AMS 530 Final Presenation

Problem 4.4: Strassen's Algorithm

Sophia Nolas

12/1/22

Introduction

- Background
- The Algorithm
  - How does it work?
  - Why would we use it?
- Parallelization
- Implementation
- Performance Results
- Future Study

The Strassen's Algorithm

- How does it work?
- Algorithm to multiply two large matrices, A and B, size n
  - Divide the matrices into quadrants

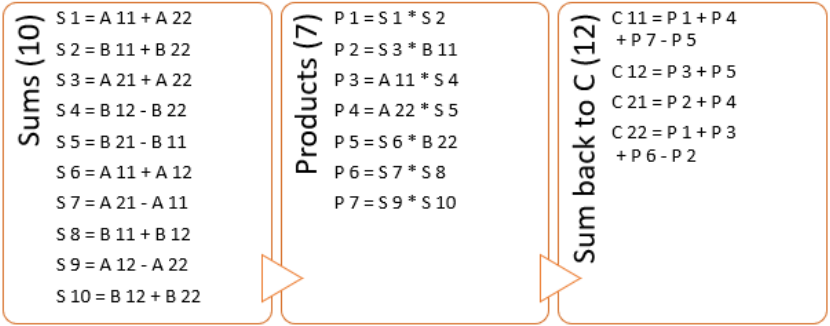
A11	A12
A21	A22

A11	A12	A13	A14
A21	A22	A23	A24
A31	A32	A33	A34
A41	A42	A43	A44

A11	A12	A13	A14	A15	A16	A17	A18
A21	A22	A23	A24	A25	A26	A27	A28
A31	A32	A33	A34	A35	A36	A37	A38
A41	A42	A43	A44	A45	A46	A47	A48
A51	A52	A53	A54	A55	A56	A57	A58
A61	A62	A63	A64	A65	A66	A67	A68
A71	A72	A73	A74	A75	A76	A77	A78
A81	A82	A83	A84	A85	A86	A87	A88

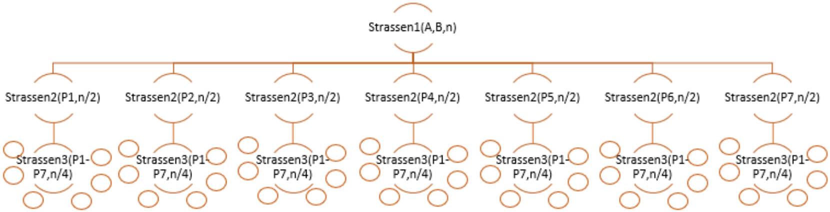
The Strassen's Algorithm

- How does it work?
- Algorithm to multiply two large matrices, A and B, size n
  - Divide the matrices into quadrants
  - Perform matrix operations on these subdivided matrix sections; each will have size n/2, which reduces the operational cost of each operation
  - Additionally, the naive algorithm requires 8 matrix multiplications; this one, only 7



The Strassen's Algorithm

- How does it work?
- Apply Strassen's Algorithm recursively on those 7 products
  - In this project, I did 3 levels
  - The first level requires 7 matrix products
  - At each further level, we will perform 7^L products



The Strassen's Algorithm

- Why would we use it?
  - Save matrix multiplications – only 7 per level, instead of 8 from naive method
  - Save matrix operation size – at each level, the matrices being added or multiplied are only size n/2 of the total size; and if we recursively apply this more times, that size only decreases

Parallel

- How to take the algorithm from one processor to parallel computing
- Since there are 7 products, it is simple to divide each product between 7 processors
  - Now the question is: how to divide it among 7p processors?
  - I was not able to implement this: but the idea I have is, send the same tasks to processor 1 and 8, and then divide the work between them; that is, label them via modulus to split up the work.
- All the processors do not need to use the entire matrix
  - Each processor can do the product it is assigned via the rank designation, and then send the result to the root function
  - Then the root function will receive these products and complete the final step, putting the sums together and forming the matrix C.

Implementation

- case: n=256

```
int S[10][128][128]; //level1 sums
//Strassensums(n,A,B,S);
for (int i =0; i<n/2; i++){
    for (int j=0; j<n/2; j++){
        S[0][i][j]=A[i][j]+A[i+(n/2)][j+(n/2)];
        S[1][i][j]=B[i][j]+B[i+(n/2)][j+(n/2)];
        S[2][i][j]=A[i+(n/2)][j]+A[i+(n/2)][j+(n/2)];
        S[3][i][j]=B[i][j+(n/2)]-B[i+(n/2)][j+(n/2)];
        S[4][i][j]=B[i+(n/2)][j]-B[i][j];
        S[5][i][j]=A[i][j]+A[i][j+(n/2)];
        S[6][i][j]=A[i+(n/2)][j]-A[i][j];
        S[7][i][j]=B[i][j]+B[i][j+(n/2)];
        S[8][i][j]=A[i][j+(n/2)]-A[i+(n/2)][j+(n/2)];
        S[9][i][j]=B[i+(n/2)][j]+B[i+(n/2)][j+(n/2)];
    }
}
```

Implementation

- case: n=256

```
//level1 products

if (rank==1){
    //P2=S3B11
    int P2[128][128] = { 0 };
    int B11[128][128] = { 0 };
    for (int i=0;i<n/2;i++){
        for(int j=0; j<n/2; j++){
             B11[i][j]=B[i][j];
         }
    }
    Strassen2(n/2,S[2],B11,P2);
    MPI_Send(P2,128*128,MPI_INT,0,1,MPI_COMM_WORLD);
}

if (rank==2){
    //P3=A11S4
    int P3[128][128] = { 0 };
    int A11[128][128] = { 0 };
    for (int i=0;i<n/2;i++){
        for(int j=0; j<n/2; j++){
            A11[i][j]=A[i][j];
        }
    }
    Strassen2(n/2,A11,S[3],P3);
    MPI_Send(P3,128*128,MPI_INT,0,1,MPI_COMM_WORLD);
```

Implementation

- case: n=256

```
if (rank == root){
    int P1[128][128] = { 0 };
    Strassen2(n/2,S[0],S[1],P1);

    int P2[128][128] = { 0 };
    int P3[128][128] = { 0 };
    int P4[128][128] = { 0 };
    int P5[128][128] = { 0 };
    int P6[128][128] = { 0 };
    int P7[128][128] = { 0 };

    MPI_Status status;
    MPI_Recv(P2,128*128,MPI_INT,1,1,MPI_COMM_WORLD,&status);
    MPI_Recv(P3,128*128,MPI_INT,2,1,MPI_COMM_WORLD,&status);
    MPI_Recv(P4,128*128,MPI_INT,3,1,MPI_COMM_WORLD,&status);
    MPI_Recv(P5,128*128,MPI_INT,4,1,MPI_COMM_WORLD,&status);
    MPI_Recv(P6,128*128,MPI_INT,5,1,MPI_COMM_WORLD,&status);
    MPI_Recv(P7,128*128,MPI_INT,6,1,MPI_COMM_WORLD,&status);

    //level 1 combine back to C
    for (int i=0; i<(n/2);i++){
        for(int j=0;j<(n/2);j++){
            //C11=P1+P4+P7-P5
            C2[i][j]=P1[i][j]+P4[i][j]+P7[i][j]-P5[i][j];
            //C12=P3+P5
            C2[i][j+(n/2)]=P3[i][j]+P5[i][j];
            //C21=P2+P4
```

Implementation

- case: n=256

```
void NaiveMult1(int n, int matrixA[][256], int matrixB[][256], int matrixC[][256]);
void NaiveMult2(int n, int matrixA[][128], int matrixB[][128], int matrixC[][128]);
void NaiveMult3(int n, int matrixA[][64], int matrixB[][64], int matrixC[][64]);
void NaiveMult4(int n, int matrixA[][32], int matrixB[][32], int matrixC[][32]);
void Strassen2(int n, int A[][128], int B[][128], int P[][128]);
void Strassen3(int n, int A[][64], int B[][64], int P[][64]);
```

- As you can see, I had to create a new function for each level of implementation of the Strassen and naive multiplication algorithms; this is obviously not the ideal solution, and if I had more time I would write a function that takes variable sized matrices, so I could simply call the same function at every level.

Performance Results

- For one processor: used Clock() function; for parallel, used MPI_Wtime()

processors	matrix size	clock time	email time
7p	N		
1	256	140768	
7	256	145685291	0:00:02
7	1024	562125320	0:00:03
7	4096	1698745465	0:00:47

Performance Results

- Speedup analysis is trivial in this case since it only goes from 1 to 7 processors

# of processors	time	speedup
1	140768	N/A
7	145685291	0.000966247

Future Work

- Define the functions recursively; so any level of Strassen's algorithm can be taken to further reduce the products
- Further parallelize the procedure
  - Using multiples of 7 processors
  - Sending information more precisely
- Combining algorithms
  - Rather than take the naive product inside the Strassen Algorithm, use another matrix product algorithm which is more time/cost effective

Conclusion

- Summary
- Results
  - Implement the Strassen Algorithm to 3 levels, on a matrix product between matrices of size n=256, 2056, and 4096
- Further work to improve the study:
  - Increased use of parallel processing
  - Combine algorithms

Questions?

Pergamon

Computers Math. Applic. Vol. 30, No. 2, pp. 49–69, 1995

0898-1221/95 $9.50 + 0.00

0898-1221(95)00077-1

Parallelizing Strassen's Method for Matrix Multiplication on Distributed-Memory MIMD Architectures

C.-C. Chou, Y.-F. Deng,* G. Li and Y. Wang
Center for Scientific Computing, The University at Stony Brook
Stony Brook, NY 11794-3600, U.S.A.

Dedicated to Professor James G. Glimm on the occasion of his 60th birthday

(Received and accepted October 1994)