PROGRAMMING ASSIGNMENT 2

HUID: 30983793

1. Finding the theoretical crossover point

Let's denote the crossover point by *cp*. First, we need to find the run-time for both the normal matrix multiplication algorithm, and Strassen's algorithm. Let's start with the normal matrix multiplication algorithm. For each entry of the resulting matrix C, we need to make a certain number of additions and subtractions. Since both matrices are square, and of the same size (n by n), for each entry of C, we need to calculate n products of the to-be-multiplied matrices A and B. We then need to sum those multiplications up, to get the matrix product, which takes $n-1$ additions. The resulting matrix is also a n by n matrix, and thus:

$$T(n) = n^2(2n-1) = 2n^3 - n^2$$

For Strassen, we can make a recurrence to express the number of arithmetic operations involved. Each time we call the Strassen function on a matrices of dimension n, we perform 7 multiplications on matrices of size $n/2$, and 18 additions or subtractions of size $n/2$. Each of the involved matrices in the recursive call is of dimension $n/2$, so the recurrence is:

$$T(n) = 7T(n/2) + 18(n/2)^2$$

We now need to figure out when Strassen begins to have a lower cost than the normal matrix multiplication algorithm. To find this we can look at the crossover point where Strassen switches to regular matrix multiplication. We need to look at:

$$2n^3 - n^2 = 7(2(n/2)^3 - (n/2)^2) + 18(n/2)^2$$

This gives us $n_0 = 15$, or $cp = 15$. Yet, this is only true for even values of n (especially powers of 2, because we never reach an odd value in the recurrence). If we reach an odd value, we need to include an extra row and column of 0s of padding (to make the matrix size even). In this case we need to solve:

$$2n^3 - n^2 = 7(2((n+1)/2)^3 - ((n+1)/2)^2) + 18((n+1)/2)^2$$

This gives us $n_0 \approx 37$, or $cp \approx 37$. Thus, above $n = 37$, we would always want to use Strassen, for $15 \leq n \leq 37$ it would be ambiguous which algorithm to use from case to case, and for $n < 15$ we would always use the conventional method.

2. Implementation of Strassen in C++

At first, I attempted to implement Strassen in C, but then I had quite some trouble with memory allocation and pointer arithmetic, and then I couldn't figure out how to make the implementation with a dynamic padding in a static C array, so I decided to implement the algorithm in C++ using vectors and passing them by reference. This saved me a lot of pointer headaches, and allowed for dynamic resizing of the vectors.

At first, I also thought of allocating padding up to the next power of 2, yet that seemed very inefficient, because for 513 that meant going up to 1024, for 1025 going up to 2048, etc. I then decided to just go up one level, to the nearest even number, and keep dividing the matrix in 4 parts, until I reach an odd number for the dimension again. Because padding is at the bottom and the right, it wouldn't affect the structure of the original matrix.

To test the program I wrote a generate function in C++ to generate a .txt file with different matrices. I tested the algorithm with randomly generated matrices indexes from 1 to 10, with matrices comprised entirely of 1s, and matrices with consecutive numbers.

To run the program just type make in the terminal, and then type *./strassen 0 dimension inputfile*, where dimension is an integer, and inputfile is the file containing the matrices.

3. Results

On my machine the best performance seemed to happen at the crossover point $cp = 86$ for all matrix types. On the table below, we can see the performance for the 1s matrix for different values of the dimension for the Strassen and the conventional matrix multiplications.

Dimension	**Strassen Algorithm**	**Normal Multiplication**
255	216970 microseconds	251630 microseconds
256	218639 microseconds	260281 microseconds
257	245130 microseconds	267701 microseconds
511	1563620 microseconds	2143360 microseconds
512	1580466 microseconds	2122299 microseconds
513	1777262 microseconds	2157783 microseconds
1023	11086018 microseconds	19788168 microseconds
1024	14696774 microseconds	23181251 microseconds
1025	14630593 microseconds	25585148 microseconds

We can see that there is a big difference between theory and practice. I assume that this is the case because the cost is not 1 always one in practice, as we assumed in the asymptotic computation. There are many outside factors that can attribute to that: CPU usage, difference in compilers, space, efficiency of memory allocation, etc. All of these cold have positive or negative effects on the program, so we cannot make a calculation about what the performance would be in every single case.

Note that for matrices up to 512 the difference in the algorithms is negligible. However, for matrices of size 1000 and above, the difference becomes substantial and we would want to use Strassen.

4. Acknowledgements

First and foremost, I would like to acknowledge and thank professor Mitzenmacher for granting me an extension, without which I probably wouldn't have been able to do even half of this assignment. I would also like to acknowledge Fransisco Trujilo who pointed me in the right direction in Cabot Cafe when I was trying to figure out the theoretical crossover point. I would also like to thank whomever did the documentation at http://www.cplusplus.com/reference/ - dear Sir or Madam, you really saved me. Thank you for reading! :)