Fractal Practical

Investigating task farms and load imbalance

Partners

Ian Harrow Consulting

Funding

Horizon 2020
European Union Funding
for Research & Innovation

Reusing this material

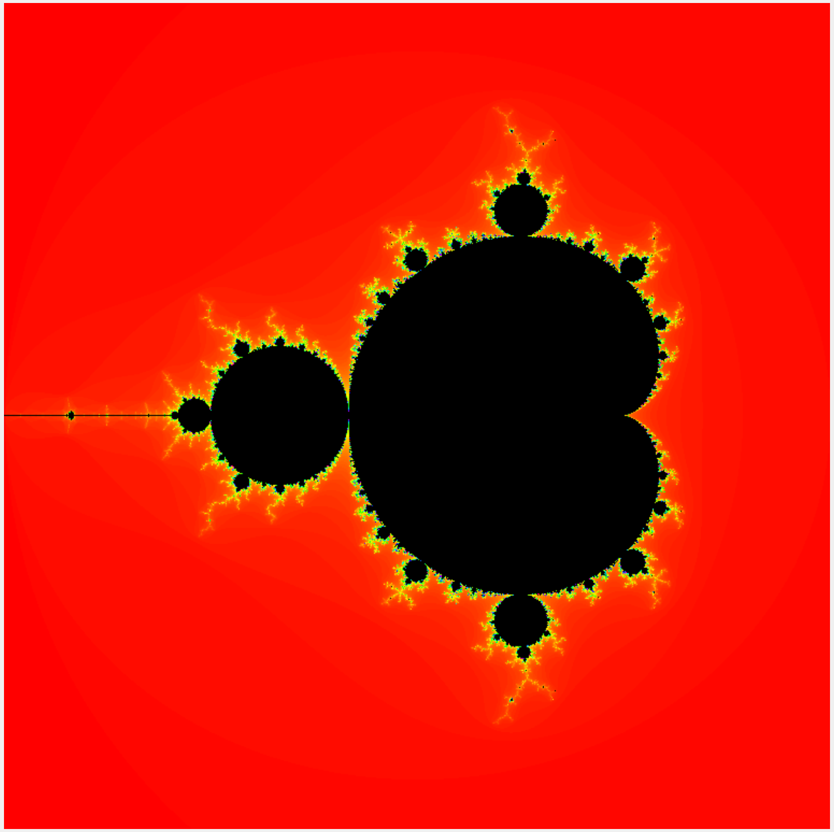
Mandelbrot Set

- Mandelbrot Set can be thought of as the set of points with 2D coordinates (x,y) that satisfy a particular property
  - not important for the exercise what this property actually is
- Supplied code works out in parallel as a task farm for a grid of points whether each point belongs to the Set or not
- For each point a calculation is repeated iteratively, with the result of each iteration serving as input to the next
  - continues until the point is proven *not* to belong to the Set, or until enough iterations have passed to decide that it *does* belong to the Set
- Use this example to investigate task farm performance
  - Look at load imbalance in particular

Visualising the Mandelbrot Set

- Can visualise the Mandelbrot Set by colouring each point:
  a) Black if it belongs to the Set
  b) Otherwise another colour chosen from a gradient in proportion to how many iterations it took to discover the point does not belong to the Set

The result is a fractal →

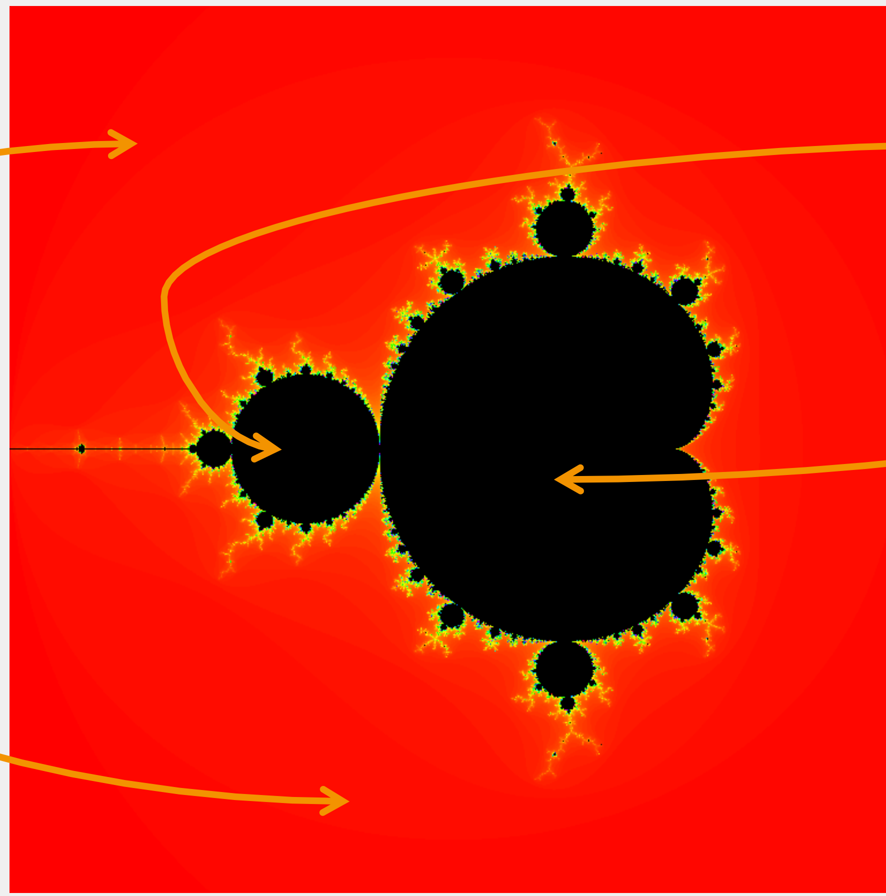


Points in the black region take more iterations (time) to compute → spatial work imbalance

Mandelbrot Set – spatial work imbalance

Very slow to compute

Very quick to compute



Parallelisation

- During the iterations for a given point, calculation values depend only on the previous calculation value at that point
  - decompose 2D grid into equally sized blocks
  - no communications between blocks needed.
- Don't know in advance how much work is needed.
  - number of iterations across the blocks varies.
  - work dynamically assigned to workers as they become available.

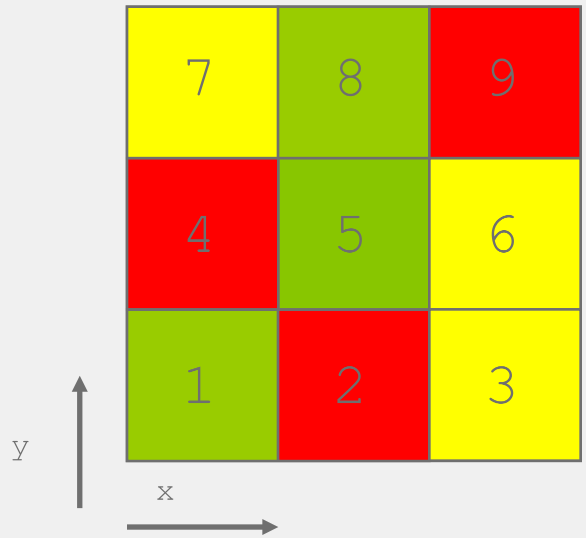
Implementation

- Split the grid into blocks:
  - each block corresponds to a task.
  - **master** process hands out tasks to **worker** processes.
  - workers return completed task to master.

Example: Parallelisation on 4 CPUs

master

CPU 1

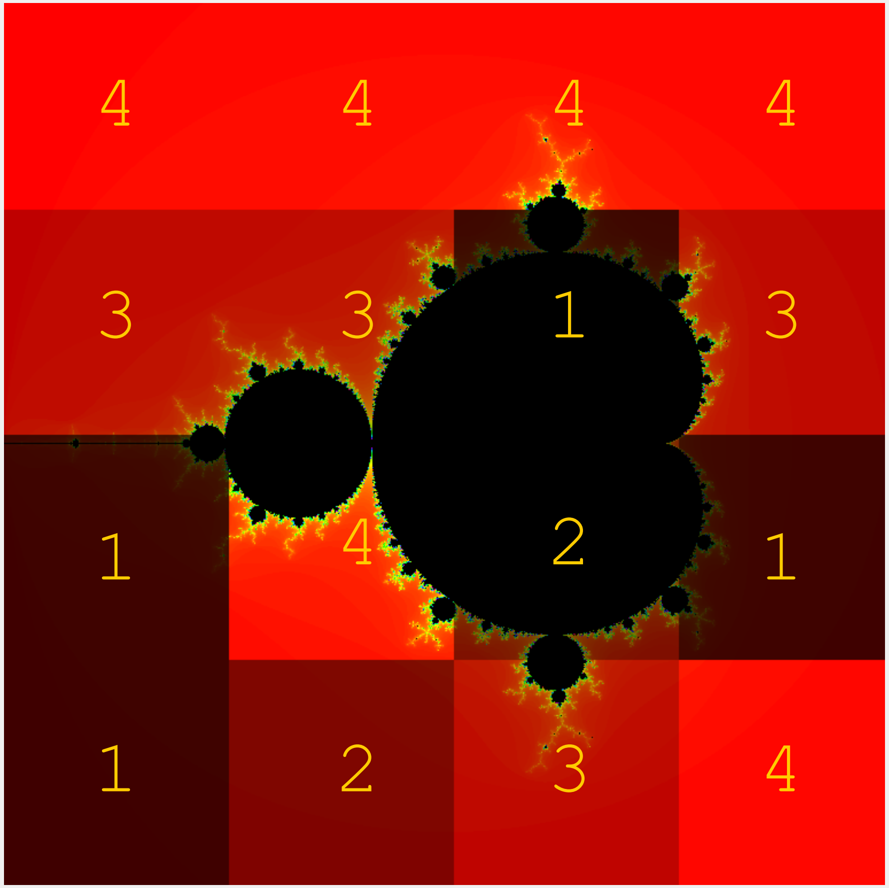


workers

CPU 2 CPU 3 CPU 4

- In diagram, colour represents which worker did the task
  - number gives the task id
  - tasks scan from left to right, moving upwards

Parallelisation cont.



- in images made by supplied code:
  - shading represents worker id
  - here we have added worker id as a number by hand
- e.g. taskfarm run on 5 CPUs
  1 master
  4 workers
- total number of tasks = 16

Exercise

- You are supplied with source code etc.
- Compile and run on ARCHER
    - visualise results
- Quantify performance results
- For a fixed number of workers
    - improve load balance by increasing number of tasks (decrease size)
    - compute LIF (load imbalance factor) to estimate minimum achievable runtime
    - is this minimum ever reached?

Fractal Practical

Outcomes

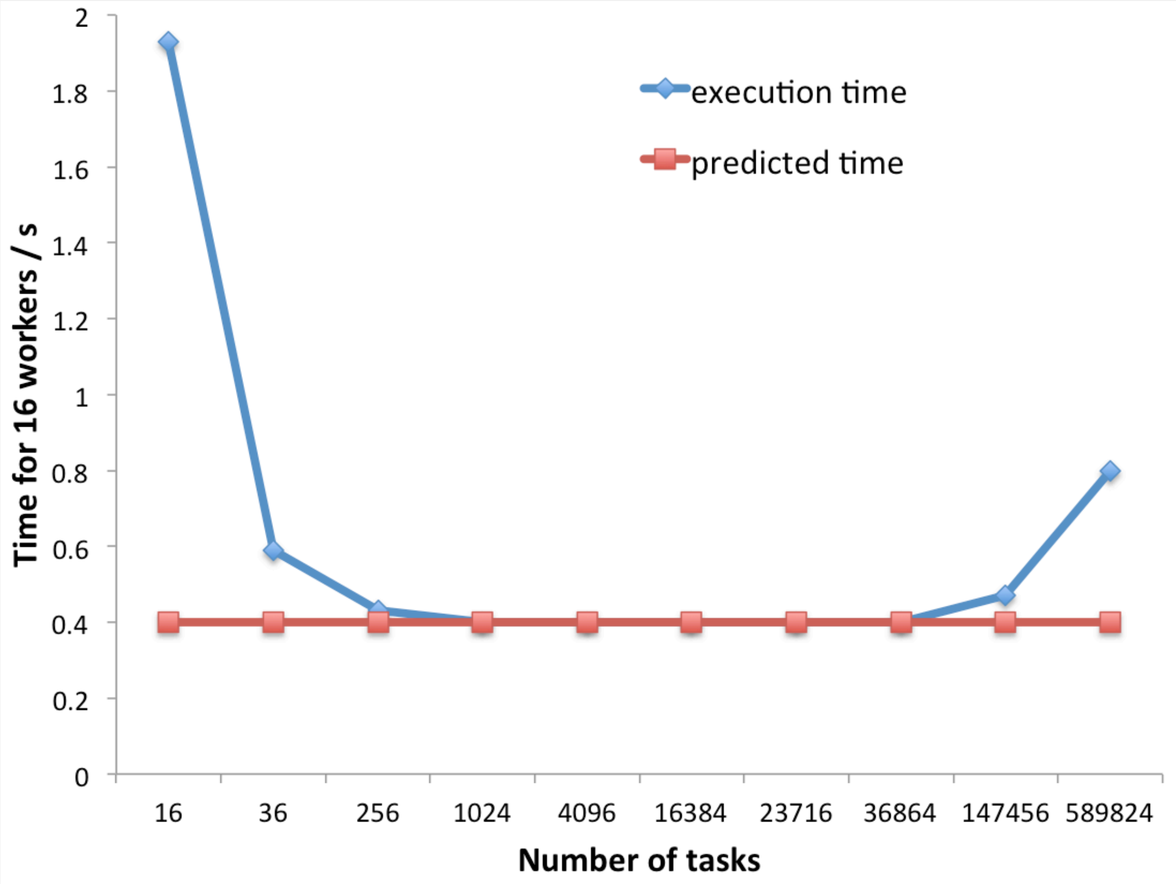
Example results
fixed number of workers
varying number of tasks

Example results for the default image size (768 × 768 pixels), fixed number of iterations (5000), fixed number of workers (16) and varying number of tasks :

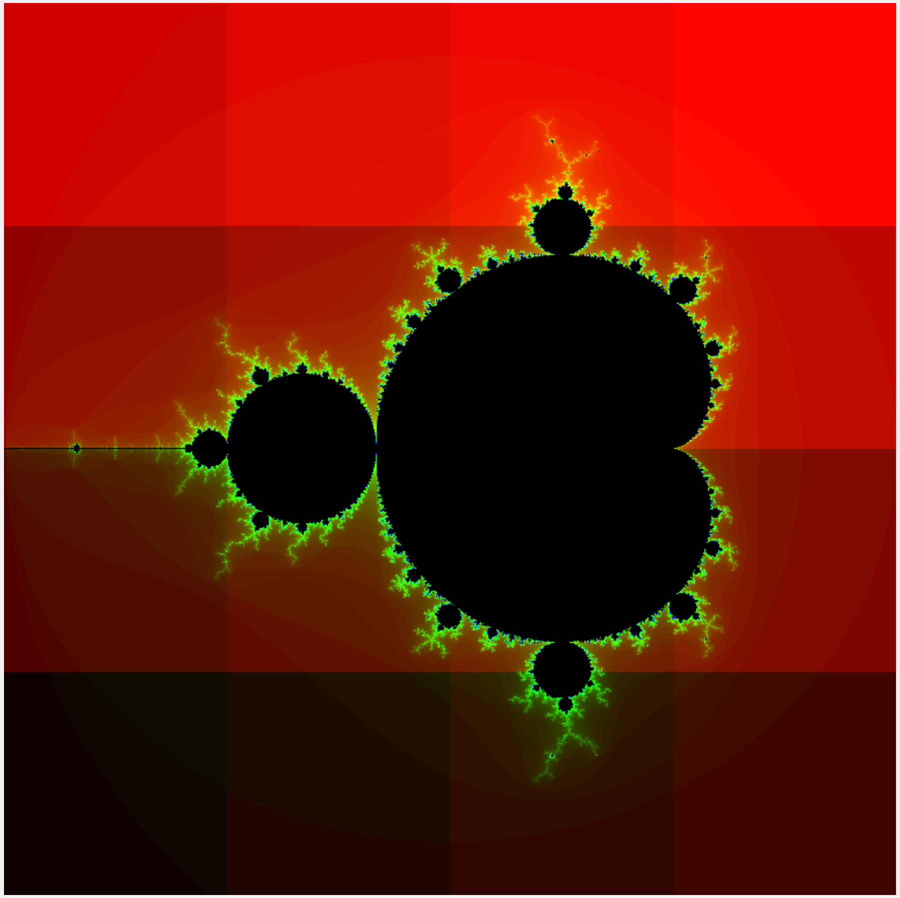
Number of Tasks (Task Size)	Time (s)	Load Imbalance Factor
16 (192 × 192)	1.93	5.034
64 (96 × 96)	0.59	1.501
256 (48 × 48)	0.43	1.108
4096 (12 × 12)	0.4	1.017
36864 (4 × 4)	0.4	1.003
147456 (2 × 2)	0.47	1.017
589824 (1 × 1)	0.80	1.006

Table 2: Example execution Times for 16 workers and varying number of Tasks.

Example results
fixed number of workers
varying number of tasks



16 workers and 16 tasks



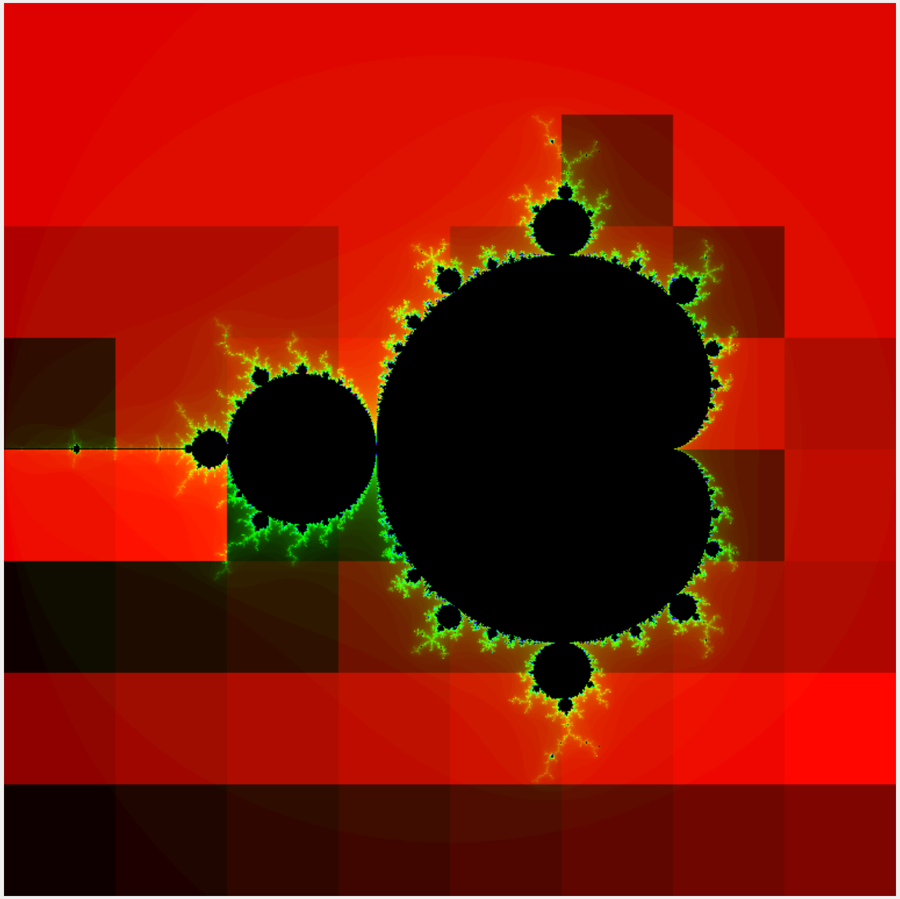
```
-----Workload Summary (number of
iterations)----

Total Number of Workers: 16
Total Number of Tasks:   16

Total   Worker Load: 498023053
Average Worker Load: 31126440
Maximum Worker Load: 156694685
Minimum Worker Load: 62822

Time taken by 16 workers was
1.929219 (secs)
Load Imbalance Factor: 5.034134
```

16 workers and 64 tasks



```
-----Workload Summary (number of
iterations)---------

Total Number of Workers: 16
Total Number of Tasks:   64

Total   Worker Load: 498023053
Average Worker Load: 31126440
Maximum Worker Load: 46743511
Minimum Worker Load: 10968369

Time taken by 16 workers was
0.586923 (secs)
Load Imbalance Factor: 1.501730
```

Key points to take away

- TASK FARMS
- Also known as the master/worker pattern
- Allows a master process to distribute work to a set of worker processors.
- Can be used for other types of tasks but it complicates the situation and other patterns may be more suitable for implementing.
- Master process is responsible for creating, distributing and gathering the individual jobs.
- Can improve load balance by using more tasks than workers
  - with some overhead
- Load imbalance adversely affects performance
  - especially as number of processors increases

Key points to take away

TASKS

- Units of work
- Vary in size, do not have to be of consistent execution time. If execution times are known it can help with load balancing.

QUEUES

- Master generates a pool of tasks and puts them in a queue
- Workers assigned task from queue when idle

Key points to take away

LOAD BALANCING

- How a system determines how work or tasks are distributed across workers (processes or threads)
- Successful load balancing avoids idle processes and overloading single cores
- Poor load balancing leads to under-utilised cores, reducing performance.

Key points to take away

COST

- Increasingly important
- Finite budgets require optimal use of resources requested.
- Load balancing is just one method of ensuring optimal usage and avoiding wasting resources.
- More power and resources do not necessarily mean improved performance.
- Always ask – is it necessary to run this on 4000 cores or could it be run on 2000 more efficiently?