Exercise 2: Performance Benchmarks, Finite Difference Discretization - a Report

Paul Kang, Bailiiang Li, Max Spannring, Moritz Techen

November 2024

Introduction

The following is a report, discussing the results we obtained for the three tasks of exercise 2, as a part of the lecture 360.242 Numerical Simulation and Scientific Computing I.

Task 2: Estimate and Benchmark the Performance of dense Matrix-Matrix Multiplication

Estimating & Modeling

The goal of this subtask was benchmaking the performance of different implementations of dense matrix-matrix multiplications (MMMs) on a single thread. All of these calculations were made with a 11th Gen Intel i7-1165G7 in mind, and the relevant CPU stats were taken from the official product website [1].

In order to calculate the machine Balance B_m, defined as $B_m = \frac{\text{memory bandwidth [Byte/s]}}{\text{peak performance [FLOPs/s]}}$, we first need to know the peak performance of our machine. Results from Task 1 were fluctuating quite a bit, ranging from 1.2e10 to 1.7e10 FLOPs/s, so we decided to use a number calculated from the datasheet of the CPU [1]:

$$\text{peak FLOPs/s} = \text{clock speed} \cdot \text{FLOPs per cycle}$$

With a clock speed of 4.7 GHz and 16 double precision floating point operations per cycle, we arrive at 75.2 Giga FLOPs per second. The memory Bandwith is the product of effective data rate, bus width and the number of channels. Plugin in the numbers for our CPU, we get

$$\begin{aligned}\text{Memory Bandwidth [Bytes/s]} &= 3200 \cdot 10^6 \text{ [Transfers/s]} \cdot 8 \text{ [Byte]} \cdot 2 \text{ [Channels]} \\ &= 512 \cdot 10^8 \text{ [Bytes/s]}\end{aligned}$$

This gives us for the machine balance:

$$B_m = \frac{7.52 \cdot 10^9 \text{ [Byte/s]}}{1.5 \cdot 10^{10} \text{ [FLOPs/s]}} = 0.680850... \text{ [Bytes/FLOP]}$$

In Order to calculate the code balance $B_c = \frac{\text{data traffic [Bytes]}}{\text{floating point operations [FLOPs]}}$, we need to know the size of the data that needs to be accessed for one MMM with matricies of the size N x N. As long as we are working with double precision floats, which have a size of 8 Bytes, the total data size is given by

$$\text{data traffic [Bytes]} = 3 \cdot N \cdot N \cdot 8$$

The Number of Floating point operations needed o perform a MMM of two N x N matrices is $2 \cdot N^3$. Combining this expression with the one above, we get an expression for the code balance:

$$B_c = \frac{3 \cdot N^2 \cdot 8 \text{ [Bytes]}}{2 \cdot N^3 \text{ [FLOPs]}} = \frac{12}{N} \frac{\text{[Bytes]}}{\text{[FLOPs]}}$$

If we want to use the maximum power of our CPU, the data transfer to our CPU has to be running on the maximum capacity as well. This means, that the code balance has to be larger than the machine balance. We can then rearrange this inequality to figure out which size N of the matrix would utilize the peak performance of our system.

$$B_c \geq B_m \implies \frac{12}{N} \geq 0.68 \implies N \gtrapprox 18$$

That means, as soon as the matrix size is roughly larger than 18x18, the CPU is working to its full potential. Assuming the runtime for Matrix-Matrix Multiplications is Computationally Bound, we can derive a formula for calcutaing the minimum runtime of such an operation by dividing the total number of FLOPs required through the maximum number of FLOPs/s.

$$t_{runtime} = \frac{3 \cdot N^3}{1.504 \cdot 10^{10}}$$

In the following table, the runtimes for a few specific values were calculated:

N	$t_{runtime}[s]$
1000	0.1995
2000	1.596
5000	24.333

Table 1: Runtime for MMMs with different sizes

Results

In the following graphic, the results are plotted. Apparently, the runtime of BLAS and Eigen is lower than the theoretical minimum that was calculated above. This suggests that those frameworks might be using more sophisticated algorithms, that require less floating point operations than assumed. Another observation: for large N ($N \gtrapprox 1000$), the actual runtime for the custom implementation of the MMM is much higher than the calculation suggests. Since the calculation is based on the assumption that the data can be accessed instantly, we can conclude that the calculation for large matrices is memory-bound, not computationally bound.

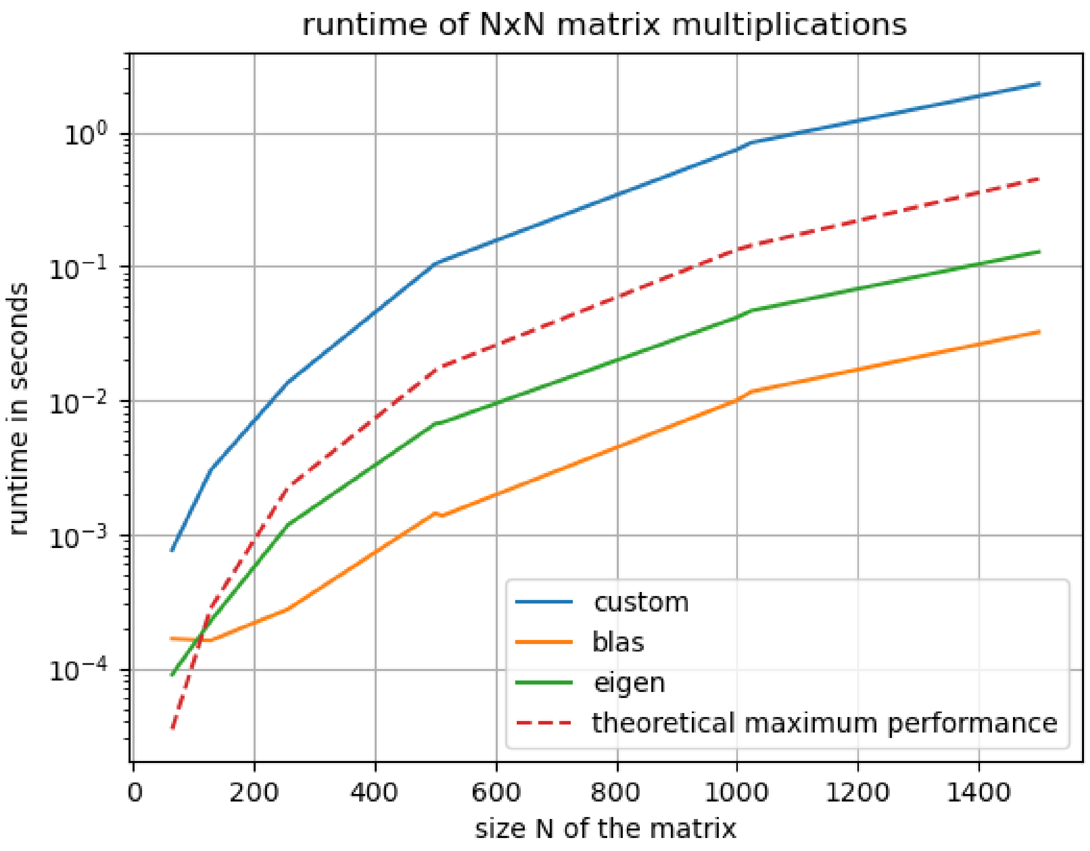

Figure 1: Runtimes of MMMs with different implementations

References

[1] *Intel® Core™ i7-1165G7 Processor (12M Cache, up to 4.70 GHz, with IPU) - Product Specifications*. en. URL: `https://www.intel.com/content/www/us/en/products/sku/208921/intel-core-i71165g7-processor-12m-cache-up-to-4-70-ghz-with-ipu/specifications.html` (visited on 11/20/2024).