Parallel design patterns ARCHER course

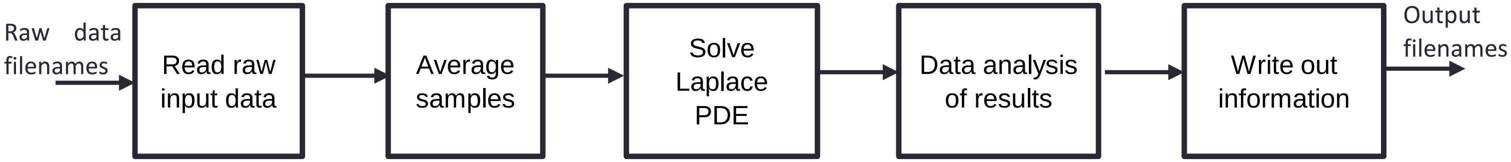
Practical two: Pipeline for pollution problem

Reusing this material

Extending the problem

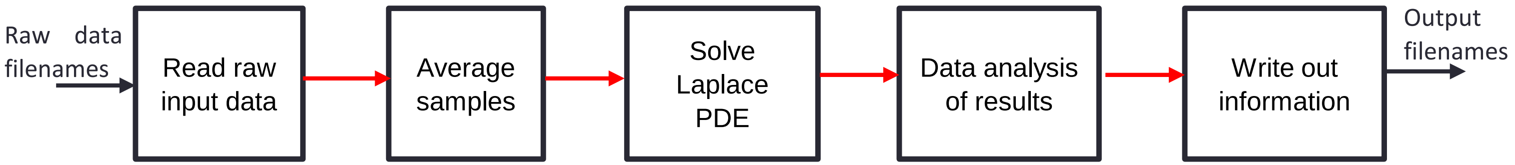
- The geologists wish to use the pollution calculation code in a more automated, high volume, approach.

- Take some raw measured data and feed this into the calculation code, then generating some final result answer which tells them where about in the pipe the pollution is above a specific threshold and the severity of the pollution.
  - Input, raw, data provided in a directory of files – one for each pipe we are testing



Your task…….

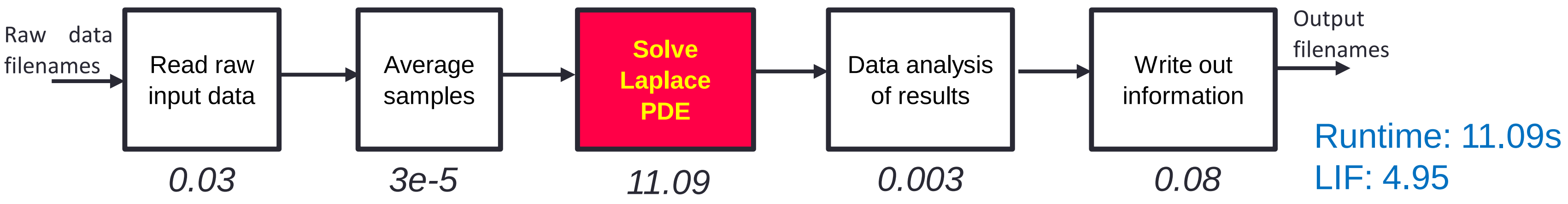
- You are supplied with the logic of each stage, but these are currently unconnected
  - Complete the code so each stage runs on a UE and communicates with neighbouring stages
  - You also need to consider termination via a poisoned pill



- Once you have done this you will calculate the load imbalance
  - And as an advanced exercise look at addressing this

Wash up of practical

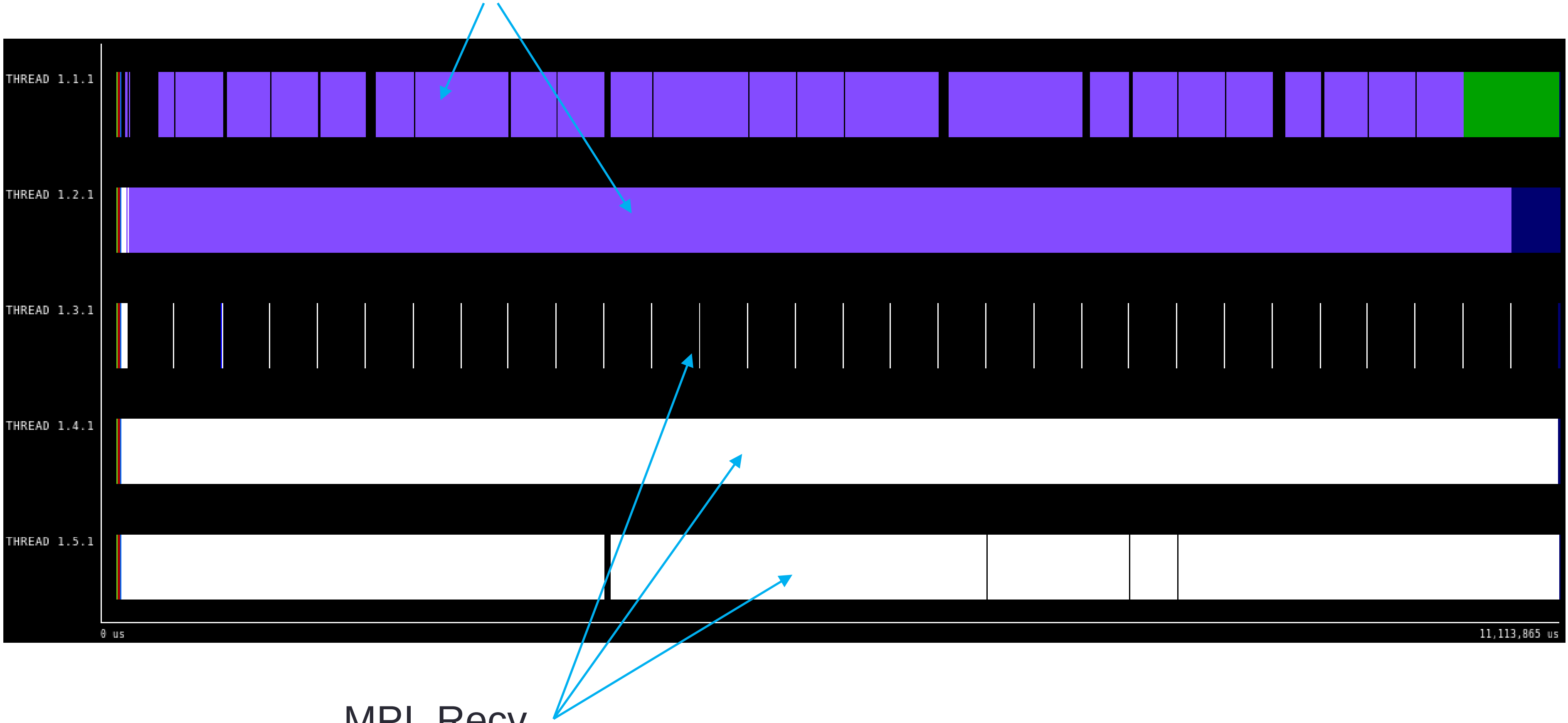
- Sample solutions are available
  - MPI P2P messages correspond well to communication between stages
  - For the termination poisoned pill an empty (NULL) message can be sent



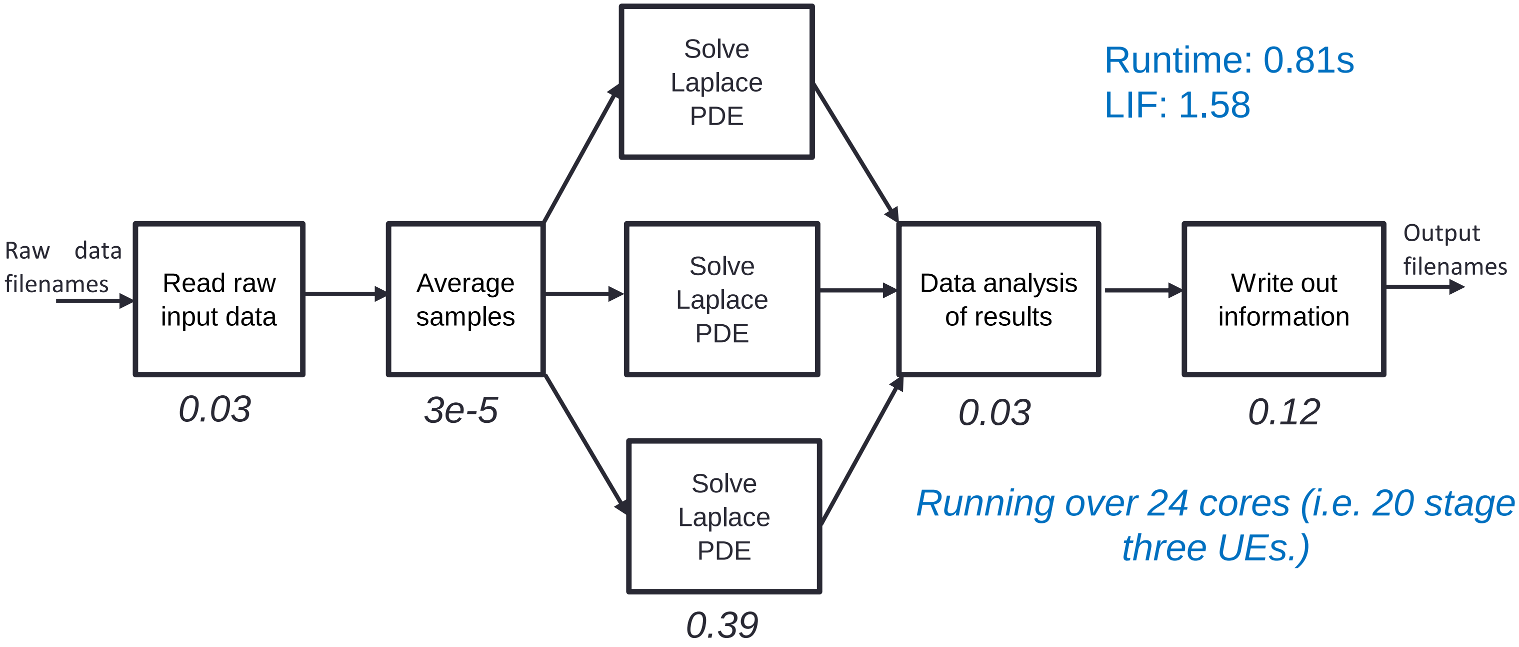
Runtime: 11.09s
LIF: 4.95

- But the stages of the pipeline are heavily imbalanced
  - Figures reported here are for a pipe 128 high, 1024 long
  - Not necessarily easy to give lightly loaded stages more work, but can do something to optimise the heavily loaded stage(s)

Let’s look at this with Paraver



Duplicating the third stage

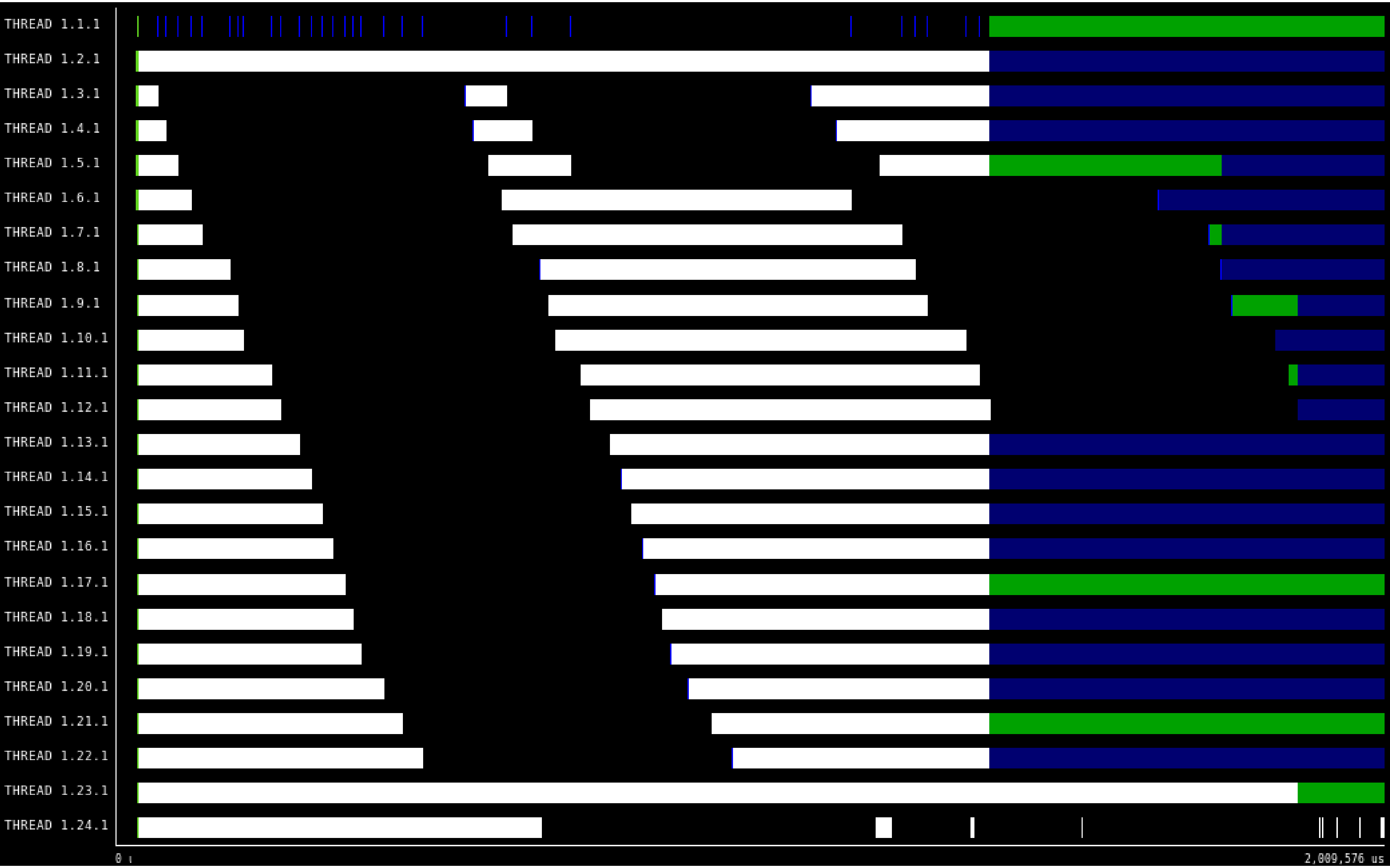


Runtime: 0.81s
LIF: 1.58

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
2,009,576 us

With the LIF what's average?

- *LIF = maximum load / average load*
- This tells us how much faster the code could run if the load were perfectly balanced (1.0 being the best.)

- Assume we take the mean (i.e. sum up all values and divide by the number of UEs)
  - But in extreme cases, where we have small amounts of load and one very large value then this can be misleading as the large value pollutes things.
  - Instead the median can sometimes be a better approach

Code	Runtime	Mean LIF	Median LIF
Linear	11.09s	4.95	366
Multiple stage three	0.81s	1.58	13