PROJECT SPECIFICATION
CS 4504 – Distributed Computing

Part 1 and Part 2
Due Dates: See the Syllabus or TBD

A. Problem Statement

This project has two goals. The first goal is to design and implement a Client-Server paradigm using TCP to demonstrate that any two processes – a client and a server - can communicate using a server-router as a bridge. The second goal is to use threading capability to allow clients, as many nodes as possible, to be spawned for dynamic (re)configurability and scalability of the network. Each client process is then connected to a server-side core (simulated as a thread), where the cores/threads are invoked to perform certain computations as services.

The following is a hypothetical model of hybrid architecture – partly a Client-Server Distributed computing paradigm and partly a Parallel System computing paradigm.

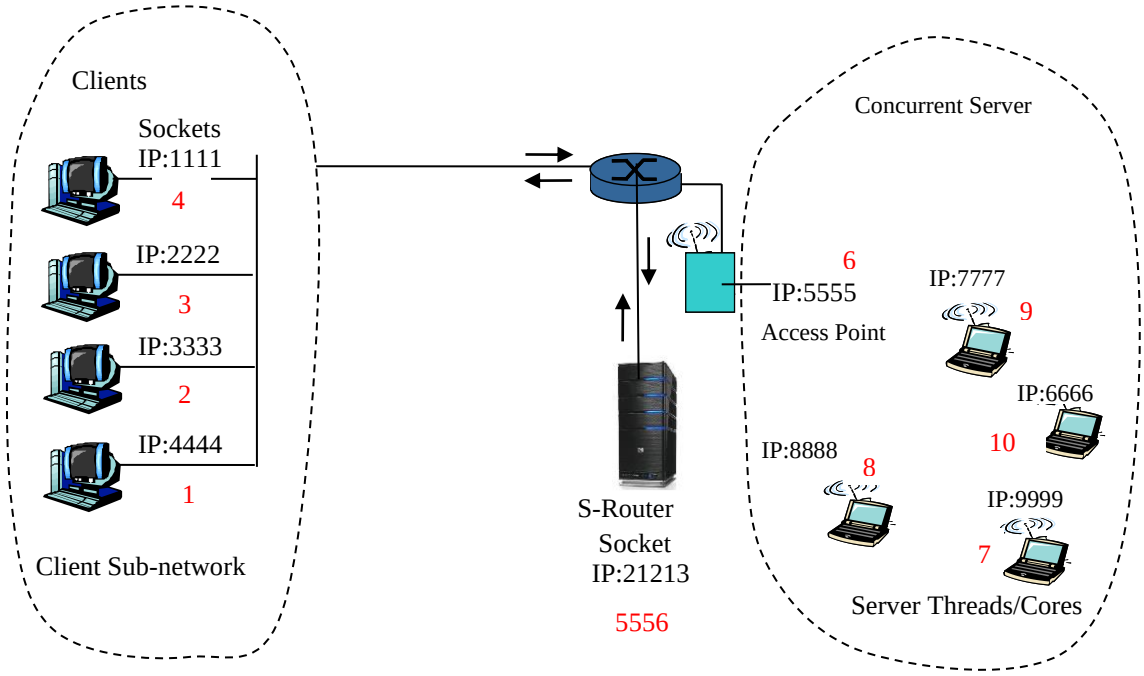


Figure 1: A Conceptual Model of a Concurrent Connection-Oriented Multithreaded/Multicore Server

Figure 1 depicts two subnets of nodes: Client Sub-network and Server-side subnet which is comprised of a multithreaded single node or a multicore cluster (if more than one node is used); or a mix of both. *The server-side subsystem has no 'client' functions. It simply computes or provides services to clients.*

The desire is to have as many clients as possible make connections simultaneously or concurrently to the server via TCP connections for direct communication through the

usual Server-Router, as depicted in Figure 1. The *S-Router has in its routing tables ONLY the necessary routing information for all the clients and server-threads (pairs) that are interconnected*.

The IP numbers in Figure 1 are *hypothetical*. For your design and implementation, use nslookup, ipconfig, or any tools on the laptops used in (your) configuration to determine the actual IP addresses. (Note: Over time, it has been found that the KSU-UITS has firewalled the PCs on campus; thus, only wireless/laptops in the *same network, or the Atrium building,* can be set up in a configuration for reachability.)

B. What to Do

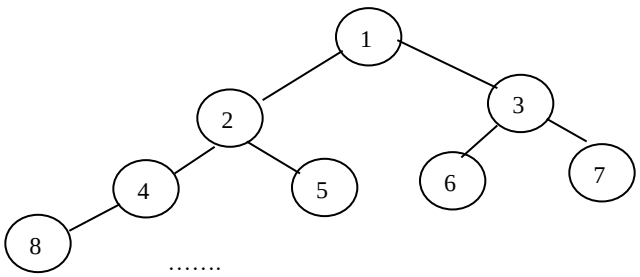
Using this basic architecture design the Concurrent-Server system described above allows as many pairs of nodes (from both clusters) as possible to communicate content data. *The computational problem to be solved as services provided by the server-side cores/threads are described in the following*.

Experimental Design

The clients are to generate matrices of data (integers) to be multiplied using a parallel algorithm technique called Matrix-Chain-Multiplication (MCM) based on Strassens' algorithm (https://www.tutorialspoint.com/data_structures_algorithms/strassens_matrix_multiplication_algorithm.htm). (This URL lists a program that implements the Strassens' algorithm, and you can use it to simplify the effort, or modify it using your preferred language.)

First, transmit each matrix, M, from the client to the server, where the number of matrices to be multiplied per request is a power of 2; and the dimensions are also a power of 2. Second, develop a binary tree with the leave nodes representing the matrices. (See below.) Thirdly, assign the leave-nodal matrices, two for each threads/cores (acting as parent nodes) to perform the M x M multiplications using the Strassen's method. Continue to propagate the multiplication subproducts towards the root of the imaginary binary tree, where the root acts as a single collector-core that collects the final product, the timing data, etc.

The server side threads/cores multiply the pairs of matrices using p = 1, 3, 7, 15 and 31 threads/cores for 2 matrices, 4 matrices, 8 matrices, 16 matrices, and 32 matrices, respectively). Run multiple test data or matrices of varying dimensions *N x N*: 1k x 1k, 2k x 2k, 4k x 4k, 8k x 8k and 16k x 16k, between the client side and the server side as discussed and depicted in the (partial) binary tree below. Generate three tables from the timing data *collected from each run* for each of the performance metrics – Parallel Execution Time, Speed Up, and Efficiency.



A Binary Tree Model of the MCM Technique

Strassens' Algorithm: For this implementation, the server-side multiplication will work by using multithreading, where each matrix of integers will be decomposed into equal subsets of N/2 x N/2 data points, multiplied in parallel, and then the subproducts are percolated up the binary tree for final multiplication (by the root). The idea is to use the server as a multi-core device to achieve faster computation, which will be evident in the performance outcomes. Read more about the Strassens' algorithm, which has a time complexity of $O(n^{log7})$ from the literature.

Overall Objective: Use the binary tree model and Strassens' algorithm to help guide your team in this 'awesome' hands-on exercise. (Note: We are exploring the neat things about SPMD parallel computing paradigm and Client-Server distributed computing paradigm to affirm or refute the theories and concepts we are learning in the course. So, fully participate in the exercise to benefit from it. Enjoy it!)

Run your code for twenty-five (25) scenarios based on the following table, resulting in 25 sets of timing data.

#Cores (#Matrices)	1 (2 M's)	3 (4 M's)	7 (8 M's)	15 (16 M's)	31 (32 M's)
# of Matrix					
1k x 1k					
2k x 2k					
4k x 4k					
8k x 8k					
16k x 16k					

Tabulating the Simulation Data

The first table is to be filled in with timing data collected for each scenario. Then, the remaining tables are to be completed using the appropriate formulae for the speed-up and efficiency metrics, as studied in the beginning of the semester. (Do not manually compute the speed up and efficiency metrics, but rather have them computed in your code.)

Run or Execution Times	*Matrix Sizes*				
No of Threads (cores)	*1k x 1k*	*2k x 2k*	*4k x 4k*	*8k x 84*	*16k by 16k*
p = 1					
p = 3					
p = 7					
p = 15					
p = 31					

Speed Up	*Matrix Sizes*				
No of Threads (cores)	*1k x 1k*	*2k x 2k*	*4k x 4k*	*8k x 84*	*16k by 16k*
p = 1					
p = 3					
p = 7					
p = 15					
p = 31					

Efficiency	*Matrix Sizes*				
No of Threads (cores)	*1k x 1k*	*2k x 2k*	*4k x 4k*	*8k x 84*	*16k by 16k*
p = 1					
p = 3					
p = 7					
p = 15					
p = 31					

C. What To Turn In

Part 1: A Progress Report is to be submitted for the first step of successfully implementing the communication between one **client** process and one **server** process, via the Sever-Router. In this scenario, a simple **lower-cased** message/text is to be sent from the client to the server for converting the text into **uppercased** message/text and returned to the client. Run this part for a reasonable number of times, using a varying length of texts/messages and a varying number of client and server processes for each run. This part will accomplish the C-S paradigm proof-of-concept implementation. This Progress Report of Part1 will be simple; it must include a **5-10-page** narrative of how your team a) modified the given Java code to achieve this step, b) the tables of data: bytes of text, tabulated timing data and associated graphs, and c) discussions of observations, and d) conclusion. There will be no need for an .mp4 demo file; just the IEEE-style formatted Report, which must include all the sections as outlined in the Rubrics file.

Part 2: The outcome of this part will be the full implementation of the MCM problem as described above. Thus, modify the Java programs by replacing the lowercased text/message with a series of matrix data from the client processes to the server threaded processes to mimic parallel computations/multiplications of the matrices as described above. Note that the Strassens' code will now replace the 'uppercased' method to implement the matrix-matrix multiplications. This part will culminate in the Final Report, the associated (modified) source code files, and the .mp4 demo file for submission towards the end the semester. [*Note: the four (4) Java programs are offered to initiate and simplify the project effort. Your task is to modify them to work accordingly to suit the project objectives.*]

For Part 2 Reports, write a **10–15-page** report of your findings, in the form of clear discussions that include 1) completed tables of the 3 sets of 25 timing data for the three metrics – parallel runtimes, speedups, efficiency, 2) their corresponding graphs and 3) the analysis and observations/comments. The body of the report must include (a) snippets of code (in the Design Implementation section) including the formulae you used in calculating the performance metrics; (b) example screenshots of the simulation run; and (c) captions for all Figures and Tables, with discussions. Review the Rubric again and pay attention to each 'section'.

Make a video-capture (an .mp4 file) of a 3-5-minute demo, including a screen-capture of the compiling and running phases of your simulation. ***Attach [this] .mp4 file, the source code(s), and a .docx version of the Report in your submission. Don't zip them. The Report structure should follow the IEEE-style 2-column format/sample given to you in the PROJECT module in D2L***.