**CSCE 735: Parallel Computing**

**Fall 2022**

**Major Project: Parallelizing Strassen’s Matrix-Multiplication Algorithm**

Submitted to the Faculty  
Of  
Texas A&M University  
By  
Sri Hari Pada Chandanam Kodi  
UIN: 932003040

Under the Guidance of

**Professor Vivek Sarin**

Department of Computer Science and Engineering

December 2022

A red and white sign

Description automatically generated with low confidence

**Objective of the Project:**

To develop a parallel implementation of Strassen’s recursive algorithm for computing matrix multiplications.

**Strategy**:

Developed a shared-memory code using OpenMP in C++. Inputs to the program are k, k', and the number of threads. The k value is used to set the dimensions of the Matrix (nxn with n=2k) & the k' threshold value to designate when the recursion should end. (Dimensions: 2k'x2k') number of threads (2numOfThreads) required to run the program. When numOfThreads = 5, the application is executed by 32 threads.

**Program Compilation & Execution:**

1. The code for parallelizing the Strassen matrix multiplication method is included in the file "strassen\_mult\_mat.cpp"
2. Run module load intel to load the compiler.
3. Use **icc** with the -**qopenmp** flag to compile C/C++ applications with OpenMP pragmas. Use o **icc -qopenmp -o strassen\_mult\_mat.exe strassen\_mult\_mat.cpp** to compile code.cpp into the executable code.exe.
4. Add the following parameters to the executable file: **k (matrix size), k' (threshold),** and **number of threads.**
5. **strassen\_mult\_mat.exe <k> <k’> <number of threads>**
6. The matrix dimensions (**nxn**) are defined by the value of "**k**" where n=2k and **k'** is the threshold value (real threshold value is **n/2k'**). number of threads is used to specify the number of threads that will be used to run the application.

**Testing**:

The code is run for the following combination of matrix sizes, threshold sizes, and thread counts. (k, k', and thread count).

k = 2,4,6,8,10. k’ = 1,2,4,6,8,10, numOfThreads = 0 to 10 (1 to 2^10 threads).

Here I have created a batch job file comprising the above combinations and ran cumulatively to get the desired results that are shown in the later part of the report.

**CASE-1: 4 x 4**

1. **k=2, k’=1 number of threads = 1 to 1024**

A picture containing text

Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=2, k’=2 number of threads = 1 to 1024**

Text

Description automatically generated with medium confidence

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

**CASE-2: 8 x 8**

1. **k=3, k’=1 number of threads = 1 to 1024**

A picture containing calendar

Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

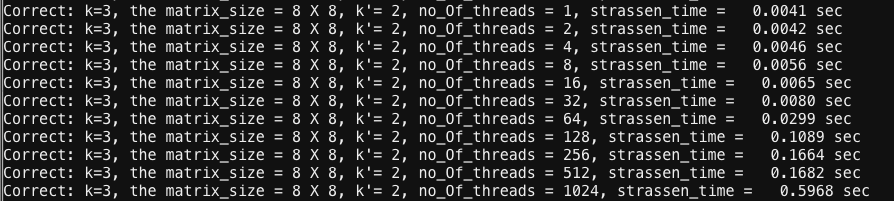
Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=3, k’=2 number of threads = 1 to 1024**



1. Speedup vs number of threads:

Chart, line chart

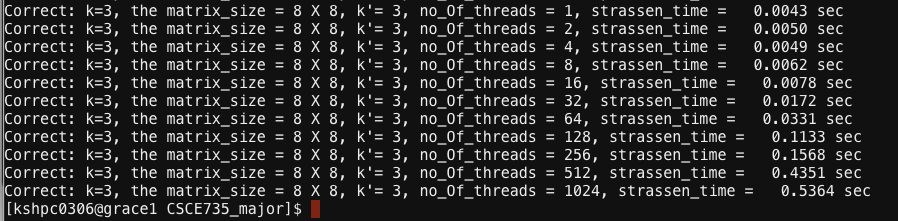
Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=3, k’=3 number of threads = 1 to 1024**



1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

**CASE-3: 16 x 16**

1. **k=4, k’=1 number of threads = 1 to 1024**

Text

Description automatically generated with medium confidence

1. Speedup vs number of threads:

Chart, line chart

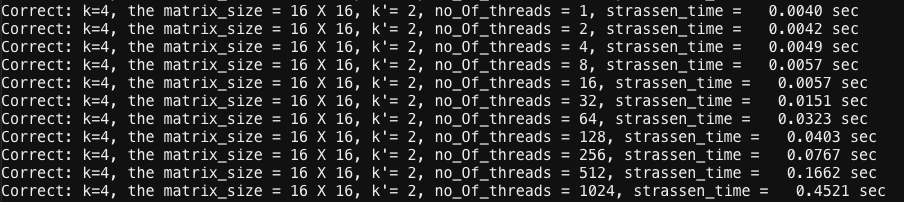
Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=4, k’=2 number of threads = 1 to 1024**



1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=4, k’=4 number of threads = 1 to 1024**

A picture containing calendar

Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

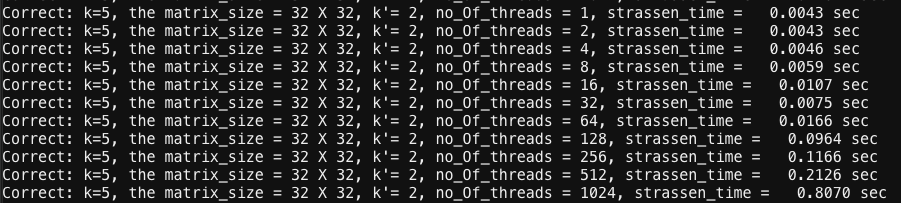
**CASE-4: 32 x 32**

1. **k=5, k’=1 number of threads = 1 to 1024**

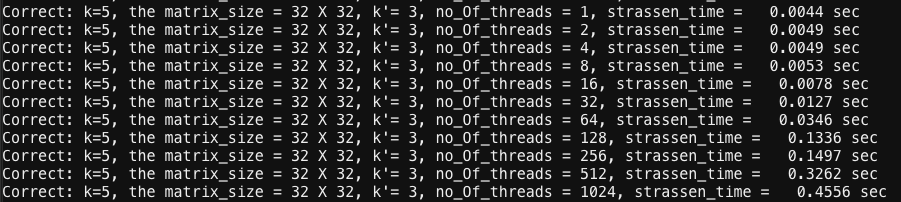
Text

Description automatically generated with low confidence

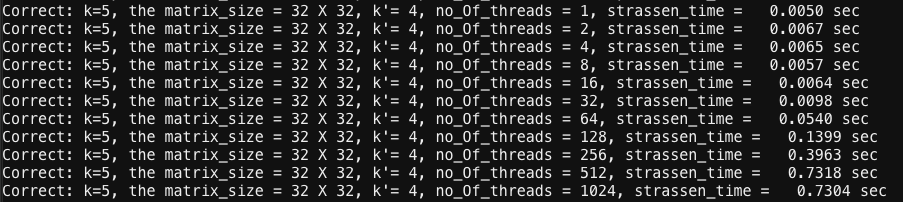
1. **k=5, k’=2 number of threads = 1 to 1024**



1. **k=5, k’=3 number of threads = 1 to 1024**



1. **k=5, k’=4 number of threads = 1 to 1024**



1. **k=5, k’=5 number of threads = 1 to 1024**

Calendar

Description automatically generated with medium confidence

**CASE-5: 64 x 64**

1. **k=6, k’=1 number of threads = 1 to 1024**

A picture containing text

Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

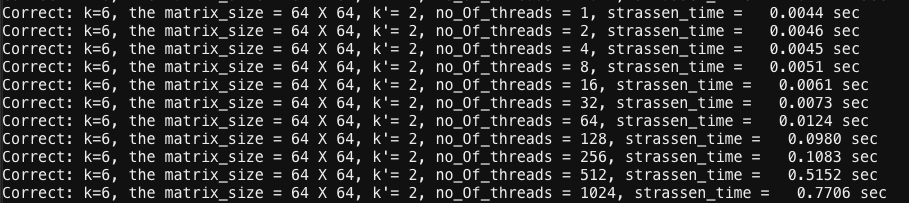
Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=6, k’=2 number of threads = 1 to 1024**



1. **k=6, k’=4 number of threads = 1 to 1024**

A picture containing text, file, keyboard, computer

Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=6, k’=6 number of threads = 1 to 1024**

Calendar

Description automatically generated with medium confidence

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

**CASE-6: 256 x 256**

1. **k=8, k’=1 number of threads = 1 to 1024**

A picture containing calendar

Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=8, k’=2 number of threads = 1 to 1024**

A picture containing text, file, computer

Description automatically generated

1. **k=8, k’=4 number of threads = 1 to 1024**

A picture containing text, file, computer

Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=8, k’=6 number of threads = 1 to 1024**

A picture containing text, file, keyboard, computer

Description automatically generated

1. **k=8, k’=8 number of threads = 1 to 1024**

A picture containing text, file, computer

Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

**CASE-7: 1024 x 1024**

1. **k=10, k’=1 number of threads = 1 to 1024**

A picture containing calendar

Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=10, k’=2 number of threads = 1 to 1024**

A picture containing text, file, computer

Description automatically generated

1. **k=10, k’=4 number of threads = 1 to 1024**

A picture containing text, file, computer

Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:

Chart, line chart

Description automatically generated

1. **k=10, k’=6 number of threads = 1 to 1024**

A picture containing text, file, computer, keyboard

Description automatically generated

1. **k=10, k’=8 number of threads = 1 to 1024**

A picture containing text, file, computer

Description automatically generated

1. **k=10, k’=10 number of threads = 1 to 1024**

A picture containing text

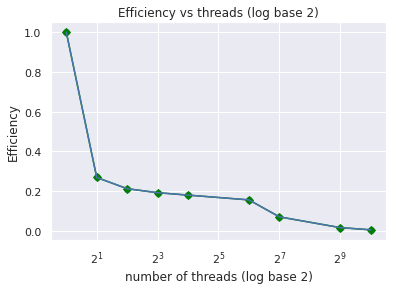
Description automatically generated

1. Speedup vs number of threads:

Chart, line chart

Description automatically generated

1. Efficiency vs number of threads:



**Analysis**:

1. The execution time for most of the tests is noted to be the shortest when threads are four, followed by a significant rise in execution time.
2. It has also been noticed that in most instances, the execution time continues to reduce until a certain number of threads (usually four) are reached, after which it begins to increase.
3. According to the results, the execution times appear to hit a low value when the number of threads is increased, and this increase in number of threads appears to be making things worse beyond the threshold, after which we see a rapid increase in the execution times due to context switching, unnecessary work assignment, and work idle time depending on other threads.

**Significance of threshold:**

1. By creating a threshold of 1 through k for all matrix sizes to understand how the threshold works and how the Strassen matrix multiplication algorithm works with varying thresholds and how it depends on the number of threads allocated to accomplish the operation.
2. As k' increases, the terminal matrix size decreases, and the number of recursive calls in the Strassen Algorithm increases, as opposed to a lower k' value, where the Strassen is run less times, producing a worse result than a basic implementation of multiplication. Consequently, a larger value of k' indicates a smaller size of matrix at the terminal condition, and therefore the Strassen multiplication approach is used for the majority of subproblems, producing a superior outcome. However, the rise in k' value has a threshold point beyond which the execution times appear to be increasing. As a result, there is an ideal k' value that yields the best outcomes.
3. As k' increases, and eventually if it is set to k, the terminal matrix becomes the smallest, and the problem becomes branched into the greatest number of recursive calls, and eventually due to a greater number of function calls, more space utilization, and more reliance on other recursive calls yields the worst result. As a result, the value of k' should be set as a compromise between the inadequacies of Standard Multiplication and the more recursive calls in Strassen.

**Design Choices:**

1. In code development, C++ and OpenMP are utilized. Because of previous assignments' established knowledge, OpenMP was employed.
2. Code is parallelized depending on the dependencies connected with certain pieces of code. The code used to compute the Strassen Matrices (M1–M7) is parallelized since their computations may be done independently. #pragma omp task was used to parallelize the parts of code that compute for M1 through M7 values.
3. All the procedures that allocate memory while building matrices run in serial, as does the code involved in creating matrices M1 through M7. Because of their interdependence, these are executed sequentially.
4. A recursive function call to Strassen Mat Mul is the longest and most time-consuming component of the code (). The use of openMP aided in parallelizing the code

**Conclusion**:

As a result, the algorithm's performance is affected by the threshold(k') and thread count. We may conclude that standard performs best for lower sizes, while Strassen computation beats standard when the size of the matrix is large.