HW1 - Getting Started with Profiling

Shiraz University

This assignment requires the implementation and optimization of matrix multiplication on CPU architectures. Students should use two powerful profiling tools, gprof and perf, to analyze how C/C++ optimizations—such as loop order, tiling, and unrolling—impact execution time and hardware metrics like cache misses.

Parallel Algorithms

Prof. Farshad Khunjush

Teacher Assistants

- **AmirMohammad Kamalinia**
- **Amirreza Rezvani**

A. Environment Preparation

You should prepare a Linux environment with the necessary development tools.

Step 1: Linux Environment

- Set up a Linux environment (**Ubuntu is recommended**) using one of the following:
  - Your host computer
  - A Virtual Machine (e.g., VMWare)
  - **WSL** (Windows Subsystem for Linux)

Step 2: Required Development Tools

- Install the `build-essential` meta-package on Ubuntu.
- Verify the installation of the following tools:
  - `make`
  - `g++` (C++ Compiler)
  - `gprof` (Profiling tool)
  - `perf` (Advanced Profiling Tool)

> #### Important Note
>
> Profiling results can vary between runs due to factors such as cache state, CPU scheduling, and background processes. To ensure reliable measurements, **run each test multiple times** (e.g., 3–5) and **report the average value** for each metric.
>
> #### ⚠️ Important Note ⚠️
>
> Learn the `make` tool first — it's really important when doing redundant tasks and can significantly speed up your experiments.

B. Optimization of Matrix Multiplication

Implement the simple matrix multiplication of two **1024×1024** matrices in C/C++.

After each step, describe your **performance analysis** and **speedup**.

For every step, execute the program and collect metrics using both **gprof** and **perf**.

Profiling Requirements

1. gprof Metric (High-Level)

- **Total Execution Time** (Wall clock time or similar duration from the gprof report)

2. perf Metrics (Low-Level)

- CPU Cycles
- Instructions (and calculate `IPC = Instructions Per Cycle`)
- Cache Misses
- L1 Data Cache Load Misses

Optimization Steps

0. The baseline

No optimization needed just some good old matrix multiplication.

1. Data Type Comparison

- Implement the multiplication using two different data types:
  - `int`
  - `double`
- Report the execution time of each using **gprof** and **perf**.

2. Matrix Size Scaling

- Using your preferred data type from Step 1, change the matrix size to:
  - **2048×2048**
  - **4096×4096**
- Report the performance metrics time of each using **gprof** and **perf**.

3. Loop Order Permutation

- Try the 6 different possible loop orders for the basic matrix multiplication algorithm on the **4096×4096** matrix:
  - `ijk`
  - `ikj`
  - `jik`
  - `jki`
  - `kij`
  - `kji`
- Report the performance metrics of each using **gprof** and **perf**.

4. Loop Tiling (Blocking)

- Implement **loop tiling (blocking)** to improve cache utilization.
- Experiment with tiling sizes:
  - 16
  - 32
  - 64
- Report the performance metrics time of each using **gprof** and **perf**.

5. Loop Unrolling

- Implement **loop unrolling** for the innermost loop with factors of:
  - 4
  - 8
- Report the performance metrics of each using **gprof** and **perf**.

> ⚠️ Important Note ⚠️
>
> When compiling your code, choose whichever optimization flags you find important based on your research.
> You must be able to justify and explain the purpose and effect of each flag on the profiling results.

Report Format

Submit a complete report and your source code as a **ZIP file** to this Quera course by **2025/11/07**.

File Naming Convention

PA-F25-YOURFULLNAME-YOURSTUDENTNUMBER-HW1

Report Contents

1. Automation

- Explain your **Makefile** and how you used **Make** to automate and obtain results for each step.

2. Execution Instructions

- Provide an explanation of how to **compile and run** your code for each part (e.g., Linux command lines).
- Explain briefly the commands and related flags (inside your `Makefile`) you used (specially the compiler flags)

3. Performance Analysis

- Include a **complete analysis** of profiling results from **gprof** and **perf** for each step.
- Provide a **comparison** (preferably in tables and/or graphs. You can automate this part using a plotter script that extracts the desired metrics from the logs and visualizes them) of performance metrics for all tested parameters:
    - Data types
    - Matrix sizes
    - Loop orders
    - Tiling sizes
    - Unrolling factors

Identify and explain the **best-performing CPU implementation**.

- Include a **hardware analysis** explaining how your CPU characteristics affected profiling results.
    - Try to find meaningful relations between hardware and performance outcomes.

Notes To Consider

- You can use **AI as a tool**, but **DO NOT** use it blindly.
- You should be the **ENGINEER**, not AI ;)
- **DO NOT** use AI to generate the full report.
- Explain **any unusual patterns** in graphs and provide **justifications**.
- **Late submissions** will incur a penalty — please plan ahead and meet the deadlines.