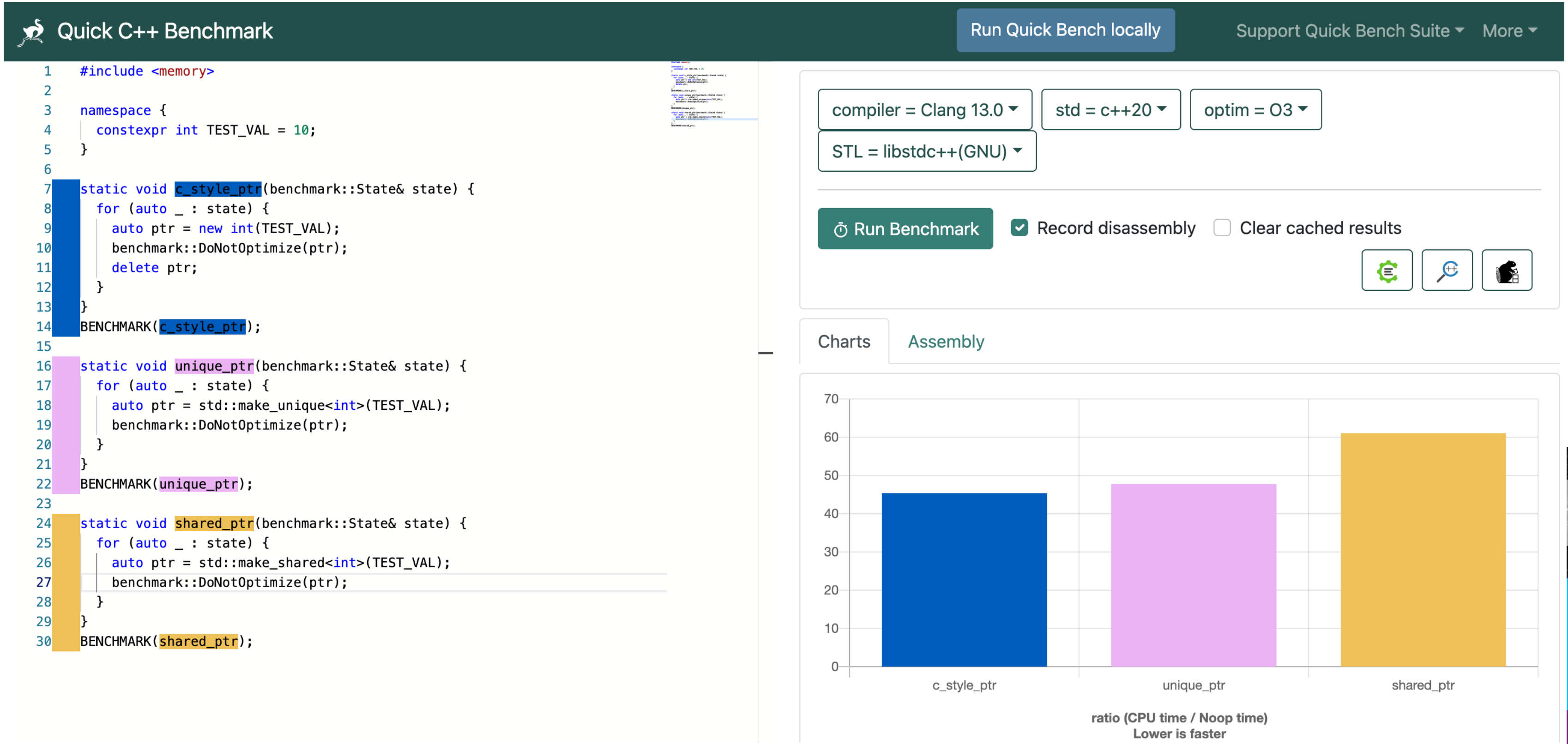
- What is performance analysis?
- Automatic tracing of MPI codes using ITAC
- Leveraging the Intel Traceanalyzer
- User defined instrumentation using ITAC

- An exercise…

What is performance analysis?

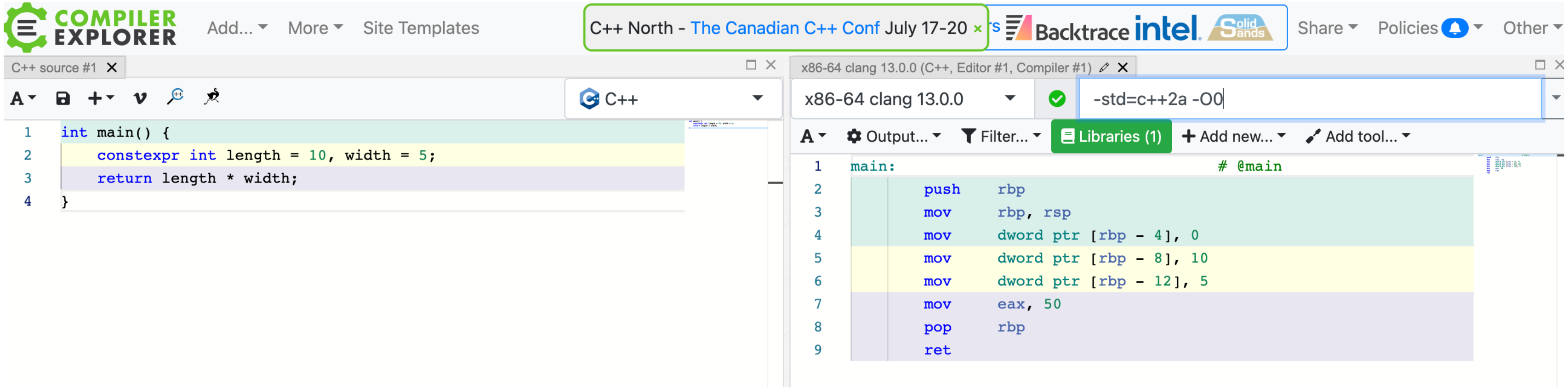
Strongly depends on what you are trying to investigate / achieve

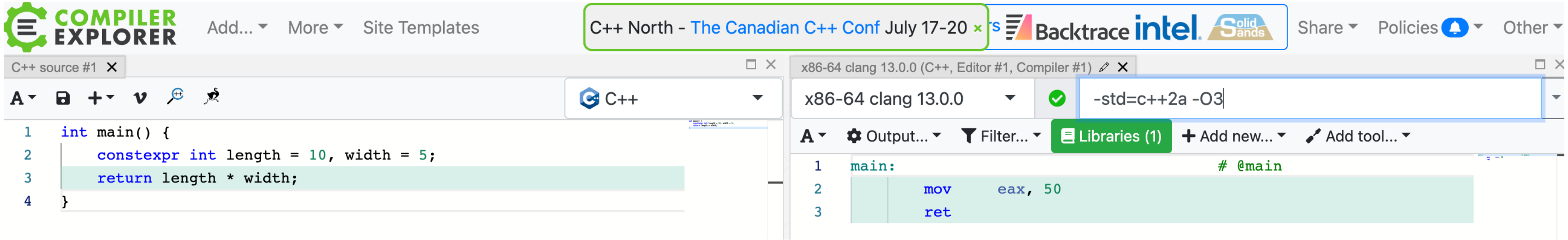
quick-bench.com



Durham University

godbolt.org


COMPILER EXPLORER
Add... More Site Templates
C++ North - The Canadian C++ Conf July 17-20
Share Policies Other
C++ source #1
C++
int main() {
    constexpr int length = 10, width = 5;
    return length * width;
}
x86-64 clang 13.0.0 (C++, Editor #1, Compiler #1)
x86-64 clang 13.0.0
-std=c++2a -O0
Output... Filter... Libraries (1) Add new... Add tool...
main: # @main
        push   rbp
        mov   rbp, rsp
        mov   dword ptr [rbp - 4], 0
        mov   dword ptr [rbp - 8], 10
        mov   dword ptr [rbp - 12], 5
        mov   eax, 50
        pop   rbp
        ret


COMPILER EXPLORER
Add... More Site Templates
C++ North - The Canadian C++ Conf July 17-20
Share Policies Other
C++ source #1
C++
int main() {
    constexpr int length = 10, width = 5;
    return length * width;
}
x86-64 clang 13.0.0 (C++, Editor #1, Compiler #1)
x86-64 clang 13.0.0
-std=c++2a -O3
Output... Filter... Libraries (1) Add new... Add tool...
main: # @main
        mov   eax, 50
        ret

Measuring like this is great, but…

how do we study performance of entire codes – in particular, those that run across supercomputers?

What if we want to understand the behaviour of an entire simulation?

Durham University

arc

A flavour of performance analysis in HPC

- Identifying code flaws:
  - Hotspots (always a good place to focus dev time)
  - Load imbalance
  - Inefficient communication patterns
  - Poor scaling
  - Inefficient use of hardware (cache misses, excessive I/O etc.)

- Establishing the obstacles to being **compute bound**
  - Memory bound; MPI bound

The aim!

How can tools understand our codes?

- **Sampling**
  - Regular reading of typical averages, patterns etc.
  - Recording runtime characteristics (e.g., hardware counters)
  - `likwid-perfctr`

- **Tracing**
  - Chronologically ordered events/timelines
  - Logging **events** (e.g. function calls) in code

For today, we're interested in tracing

Durham University

arc

Intel Trace Analyzer and Collector

- A graphical tool for profiling and understanding MPI behaviour
- Presents the chronology of your runtime in a visual way
- Easier to reason about behaviour of parallel codes and identify:
  - temporal dependencies and bottlenecks
  - load imbalance
  - Hotspots
  - And more…

So, on the surface, what does ITAC give us?

A high-level overview page:

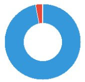
Summary: peano4.stf

Total time: **1.07e+04** sec. Resources: **2** processes, **4** threads, **1** node.

Continue >

Ratio

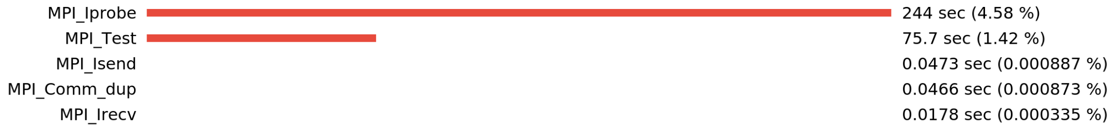
This section represents a ratio of all MPI calls to the rest of your code in the application.



Serial Code - 1.03e+04 sec	96.9 %
OpenMP - 0 sec	0 %
MPI calls - 320 sec	3 %

Top MPI functions

This section lists the most active MPI functions from all MPI calls in the application.



Where to start with analysis

For deep analysis of the MPI-bound application click "Continue >" to open the tracefile View and leverage the **Intel® Trace Analyzer** functionality:

- *Performance Assistant* - to identify possible performance problems
- *Imbalance Diagram* - for detailed imbalance overview
- *Tagging/Filtering* - for thorough customizable analysis

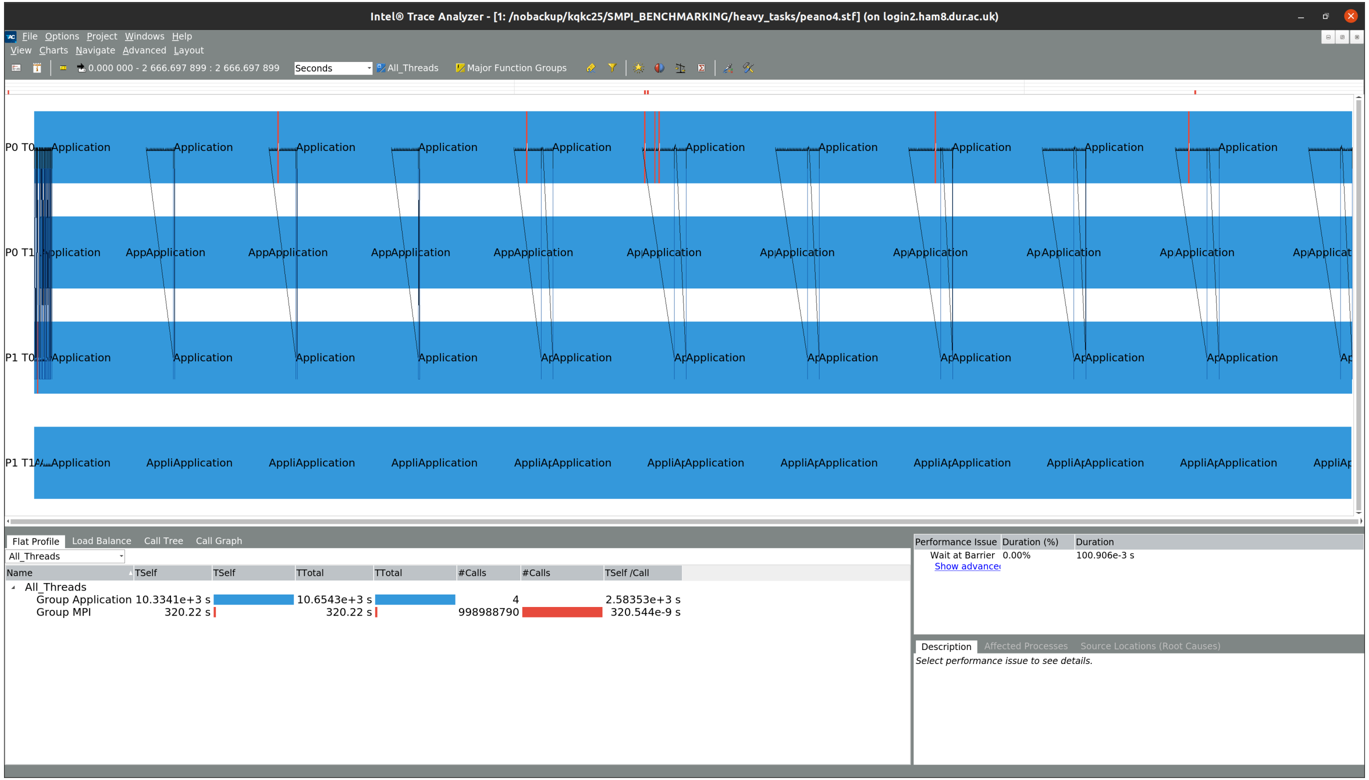
To optimize node-level performance use:
Intel® VTune™ Profiler for:
- algorithmic level tuning with hpc-performance and threading efficiency analysis;
- microarchitecture level tuning with general exploration and bandwidth analysis;

Intel® Advisor for:
- vectorization optimization and thread prototyping.

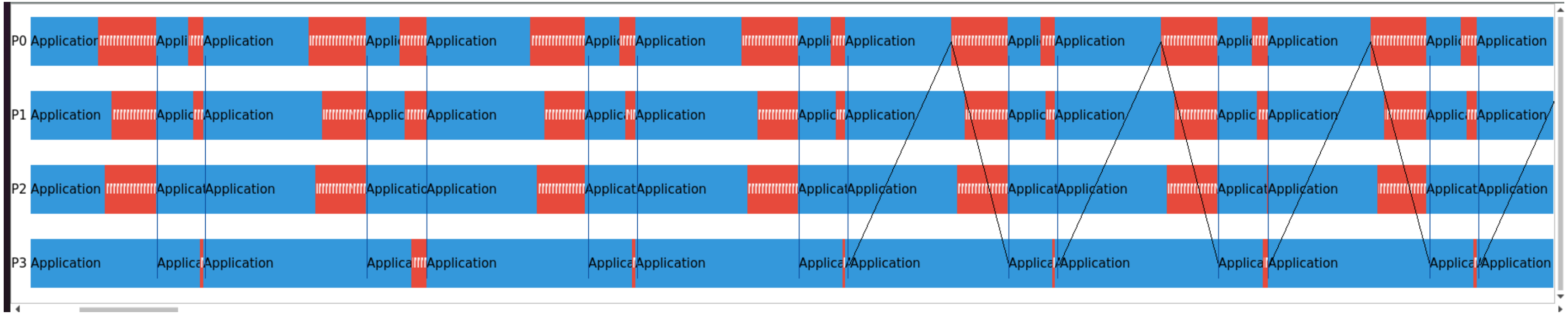
For more information, see documentation for the respective tool:
Analyzing MPI applications with Intel® VTune™ Profiler
Analyzing MPI applications with Intel® Advisor

Durham University

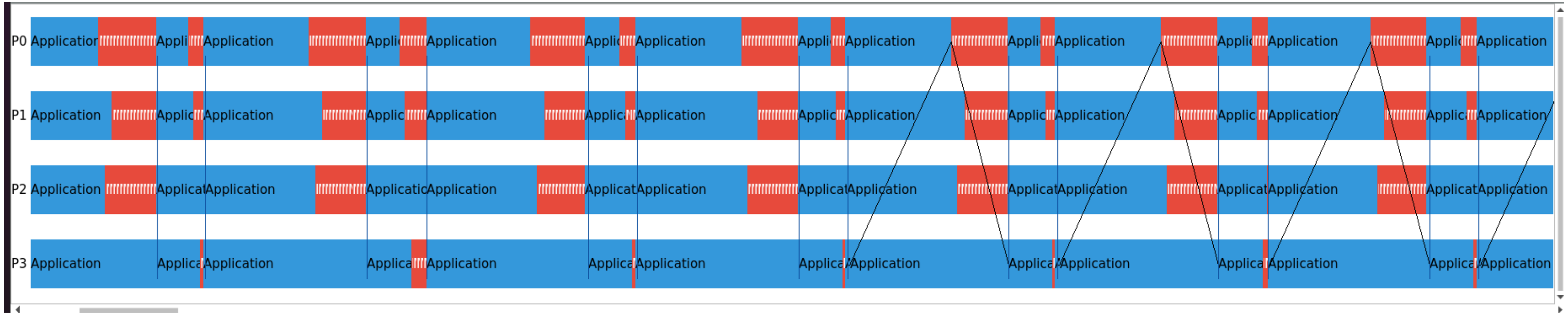
An event timeline:

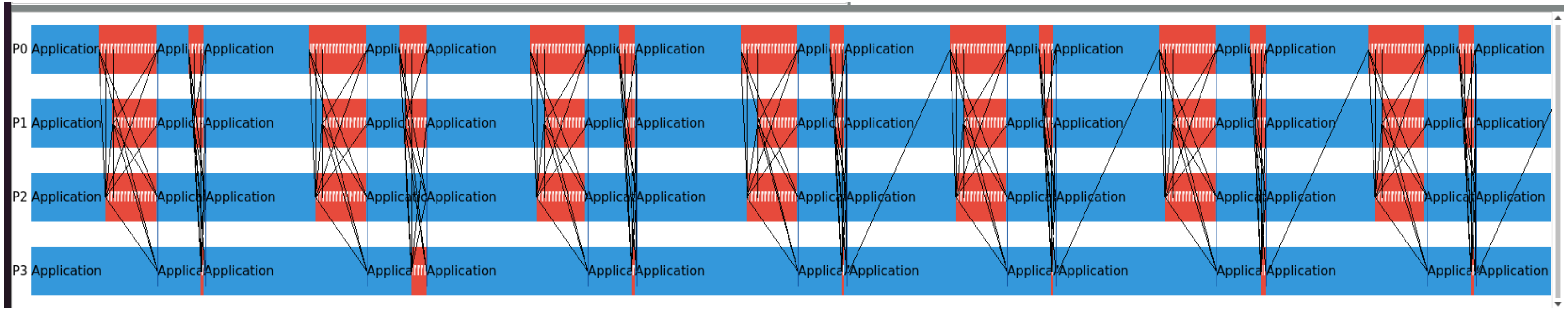


A more problematic snapshot:



Ability to filter MPI communications:





A demo

I will show how to navigate the traceanalyzer and explore an event timeline

And how can I generate these profiles for my code on Hamilton?

Generating a trace on Hamilton 8

- Compile with Intel MPI (Open MPI is not supported…)
- Load the ITAC module:
  - `module load itac`
- The trace files can be BIG. It is worth Asking SLURM for more temporary disk space on your nodes:
  - #SBATCH --gres=tmp:400G
- Then pass the `-trace` flag through to mpirun:
  - `mpirun -trace -np <N> <executable>`

Durham University

Exercise

Ask us questions!
We’re very happy to setup breakout rooms upon request

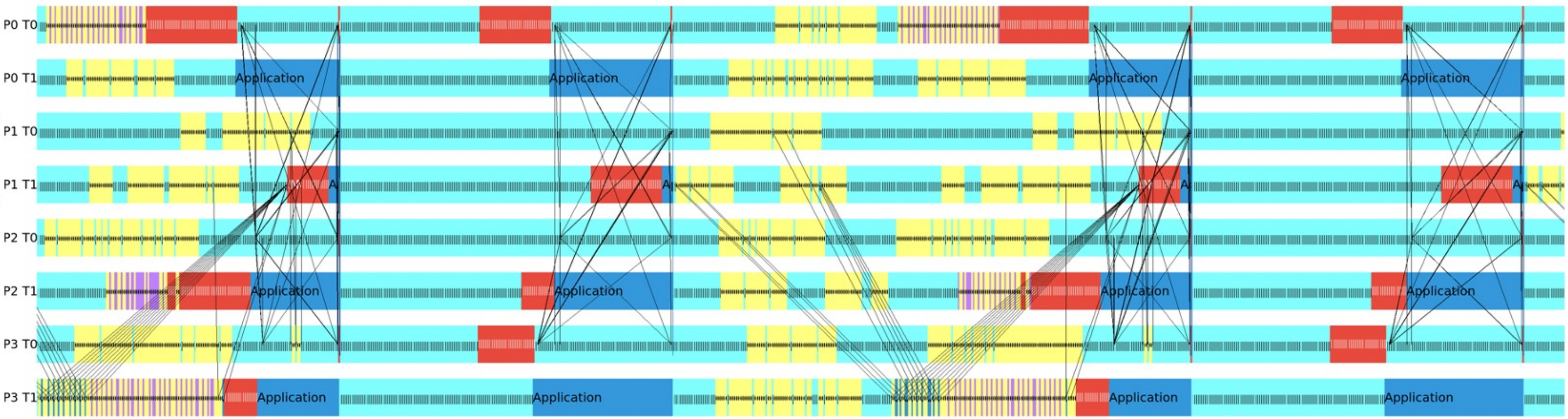
Durham University

So far, we have the tools to better understand MPI behaviour

… But how can we visualize *application* behaviour?

Durham University

User defined instrumentation:



- Red = MPI
- Light Blue = AMR framework
- Yellow = Physics
- Dark blue = Open MP

Zoom in to see more detail. Right click a cell and select “ungroup” to view function call.

Durham University

And how can I generate these profiles for my code on Hamilton?

Generating a trace on Hamilton 8

1. Add instrumentation boilerplate

```
namespace {
  std::map<std::string, const int> itac_handles;
}

void trace_in(const std::string& trace) {
  if (itac_handles.count(trace) == 0) {
    int new_handle;
    VT_funcdef( trace.c_str() , VT_NOCLASS, &new_handle );
    itac_handles.insert({trace, new_handle});
  }

  VT_begin(itac_handles[trace]);
}

void trace_out(const std::string& trace) {
  VT_end(itac_handles[trace]);
}
```

Generating a trace on Hamilton 8

1. Add instrumentation boilerplate
2. Add your code (whatever that might be ☺)

```
void do_fancy_physics(const int rank) {
  float aggregator = 0;
  for (int i = 0; i < 100'000'000; ++i) {
    aggregator += sin(i);
  }
  std::cout << "Result on " << rank << ": " << aggregator << std::endl;
}
```

Generating a trace on Hamilton 8

1. Add instrumentation boilerplate
2. Add your code (whatever that might be ☺)
3. Add tracing around your code

```
int main(int argc, char* argv[]) {
  MPI_Init(&argc, &argv);
  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  const std::array<std::string, 3> phases {"alpha", "beta", "gamma"};
  for (const auto &phase : phases) {
    trace_in(phase);
    do_fancy_physics(rank);
    trace_out(phase);
  }

  MPI_Finalize();
  return 0;
}
```

Durham University

arc

Building/executing the code

1. Load modules:
   - `export FLAVOUR_NOCONFLICT=1`
   - `module load gcc oneapi intelmpi itac`

- Launch the build:
  - `mpiicpc -I/apps/developers/tools/itac/2021.4/1/default/itac/2021.4.0/include -L/apps/developers/tools/itac/2021.4/1/default/itac/2021.4.0/slib -lVTnull itac_test.cpp -o itac_test`

- Run with ITAC tracing enabled:
  - `mpirun -trace -np 2 ./itac_test`

Ungrouped output:

- `traceanalyzer itac_test.stf`

Durham University

Code for reference:

```
#include <VT.h>
#include <cmath>
#include <iostream>
#include <map>
#include <mpi.h>
#include <string>

namespace {
  std::map<std::string, const int> itac_handles;
}

void trace_in(const std::string &trace) {
  if (itac_handles.count(trace) == 0) {
    int new_handle;
    VT_funcdef(trace.c_str(), VT_NOCLASS, &new_handle);
    itac_handles.insert({trace, new_handle});
  }

  VT_begin(itac_handles[trace]);
}

void trace_out(const std::string &trace) {
  VT_end(itac_handles[trace]);
}

void do_fancy_physics(const int rank) {
  float aggregator = 0;
  for (int i = 0; i < 100'000'000; ++i) {
    aggregator += sin(i);
  }
  std::cout << "Result on " << rank << ": " << aggregator << std::endl;
}

int main(int argc, char *argv[]) {
  MPI_Init(&argc, &argv);
  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  const std::array<std::string, 3> phases{"alpha", "beta", "gamma"};
  for (const auto &phase : phases) {
    trace_in(phase);
    do_fancy_physics(rank);
    trace_out(phase);
  }

  MPI_Finalize();
  return 0;
}
```

Durham University

Durham University

Thank you!

- Feedback would really be appreciated: **https://bit.ly/arc_trainingfeedback**
- Other training courses at ARC
- RSE support
- Join the Society of Research Software Engineering: https://society-rse.org/join-us/