Operating Systems

Homework Assignment #4

Due 1.1.2015, 23:59

Part 1

Write a program `mat_mult.o` implementing matrix multiplication.

- The program will accept four or five command-line parameters:
  - First is number of threads t, second and third are input files containing matrices to multiply, the fourth is the output file to write the result into.
  - The fifth parameter is provided when $t > 0$, and specifies the $mode$ (explained later).
- Read the two input matrices from the given input files
  - The input files are **binary**. The first **4 bytes** are a single number n, indicating the size of the matrix $n \times n$ (square matrix). The next $4 \cdot n^2$ bytes are the matrix values. Each cell is **4** bytes (`int`).
  - Assume bytes-to-int data representation is **little endian**.
  - Abort with a proper error message on any error opening or reading the input files.
- Do not use 2D arrays! To allocate matrix variables, dynamically allocate array of arrays, e.g.:
  - `mat = (int**) malloc(sizeof(int*) * n);`
  - `for (i = 0; i < n; ++i)`
  - `mat[i] = (int*) malloc(sizeof(int) * n);`
  - **Do not** try to improve performance, etc. Define/allocate the matrices exactly as stated.
- Calculate the result – multiply the two input matrix variables into the output matrix variable.
  - Use variables! Do not read / write the files during the calculation.
  - If $t = 0$, just calculate the output in the main thread.
  - Otherwise ($t > 0$):
    - Create t threads to calculate the output. The main thread waits (`join`) for all threads.
    - A thread repeatedly calculates a row of output, assigned to it according to $mode$.
      - $mode = 1$: Each thread is given its index i $(0, \dots, t-1)$ as an argument, and calculates all rows k s.t. $k(mod\ t) == i$.
      - $mode = 2$: Keep a global variable initialized to 0. To get a new row, use a global lock (`mutex`) to ++ the variable, calculating by previous value.
      - $mode = 3$: Similar to 2, but with no lock. Read about `__sync_fetch_and_add` (google) and use it instead of locking.
    - When done (assigned a row greater than n), the thread terminates.
- When calculation (and all threads) are done - write result into output file, in the same format as above.
  - If the output file does not exist, create it. Otherwise, overwrite it.
  - Abort with a proper error message on any error opening or writing the output file.
- Measure the time the actual multiplication took
  - **After** reading input, **before** writing output!
  - Print a message containing the total time the operation took, and time per thread.

Guidelines

- Use **strtol** (man 3 **strtol**) to convert a string to integer. You may assume input is valid and in correct range.
- You may assume both input files have the same matrix size n.
- Ignore overflows in calculation.

Part 2

In this part of the assignment you will execute your program from part 1 and reason about your results.

- Execute `mat_mult.o` on **nova** or **soul**!
  - You should test on your VM, but execute on the servers for better results.
  - **Nova** and **Soul** each have 4 CPUs.
  - You should execute and get results at several different times, to count for server load.
- Test all 4 modes (serial, 1, 2, 3) several times.
- Use $t = \{1,2,4,8,16\}$.
- Use varying matrix sizes and values.
- Execute each test several times, and at different times, and average the results.
- Plot a graph with your *averaged* results.
  - Y axis is the time each execution took.
  - X axis is the number of threads t.
  - Two lines for each mode (time and time per thread) except serial – total of **7** lines.
- Explain and reason about the results you've seen.

Guidelines

- Do not forget to check the return values of all functions.
- **Do not forget** to comment out any auxiliary code (debug prints, etc.) in the final submission!
- **Submit:**
  - A **single** C files: `mat_mult.c`
  - A **PDF file** containing:
    - A graph with your measurements. It should be clear & easily readable!
    - The description of your findings, in English, ½ to 1 page.

Further explanation of modes

In each mode, threads calculate rows of the output matrix.
Each thread, in a loop, takes a certain row and calculates it according to the input matrices.
When a thread takes a row that is outside the matrix bounds ($> n$), it is done, and the thread terminates.
How a thread takes a row in each iteration of the loop depends on the mode:

1. Each thread is assigned a unique index (**not** ID!) from the range $\{0, \dots, t-1\}$.
   To take a row, a thread i takes increasing values of k, such that k satisfies the following: $k \ (mod \ t) == i$
2. A global variable is kept, initialized to 0, which determines the next row that should be calculated by some thread. To take a row, a thread locks a global mutex, increases the variable, and calculates the row according to the variable's previous value.
3. Same as 2, but without a lock. Instead, to take a row, a thread uses an atomic operation `__sync_fetch_and_add` for the variable, taking a row according to the result.