COMP 605 - HW01

Akhil Perimbeti

February 11, 2022

Measuring Execution Time for Two Matrix-Multiplication Algorithms

Algorithm 1: The ijk-form

The ijk-form is not the fastest matrix-multiplication (A*B = C) method (equivalent to the jik-form) when compared to the kij-form and the ikj-form, but it is faster than the jki-form and kji-form method. **Inner loop**: The values of Matrix A are iterated through row-wise $(i, *)$. The values of Matrix B are iterated over column-wise $(*, j)$, and the values of the result Matrix C are iterated as fixed points (i,j). The ijk (and jik) forms have 1.25 misses per inner loop iteration, while the faster forms (kij and ikj) only have 0.5 misses per inner loop iteration.

Algorithm 2: The jki-form

The jki-form is the slowest matrix-multiplication (A*B = C) method, (equivalent to the kji-form) when compared to any of the matrix-multiplication algorithms. **Inner loop**: The values of Matrix A are iterated over column-wise $(*, k)$. The values of Matrix B are iterated as fixed points (k, j) , while the values of the result Matrix C are iterated over column-wise $(*, j)$. The jki (and kji) forms have 2.0 misses per inner loop iteration, which is the most compared to any other the other methods.

Summary of Code Results

Dimensions: $A = [1200]\times[1500]$, B = $[1500]\times[1100]$, **Result matrix C = A*B** =$[1200]\times[1100]$.

I was able to increase my stack size on my local machine server by writing the following line of code in my Makefile (-O3 optimization) to compile the program (with help of stackoverflow.com)

```
hw1: hw1main.c
	gcc -Wl,-stack_size,0x10000000 hw1main.c -O3 -o hw1
```

Table 1 shows the summary for both Algorithm 1 (ijk-form) and Algorithm 2 (jki-form) at two different optimization levels. The execution speeds are each algorithm's average runtime over 10 trials at each optimization level.

Optimization level	-	Execution speed (seconds)
-o0	Algorithm 1	7.462733
-o0	Algorithm 2	12.5844983
-o3	Algorithm 1	2.4710532
-o3	Algorithm 2	12.0549302

Table 1: Summary of Results - Averaged execution times (10 trials)

With an increased stack size, it is much easier to see the difference in execution times with and without optimization. We can see that Algorithm 1 has a much faster execution time compared to Algorithm 2 ($\approx$ 7.5 seconds vs. $\approx$ 12.6 seconds at the -O0 level). It is interesting to note that when the full optimization (-O3 level) is applied, the execution time for Algorithm 1 decreases significantly (almost by 5 seconds), while the execution time for Algorithm 2 only decreases by around half a second. This supports the notion that Algorithm 1 is more efficient, since even when the highest optimization (-O3) level is applied, Algorithm 2 is still significantly slower indicating that it is a fundamentally flawed method.