Speeding Up the Matrix Multiplication

Date assigned: Week 4 (Feb 13, 2024)
Date due: Monday **11 March**, 2024 at 11 a.m. sharp. (No late submission allowed.)
Submission at: Your section's GCR
Max points: 2050 points
Collaboration: Not allowed. It is a single person assignment. You can discuss only the requirements if there is any confusion in those.

The **goal** of this assignment is to provide you an opportunity to speed up a serial code using many of the ways we have been studying in the lectures. And using speed up metrics to analyze the speed up you might be achieving.

What to submit:

(1) You will need to provide us with a report that will journal your journey on how you solved the problems posed in this assignment, along with the specific answers to the questions.
(2) Your code with a README that will tell how to execute your code

We will use the matrix multiplication as an example that we will need to speed up. Many real world problems reduce to a handful of matrix operations and matrix multiplication is one of them. Example classes of problems where we need matrix multiplication include: Deep Learning and Neural Networks, Computer Graphics and Computer Vision, Scientific Computing and Simulation.

Understanding why matrix multiplication is important:

Following are three powerful examples where efficient matrix multiplication is crucial:

1. Deep Learning and Neural Networks:

- Matrix multiplication is at the heart of many operations in deep learning, such as feedforward and backpropagation in neural networks.
- In deep learning models, matrices represent the weights between neurons in different layers. During training, these weights are updated through matrix operations, including multiplication.
- With the increasing complexity and size of neural network models, efficient matrix multiplication becomes essential for reducing training time and improving performance.

2. Computer Graphics and Computer Vision:

- In computer graphics, transformations such as translation, rotation, scaling, and projection are often represented using matrices.
- Efficient matrix multiplication is crucial for rendering 3D scenes, transforming vertices, and applying various effects such as shaders and filters.
- Similarly, in computer vision tasks like image processing, object detection, and feature extraction, matrices are extensively used for representing images, filters, and transformations.
- Real-time applications like video games and augmented reality heavily rely on optimized matrix operations to achieve smooth and responsive graphics rendering.

3. Scientific Computing and Simulation:

- Matrix operations are prevalent in scientific computing for solving complex mathematical problems, including linear algebraic equations, differential equations, and finite element analysis.
- Applications include computational fluid dynamics, structural mechanics, weather modeling, and quantum mechanics simulations.
- High-performance matrix multiplication is essential for accelerating these simulations and analyses, enabling researchers and engineers to study phenomena with higher accuracy and resolution.
- Additionally, large-scale simulations often require parallel and distributed computing techniques to handle massive datasets and computational workloads efficiently.

In each of these examples, efficient matrix multiplication plays a critical role in enabling high-performance computing and accelerating the execution of complex algorithms across various domains.

You need to select some target hardware (that is reasonably recent). It could be your laptop or some machine in a lab. Though, you will need to use the same machine for all parts of this assignment. Please don't choose for example some desktop machine sitting at your home because it will be a pain to bring that box to the campus for your assignment evaluation.

Input:
We will use square matrices of dimension 4096 X 4096 for this assignment. You are welcome to write your code such that it could work on any dimensions (not just square matrices). Your program should take the path to an input file (whose format we shortly define) and should generate output in a file provided on the commandline. While calculating times, be careful to separately report IO times (reading input from file and writing to the output file).

$ ./matrixmul <fully_qualified_path_to_an_input_file> <fully_qualified_path_to_output_file>

We will recommend using a Linux based computer for this assignment. You can use non-Linux systems if you wish so, though course staff might not be able to provide any help in that case. You can try using Linux in a virtual machine (or a docker). But be wary that doing so can slow down your system and limited resources will be available for your code speed-up.

Part 0: [50 points] Please tell us about the hardware you selected. We need the full details of the processor such as processor family, vendor, its vector capabilities, and number of cores and number of hyper-threads (if present), amount of RAM installed and the types of hard disks of any flash disks. You might like to see the following command for Linux.

(1) lspci
(2) lscpu
(3) cat /proc/cpuinfo

Please tell us your machine's state after boot up in terms of cpu utilization and memory usage if you do not run any program after booting. During this assignment, please turn off all other programs and any background processes to get a good speed up and reliable readings.

Input file format:
Opcode
Matrix data type
Dimensions of the first matrix
First Matrix data in rows
Dimensions of the second matrix
Second Matrix data in rows

Opcode:
1: run python based matrixmul
2: run java based matrixmul
3: run c based matrixmul
4: run c based matrixmul but with vector instructions based speedup
5: run c based matrixmul but with pthreads based multithreading but without any vector instructions
6: run c based matrixmul with pthreads based mutlithreading and vector instrutions together

Matrix data type:
1: 32 bit integers (positive and negative)
2: 32 bit floats
3: 64 bit ints
4: 64 bit doubles
(You should think about protability of your code a bit at this point)

Dimensions of the matrix:
4096X4096

Note: When you will be generating test data for yourself, please use some script to generate the data. Don't do it manually! This is a little opportunity to learn some scripting language to quickly do such tasks. Each matrix element will be separated by a space. The line will terminate on line feed (in case of Linux. It might be carriage return and line feed for Windows).

(We will use the above dimensions for this assignment but it is good to be of variable length and to verify that dimensions of both matrices are amenable for multiply operation.)

Output file format:
Your program should output:

Matrix data type
Dimensions of the resultant matrix
Rows of the resultant matrix after multiplication

Note: Beware of overflow or underflows after multiplication.

Part 1: [300 points] Programming language matters!

In the first part you will code a simple serial program to do matrix multiplication using:

(1) Python
(2) Java
(3) And then finally C (please dont use C++)

Calculate the time of execution on the same data, on the same target machine you selected earlier. (That instruction is for all parts of this assignment.) Calculate any speed up (or slow down) by changing programming language. Also draw a graph of execution times. Please execute your program at least three times, take the average execution time with the standard deviation. Your graph should draw average execution time of multiple readings and should have the error bars by using standard deviation.

Part 2: [500 points] Using vector instructions for speed up

Figure out what vector instructions are available in your processor. Please don't use any external co-processor such as GPUs for this part. Please let us know about the list of vector instructions. If multiple choices are available, experimentally decide which vector instructions are giving you the best speed up. How will you deal with any overflow and underflow situations?

Add the execution times (again average of at least 3 runs and standard deviation) with your previous graph.

Part 3: [500 points] Using pthreds to use multiple cores

Use pthreads library in C code to use multiple cores for the matrix multiplication. Tell us how many cores you have and how many threads you made to get the best execution time. You can empirically find that number. Does using more threads than available cores hurts performance or not, or at what point does it start hurting?

Add the execution times (again average of at least 3 runs and standard deviation) with your previous graph.

Part 4: [200 points] Vectors instructions and multiple cores

Now use vector instructions and multiple cores to get the speed up.

Add the execution times (again average of at least 3 runs and standard deviation) with your previous graph.

Part 5: [500 points] Analyses

(a) What did you learn in this assignment? Did you have some interesting insights?
(b) Which single way gave you the maximum speed up in relative terms? Programming language change, vector instructions or multithreading?
(c) The Karp-Flatt metric looks at serial part+overhead while we change the number of processors, though if we extend the definition of “processor” we might be able to use that metric. Assuming that any speed up from language change to vectors to multithreading to (vector+multithreading) is due to increasing processor count, what do your metric numbers suggest about the quality of your speed up and is it plateauing?

Note: Don’t forget about **correctness**. Whatever you do to speed up your serial code, the result should be correct. There is not much fun getting wrong results quickly. Getting points in this assignment, it is absolutely necessary to get the correct results. You will not be able to say for example that look I have used vector instruction x to speed up. Though my results are wrong, but give me some marks for using vector instructions.