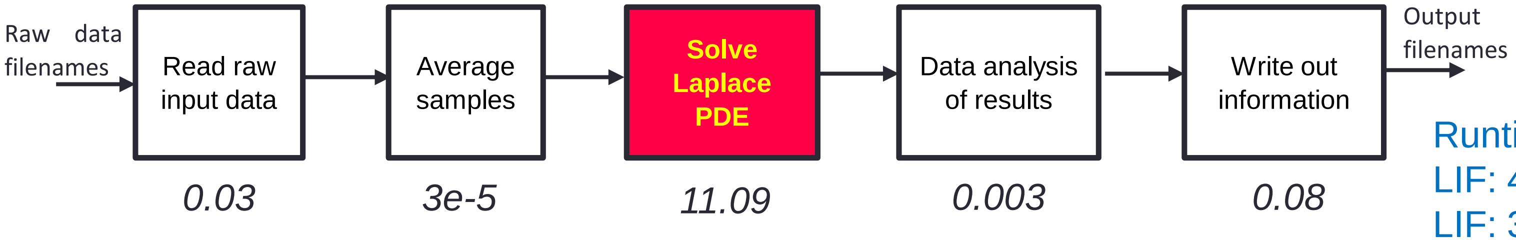
Practical three

Wrap up

Load imbalance factor

- Sample solutions are available
  - MPI P2P messages for communication between stages
  - For the termination poisoned pill an empty (NULL) message can be sent



Raw data filenames

Read raw input data

Average samples

Solve Laplace PDE

Data analysis of results

Write out information

Output filenames

0.03

3e-5

11.09

0.003

0.08

Runtime: 11.09s
LIF: 4.95 (mean)
LIF: 366 (median)

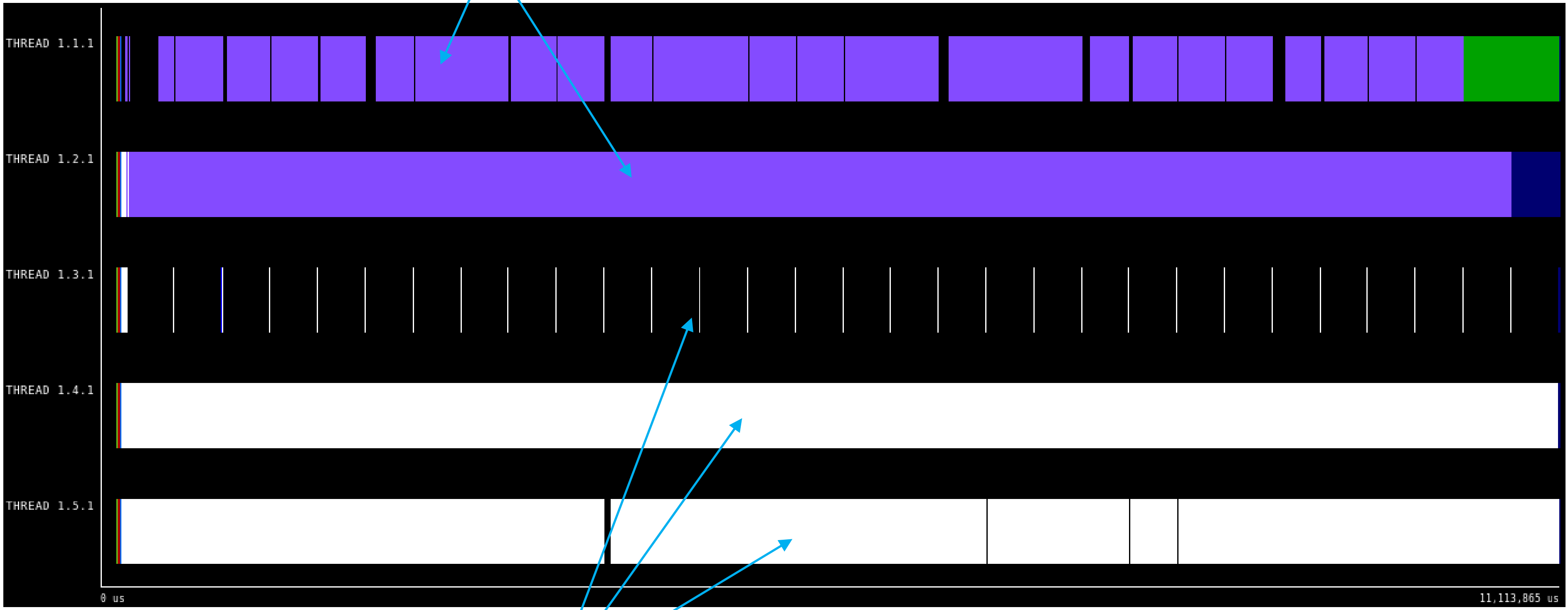
- But the stages of the pipeline are heavily imbalanced
  - Fact that we have one very large number and the rest small numbers impacts how we define the term *average*
  - Not necessarily easy to give lightly loaded stages more work, but can do something to optimise the heavily loaded stage(s)

Load Imbalance Factor (LIF):
maximum load / average load

Where 1 is ideal

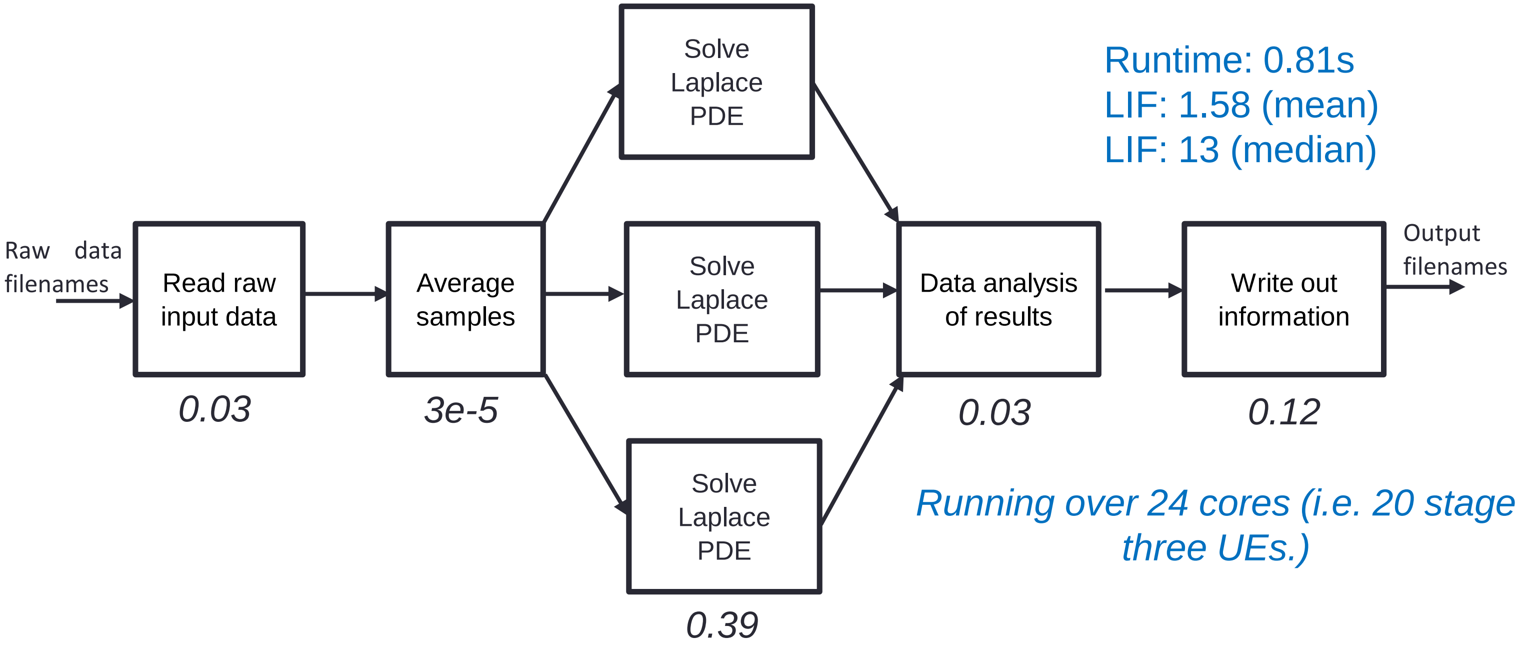
Let’s look at this with Paraver

MPI_Send



MPI_Recv

Duplicating the third stage

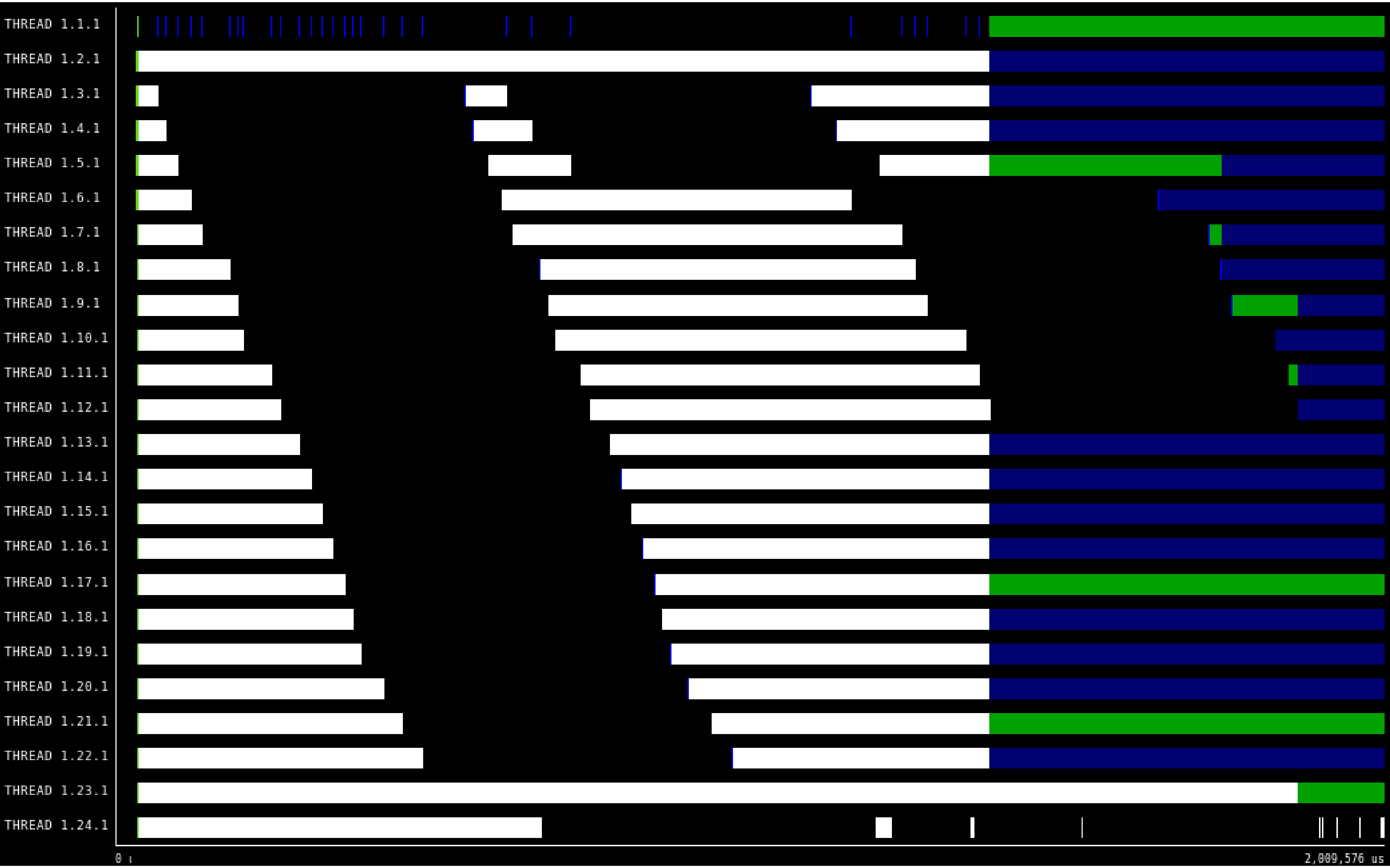


Runtime: 0.81s
LIF: 1.58 (mean)
LIF: 13 (median)

Running over 24 cores (i.e. 20 stage three UEs.)

- All extra UEs make up duplicate stage three.
  - No stage three UEs communicate, but instead work concurrently on different pieces of data
- Fairly simple to do, but termination does require a little more thought

In Paraver


THREAD 1.1.1
THREAD 1.2.1
THREAD 1.3.1
THREAD 1.4.1
THREAD 1.5.1
THREAD 1.6.1
THREAD 1.7.1
THREAD 1.8.1
THREAD 1.9.1
THREAD 1.10.1
THREAD 1.11.1
THREAD 1.12.1
THREAD 1.13.1
THREAD 1.14.1
THREAD 1.15.1
THREAD 1.16.1
THREAD 1.17.1
THREAD 1.18.1
THREAD 1.19.1
THREAD 1.20.1
THREAD 1.21.1
THREAD 1.22.1
THREAD 1.23.1
THREAD 1.24.1
0 us
2,009,576 us

A note on MPI P2P communication calls

- Blocking calls
  - MPI_Send, MPI_Ssend, MPI_Bsend, MPI_Rsend
  - Will only return from the call when the communication has *completed*
    - The data buffer can be reused

- Non-blocking calls
  - MPI_Isend, MPI_Issend, MPI_Ibsend, MPI_Irsend
  - Does not wait until the communication completes, instead returns pretty much immediately and need to check request handle to track completion
    - Via MPI_Wait, MPI_Test, MPI_Cancel etc.…

Irrespective, can not reuse data buffer until P2P communication call has completed!

But what does “completed” mean?

- Depends on whether it’s a send, ssend, bsend, rsend

- Standard send (MPI_Send)
  - Completes either when the message has been copied to an internal buffer or started to be received by the receiver
- Buffered send (MPI_Bsend)
  - Completes when the message has been copied to an internal buffer
- Synchronous send (MPI_Ssend)
  - Completes when the message has started to be received by the receiver
- Ready send (MPI_Rsend)
  - Same as a standard send, but the receiver must have already posted a corresponding receive call, otherwise is erroneous
    - Suggest not so useful nowadays, but could improve performance historically

My suggestions

- Realise that blocking vs non-blocking is entirely separate than the semantics of completion
  - For MPI send calls, it’s simpler for receives!

- Lots of people find this a difficult part of the standard to understand, so when looking at a user’s parallel code can be a source of bugs/performance issues

- If you have lots of synchronous sends (MPI_Ssend) in your code, then do you really need them?
  - If you replace with standard send will this maintain correctness, as by doing so could give you a performance boost