UiT

THE ARCTIC
UNIVERSITY
OF NORWAY

Mandatory Assignment 1

INF-2200 (Fall 2016)

Department of Computer Science
University of Tromsø

The assignment

- Select a micro-benchmark
- Identify hotspots
- Implement “main loop” as a function in x86 assembly.
- Write C-code that:
  - initializes necessary data structures for your function.
  - calls your function.
  - tests the results for correctness
- Write a report

Find a benchmark

- A benchmark should be:
  - relevant: it should stress the systems we are interested in (CPU and memory).
  - realistic: it should solve a real-world problem.
  - repeatable: two executions should produce the same results.
- Note: in addition to the code, you also need to get or create realistic datasets.

Where to find a benchmark?

1. Use a benchmark from a benchmark suite
   - SpecCPU, phoronix, …
   - Advantage: relevant, realistic, and repeatable
   - Disadvantage: may be complicated to run, and source code may not be available.
2. Use a small CPU intensive tool/program
   - Compression, encryption, image conversion…
   - Advantage: relevant, realistic, and repeatable
   - Disadvantage: code may be complicated, and you need to create realistic datasets
3. Use an important algorithm
   - Sort, search, trees, graphs…
   - Advantage: simple code
   - Disadvantage: you need to find a program that uses the algorithm and create realistic datasets
4. If it is hard to find a benchmark, you can write the implementation yourself!

Important

- Benchmark must be single-threaded.
- Benchmark must be CPU bound (not I/O bound).
- Benchmark must not be selection sort, bubble sort, insertion sort, or quick sort.
- Benchmark should not use FPU instructions
  - Because floating point operations are executed by co-processor (x87 FPU), so CPU will be “off-loaded”.
  - And because FPU operations are harder in assembly, and won’t be covered by our lectures or colloquiums.
  - Which means…
    - Only use integer operations ☺

What to benchmark

- This assignment:
  - Create micro-benchmark, implement in assembly
- Assignment 3:
  - Use benchmark to evaluate a memory system design.
- Tip: make sure to understand what the benchmark will be used for before selecting one.
- Tip: make sure to do the profiling before you commit to a benchmark.

Identify hotspots

- Where in the benchmark is most of the execution time spent?
  - Often a loop or a function
- Use a profiler to find the hotspots
  - Most profilers “stop” the program after N instructions and record which part of the code is currently executing
  - Results show the distribution of the samples

Profiling using gprof

- Compile your code with gcc using the `-pg` flag.
- Find input data such that the execution time is at least a couple of seconds.
- Run the program to generate the `gmon.out` file.
- Analyze the file by running `gprof <executable>`
- Tip: profilers such as Kcachegrind do not require compiling the code with the `-pg` flag.
- Tip: make sure that you get the same results when re-running the program.
- Tip: reduce I/O overhead by running the program multiple times before analysis (also use /dev/null for output).

Deliverables

- Code
- Written report
  - Maximum 6 pages
  - One report per group (if working in group)
  - Goal: expert reader should be able to redo your work by reading only the report
- Use directory named “abc001-Px/”
  - Or all usernames of the group separated by hyphen (-)
  - Replace X with the project number (1 to 3)
  - E.g. abc001-P1/
- The directory must contain:
  - A directory named “doc”, containing report.**pdf**
  - A directory named “src” containing code, Makefiles, READMEs etc
    - **NO** compiled files. Delete executables etc before you hand in.
- Groups only need one member to hand in! Use group hand in!
- Maximum two per group

Report

- Background: your benchmark
- Methodology: how did you measure hotspots?
- Don’t use full code in report
  - You may discuss parts of code, e.g. expensive assembly instructions etc.
- How to run and test the code
- Problems: bugs and other issues?
- References

Deadline

- September 14th @ 12:00 PM (noon)
- Hard deadline!

- Also, send an e-mail to marius.f.wiik@uit.no by Monday September 5th with names (of both students if working in group) and choice of benchmark.
- You do not need to wait for your benchmark to be accepted before starting
  - We will tell you if you need to change it
- It doesn't matter if several groups use same algorithm, but implementation *should* be different.

A few suggestions to benchmarks…

- Sieve of Eratosthenes
- Merge sort
- Heap sort
- (Square) matrix multiplication
- Encryption algorithms:
  - One of the TEA encryption algorithms
  - RC4
  - …
- Hash algorithms:
  - Adler-32
  - …

Extra credit

- Measure run time of your benchmark and compare with the C equivalent. Graphs!
- Enable optimization on the compiler, try to beat the different levels (O0, O1.. Etc)
- Any place in your code where you can save memory accesses?
- Not a requirement, focus on learning assembly.

Questions?