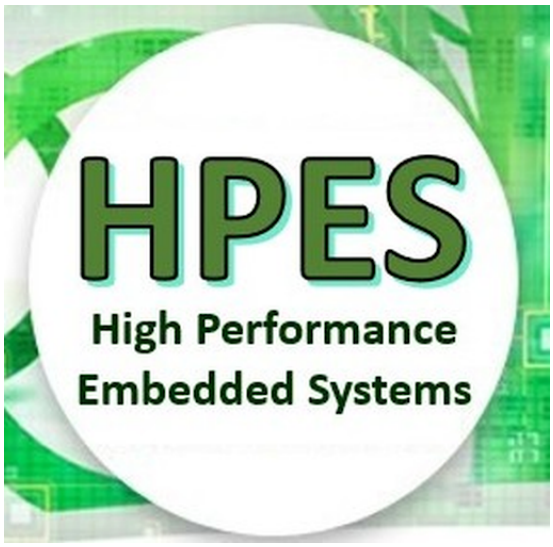

EEE4120F

High Performance Embedded Systems

PRACTICAL 2

Introduction to MATLAB Parallel Computing Toolkit(PCT)

Course Team

Course Convener: Associate Professor Simon Winberg

Teaching Assistant: Travimadox Webb

University of Cape Town 2026

Practical 2

Parallel Computing Toolkit

Summary of Deliverables

- **Duration:** 2 Weeks (27th Feb - 13th Mar)
- **Report:** A short technical report in IEEE Confrence Format (PDF, Max 3 Pages) via Gradescope assignment **Practical 2 Report**.
- **Code:** Submit your code(run_analysis.m) to Gradescope assignment **Practical 2 Code**.

1 Objectives

In this practical, you will explore the MATLAB **Parallel Computing Toolbox (PCT)**. You will move beyond sequential execution to utilise multi-core CPUs.

Core Concepts:

- **Workers & Pools:** Managing overhead.
- **Loop Parallelism:** `parfor` vs `for`.
- **Load Balancing:** Handling tasks of unequal duration.

2 Introduction & Theory

The Mandelbrot set is a set of complex numbers c for which the function $f_c(z) = z^2 + c$ does not diverge when iterated from $z = 0$. The sequence is defined as:

$$[c,\ c^2 + c,\ (c^2 + c)^2 + c,\ \dots]$$

The Coordinate System

The complex number $z = a + ib$ corresponds to (x, y) coordinates on an image. For this practical, we will render the standard region of the Mandelbrot set defined by:

Real Axis (x): $[-2.0,\ 0.5]$ **Imaginary Axis (y):** $[-1.2,\ 1.2]$

The Algorithm (Pseudocode)

The color of each pixel represents the number of iterations required for the magnitude of z to exceed 2 (the escape condition). If $|z| \leq 2$ after `max_iterations`, the point is considered part of the set (usually colored black).

```
for each x,y coordinate
    x0, y0 = x, y
    x = 0
    y = 0
    iteration = 0
    while (iteration < max_iterations and x^2 + y^2 <= 4 )
        x_next = x^2+y^2 + x0
        y_next = 2*x*y + y0

        iteration = iteration + 1

        x = x_next
        y = y_next

    return color_map(iteration)
```

3 Standard Testing Resolutions

Use these specific resolutions and iteration limits to benchmark your Mandelbrot set generation.

Name	Width	Height	Megapixels	Max Iterations
SVGA	800	600	0.48 MP	1000
HD	1280	720	0.92 MP	1000
Full HD	1920	1080	2.07 MP	1000
2K	2048	1080	2.21 MP	1000
QHD	2560	1440	3.69 MP	1000
4K UHD	3840	2160	8.29 MP	1000
5K	5120	2880	14.75 MP	1000
8K UHD	7680	4320	33.18 MP	1000

4 Tasks

4.1 Task 0: Image Plotting

Deliverable

Function to implement: `mandelbrot_plot.m`

Implement a function to plot and save the Mandelbrot image generated.

4.2 Task 1: The Sequential Baseline

Deliverable

Function to implement: `mandelbrot_sequential.m`

Implement a standard sequential function to:

1. Translate the Pseudocode from Section 2 into MATLAB code using nested `for`-loops. Use the standard region coordinates defined in Section 2.

2. Run this function for **ALL** resolutions in the table and store the corresponding mandelbrot images.
3. Compare performance against the parallel version.

Warning

Constraint: The sequential version execution times on resolutions higher than Full HD might become a bit long.

4.3 Task 2: Parallel Processing

Deliverable

Function to implement: `mandelbrot_parallel.m`

Implement a parallel version using the parallel pools and/or parfors.

1. Adapt your sequential code by replacing the outer loop with a `parfor` loop. Ensure that the variables are correctly classified (sliced vs. broadcast).
2. Set your parallel pool to use **2 workers**.
3. Run this function for **ALL** resolutions in the table and store the corresponding mandelbrot images.
4. Repeat steps 1-3 by adjusting the number of workers gradually from 2 to max physical cores of your machine.
5. Compare performance against the sequential version.

4.4 Task 3: Performance Analysis

Deliverable

Function to implement: `run_analysis.m`

Perform a detailed benchmarking analysis comparing your `mandelbrot_sequential` against the `mandelbrot_parallel`.

- Present your execution time/speed up results clearly (e.g., using tables or graphs).
- Analyse the performance difference.
- Comment on the scalability of each approach as image size increases.

Performance Metrics

Speedup

Speedup measures how much faster the parallel version runs:

$$\text{Speedup} = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

Interpretation:

- Speedup = 1: No improvement
- Speedup = 2: Twice as fast (on 2-core system)
- Speedup = N: Near-ideal scaling (on N-core system)

Efficiency

Efficiency measures how well cores are utilized:

$$\text{Efficiency} = \frac{\text{Speedup}}{N_{\text{cores}}} \times 100\%$$

Interpretation:

- Efficiency = 100%: Perfect scaling
- Efficiency = 50%: Each core utilized 50% of potential
- Typical realistic efficiency: 70–90% (overhead from parallelization)

Amdahl's Law

Theoretical maximum speedup with P processors:

$$\text{Speedup} = \frac{1}{(1-f) + \frac{f}{P}}$$

Where f is the fraction of parallelizable code.

Report Requirements

Format your report as a technical article. Include the following sections:

- **Introduction:** Briefly outline the practical aim, your implementation strategy, and the benchmarking goals.
- **Methodology:** Describe the your mandelbrot implementation(s) approach, and the timing methodology briefly.
- **Results:** Present tables or graphs comparing performance across different image sizes.
- **Discussion:** Analyse the performance difference.
- **Conclusion:** Summarise findings, identify limitations, and suggest potential improvements.

5 Marking Rubric

Category	Component	Marks
Implementation (25%)	`mandelbrot_sequential`	10
	`mandelbrot_parallel`	10
	Code Quality & Verification	5
Report (75%)	Introduction	10
	Methodology	15
	Results Presentation	15
	Discussion	**20**
	Conclusion	10
	Formatting	5
Total		**100**

Table 1: Mark Allocation for Practical 2