Week 4 Discussion Questions
Chapter 2.3 - 2.4:

1. We used Flynn’s taxonomy to identify three types of parallel systems: SISD, SIMD, and MIMD. How might a MISD (multiple Instruction, single data) work? (Give an example if possible)

2. The TOP500 list (http://www.top500.org/) provides a ranked list of the most powerful general purpose supercomputers systems. It also provides “historical data” at http://www.top500.org/statistics/overtime/
If you change the category to “Architecture” and hit the Submit button, you can see a graph for the last 10 years.
a) What architectures dominate the past 10 years and why?

b) What is meant by the architecture “MPP”?

At the top of the graph is a gray “slider” in the middle of the graph. Slide it to the left all the way to expand the time frame another 10 years or so. If you position the mouse pointer along the time-line at the bottom of the graph you can see data values for each architecture at specific dates.

c) What do you observer about the Single Processor and SIMD architecture types?

d) What is meant by the architecture “Constellations”?

e) Between Nov 01, 1998 and Nov 01, 2000 the SMP % drastically decreased while the Constellations % drastically increased. What do you supposed happened to explain this?

3. If you change the category to “Interconnect Family” and hit the Submit button, you can see a graph for the last 10 years.
a) What Interconnection families have dominate the past 10 years and why?

4. A common Snooping cache-coherence scheme used on multi-core PCs is the MESI protocol. Some helpful links are at:
a) What is the main difference between the M and E states?

b) How does the M state reduce accesses to the shared main memory?

c) How does the E state reduce accesses to the shared main memory?
Learning Objectives:
- Write correct programs that dynamically allocate and use 2D arrays.
- Write correct programs that embeds a 2D array into a 1D array.
- Write correct programs that read and write text files.

To start the lab:
- watch the Lab 4 Video on the eLearning system
- download lab4.zip from the eLearning system and unzip/extract it locally on your computer
- copy the lab4 directory to student.cs.uni.edu using a secure ftp client (winSCP, FileZilla, scp, etc.)
- log-on to student.cs.uni.edu using Putty/ssh

Part A: Using an editor on student.cs.uni.edu open the file cmdLineMultTable.c which contains a simple C program that allows the user to enter two integers on the command-line, and prints the corresponding multiplication table.

Answer the following questions about the cmdLineMultTable.c program:
a) What is the maximum size of value1 and value2 that the program is designed for?

b) Compile (gcc -o table cmdLineMultTable.c) and run the program with values 15 and 10 (./table 15 10). Did it produce the correct results?

c) Re-run the program with values 150 and 100 (./table 150 100). Explain the results.

Notice that the multiplication table is stored in a statically declared 2D array defined in the main as:
```c
int multiplicationTable[SIZE][SIZE];
```
The main passes multiplicationTable to the calculateProducts function to be “filled” with the call:
```c
calculateProducts(value1, value2, multiplicationTable);
```
d) Why is & not needed before the multiplicationTable parameter eventhough it “returns” a changed value?

The calculateProducts function definition is:
```c
void calculateProducts(int rows, int columns, int multiplicationTable[][SIZE])
```
Note: C requires sizes for all, but the first dimension of arrays to be specified. This is a holdover of when multi-dimensional arrays were stored as a single contiguous chuck of memory:

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>[0][0]</th>
<th>[0][1]</th>
<th>[0][2]</th>
<th>[1][0]</th>
<th>[1][1]</th>
<th>[1][2]</th>
<th>[2][0]</th>
<th>[2][1]</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Row 0</td>
<td>Row 1</td>
<td>Row 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Thus, address of multiplicationTable[r][c] = starting address of array + (r * SIZE + c) * (size of an element).
However, C now allocates two-dimensional arrays as a series of one-dimensional arrays:

Compile `printAddr.c (gcc -o addr printAddr.c)` and run the program with values 4 and 4 (`./addr 4 4`) which is the same as the `cmdLineMultiTable.c` program, except it also prints the addresses of each array element.

e) How many bytes is used to store each integer product?

f) How many bytes are used to store each row (recall that SIZE is 20)?

g) How big of a "gap" it there between the end of one row and the start of another?

**Part B:** Using an editor open `printAddrDyn.c` which contains a similar C program to print a multiplication table. Notice that the multiplication table storage is NOT allocated when the `main` starts, instead only the pointer:

```c
int ** multiplicationTable;
```

The `main` assigns `multiplicationTable` a pointer value to a dynamically allocated 2D array by:

```c
multiplicationTable = allocate2DArray(value1, value2);
```

Answer the following questions about the `allocate2DArray` function:

a) What is the maximum size of `value1` and `value2` that the program is designed for?

b) How much "wasted" storage space is allocated for storing integer products using dynamically allocated 2D array?

c) Explain each part of the assignment state in the `allocate2DArray` function:

```c
local2DArray = (int **) malloc(sizeof(int *) * rows);
```
d) What is the purpose of the loop in the allocate2DArray function?
   
   ```c
   for (r=0; r < rows; r++) {
       local2DArray[r] = (int *) malloc(sizeof(int)*columns);
   } // end for
   ```

   Compile `printAddrsDyn.c (gcc -o addrDyn printAddrsDyn.c)` and run the program with values 4 and 4
   (`./addrDyn 4 4`) which is the same as the `printAddrs.c` program, except it prints the pointer address contained
   in `multiplicationTable`, each element (i.e., a pointer addresses to the start of each 1-D array for a row) in
   the 1-D array pointed at by `multiplicationTable`, and the `address` of each element.

   e) How many byte are between each element in the 1-D array pointed at by `multiplicationTable`?

f) To practice passing 2-D arrays as parameters, modify the `printAddrsDyn.c` program so that
   `allocate2DArray` is passed the `multiplicationTable` as a parameter which gets modified inside of
   `allocate2DArray`, instead of the `allocate2DArray` returning a `int **` value. Re-compile and re-run
   the program to make sure that it still works.

**Part C:** Sometimes in this course we will want to explicitly embed a 2D array inside a 1D array as in the first
picture (p. 1). Typically, we do this if we want to send a 2D array as a single message, or copy a 2D array easily.
Open the `multTable2Din1D.c` program which prints a multiplication table using a single 1D array containing an
embedded 2D array.

a) How did the `allocate2DArray` function change?

b) In the `calculateProducts` function explain the pointer arithmetic on the left-hand side of the assignment:
   
   ```c
   *(multiplicationTable+r*columns+c) = (r+1) * (c+1);
   ```

c) An alternate way of accessing the 1D array is to "walk a pointer" down it. Comment out the above assignment
statement and uncomment the alternate using the `nextElementPtr`. Re-compile and re-run the code to see that
it works. This alternative uses the pointer and then "post-increments" it (the `++` operator after the pointer name).
How does the `++` operator know how much to increment the pointer by?

d) Modify the `printRow` function so that it also walks a pointer down the array. Re-compile and re-run the code
to see that it works.
Part D: Open the file fileMultiTable.c which allows the user to enter two integers and a text-file name on the command-line, and prints the corresponding multiplication table to the specified text file.

In C as in most languages, to use a file you need to:

1) Open the file for reading or writing - connects the program to the file
2) Read or Write to the file
3) Close the file - if the file was opened for writing, the operating system often buffers the writes in main memory because the disk is so slow. Closing the file flushes the writes to disk.

a) An open file is C use a "file pointer" variable of type: FILE *. What is the variable name of the file pointer used in this program?

b) What statement(s) cause the file to open?

c) The printTableHeading and printRow functions write to the text-file using fprintf statements. What's different about the parameters to the fprintf statement vs. the printf statement?

d) What statement(s) cause the file to be closed?

e) To read text-input from the keyboard scanf is used, to read input from a text-file what statement (and parameters) do you think would be used?

Submit lab4.zip containing question answers and completed programs on the eLearning system
Learning Objectives:
- Contrast the architectural differences between shared-memory vs. distributed-memory machines.
- Compare the general characteristics of interconnection networks.
- Explain the snoopying bus cache coherence scheme.

1. For a 64 processor system, compare the interconnection network for each of the following topologies. (We normalize the bandwidth of a single link to “1”).

<table>
<thead>
<tr>
<th></th>
<th>Bus</th>
<th>Ring</th>
<th>2-d Torus</th>
<th>6-d Hypercube</th>
<th>Fully Connected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total # of Switches</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Links per Switch</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total # of links</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Bandwidth</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bisection Bandwidth</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. The above table focuses on the overall characteristics of different interconnection networks. If we focus on a single data transmission of \( n \) bytes between two processors (the source and destination), then transmission time is effected by:

- latency \( (l) \) - the time that elapses between the source’s beginning to transmit the data and the destination’s receiving the first byte of data.
- bandwidth \( (b) \) - the rate at which the destination receives data after it has started to receive the first byte (i.e., \( b \) B/sec.)

a) What is the formula for transmitting an \( n \) bytes message between a source and destination with a bandwidth of \( b \) B/second?

   \[
   \text{message transmission time} = \frac{n}{b}
   \]

b) What components in the above table effect the latency?

c) What components in the above table effect the bandwidth?

3. Textbook exercise 2.10 on page 78.

4. Textbook exercise 2.15 on page 79.
2.10. Suppose a program must execute $10^{12}$ instructions in order to solve a particular problem. Suppose further that a single processor system can solve the problem in $10^6$ seconds (about 11.6 days). So, on average, the single processor system executes $10^6$ or a million instructions per second. Now suppose that the program has been parallelized for execution on a distributed-memory system. Suppose also that if the parallel program uses $p$ processors, each processor will execute $10^{12}/p$ instructions and each processor must send $10^9(p-1)$ messages. Finally, suppose that there is no additional overhead in executing the parallel program. That is, the program will complete after each processor has executed all of its instructions and sent all of its messages, and there won’t be any delays due to things such as waiting for messages.

a. Suppose it takes $10^{-9}$ seconds to send a message. How long will it take the program to run with 1000 processors, if each processor is as fast as the single processor on which the serial program was run?

b. Suppose it takes $10^{-3}$ seconds to send a message. How long will it take the program to run with 1000 processors?

2.15. a. Suppose a shared-memory system uses snooping cache coherence and write-back caches. Also suppose that core 0 has the variable $x$ in its cache, and it executes the assignment $x = 5$. Finally suppose that core 1 doesn’t have $x$ in its cache, and after core 0’s update to $x$, core 1 tries to execute $y = x$. What value will be assigned to $y$? Why?

b. Suppose that the shared-memory system in the previous part uses a directory-based protocol. What value will be assigned to $y$? Why?

c. Can you suggest how any problems you found in the first two parts might be solved?
2D arrays

```
M = [0 1 2 3 4 5]
```

```
5 rows
```

```
[1 2 3 4 5] = 5
```

```
int * M[5][6]
```

```
int ** M;
M = (int**) malloc(sizeof(int*) * 5)
```

```
for (i = 0; i < 5; i++) {
    M[i] = (int*) malloc(sizeof(int) * 6)
}
```
$M_{[i]} = \sum_{j} \text{column}_j^i$

where $\frac{1}{2} \leq \text{column}_j^i \leq 1$
Comp. Arch.  Week 4 Monday

Interconnection Network - Effects performance on both distributed and shared memory systems

Distributed Memory Interconnects:

Direct Interconnect:

Termimology:

Bandwidth - rate at which a link can transmit data (e.g., megabits / second)

Bisection bandwidth - a measure of the network quality which sums the bandwidth connecting halves of the processors.

The transmission time of single data transmission of $n$ bytes between two processors (the source and destination) is effected by:

- latency ($l$) - the time that elapses between the source’s beginning to transmit the data and the destination’s receiving the first byte of data,
- bandwidth ($b$) - the rate at which the destination receives data after it has started to receive the first byte

message transmission time = $l + n / b$
Generic Indirect Interconnection Network: Processors (boxes on left) have an in-coming and an out-going link from and to the switching network.

Crossbar interconnection:

Omega network:
An Omega switch
Comp. Arch.

Cache Coherence Solution - bus watching with write through / Snoopy caches - caches eavesdrop on the bus for other caches write requests. If the cache contains a block written by another cache, it take some action such as invalidating it's cache copy.

"system bus"

The MESI protocol is a common write-back cache-coherency protocol. Each cache line is marked as: Modified, Exclusive, Shared or Invalid.

<table>
<thead>
<tr>
<th></th>
<th>Modified</th>
<th>Exclusive</th>
<th>Shared</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>This cache line valid?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>The memory copy is</td>
<td>out of date</td>
<td>valid</td>
<td>valid</td>
<td></td>
</tr>
<tr>
<td>Copies exist in other</td>
<td>No</td>
<td>No</td>
<td>Maybe</td>
<td></td>
</tr>
<tr>
<td>caches?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A write to this line</td>
<td>does not go to the bus</td>
<td>does not go to the bus, but change to M</td>
<td>goes to the bus and update cache</td>
<td>goes directly to bus</td>
</tr>
</tbody>
</table>

3. How can distributed shared memory machines do cache coherency? - directory protocol
Comp. Arch.

Week 4 Monday

Interconnection Network - Effects performance on both distributed and shared memory systems

Distributed Memory Interconnects:

Direct Interconnect:

- **Switch**

- **Ring**

- **2D Toroidal Mesh**

- **Fully Connected**

- **3D Torus Mesh**

Terminology:

*Bandwidth* - rate at which a link can transmit data (e.g., megabits / second)

*Bisection bandwidth* - a measure of the network quality which sums the bandwidth connecting halves of the processors.

The transmission time of single data transmission of $n$ bytes between two processors (the source and destination) is affected by:

- **Latency ($l$)** - the time that elapses between the source's beginning to transmit the data and the destination's receiving the first byte of data.
- **Bandwidth ($b$)** - the rate at which the destination receives data after it has started to receive the first byte

\[ \text{message transmission time} = l + n / b \]
Generic Indirect Interconnection Network: Processors (boxes on left) have an in-coming and an out-going link from and to the switching network.

Crossbar interconnection:

Omega network:
An Omega switch
1. What are the main motivation(s) of writing a parallel program?

We can categorize sources of overhead in parallel programs that limit speedup as follows:
- interprocess communication/interaction - processors need to communicate data (i.e., intermediate results)
- idle processors - processors can be idle for a variety of reasons:
  - load imbalance - a processor is assigned less work than others so it sits idle waiting of others to finish
  - synchronization - processor need to to coordinate their operations (e.g., barrier synchronization) so processors done sooner must wait for others to complete
  - non-parallelizable computation/task - some serial component that cannot be done in parallel.  
Amdahl’s law applies here.
- parallelization overhead - additional costs in the parallel solution that are not in the sequential computation

2. For each of the following scenarios identify the type of overhead(s) that occur.

a) threads doing extra computation to determine which part of the parallel computation they need to perform

b) parallel computation is unevenly distributed to processors so some finish before others

c) a spin lock in which a waiting thread repeated checks for the availability of a lock on a shared variable

d) thread/process setup and teardown time when a thread/process is created and later destroyed

e) Sequential computation performed redundantly across all processors
Learning Objectives:
- Utilize the random number generator in C to generate 2D arrays.
- Write code to perform the matrix multiplication calculation efficiently.
- Write correct programs that read and write binary files.

To start the lab:
- watch the Lab 5 Video on the eLearning system
- download lab5.zip from the eLearning system and unzip/extract it locally on your computer
- copy the lab5 directory to student.cs.uni.edu using a secure ftp client (winSCP, FileZilla, scp, etc.)
- log-on to student.cs.uni.edu using Putty/ssh

Part A: Using an editor on student.cs.uni.edu open the file writeRandom2DArray.c which contains a simple C program that allows the user to enter two integers (# rows and # columns), a binary file name, and two reals (min. random # and max. random # range) on the command-line. The program writes to the specified file name binary data consisting of:
- integer number of rows
- integer number of columns
- the randomly generated 2D array of doubles in row-major order

Answer the following questions about the writeRandom2DArray.c program:
a) How does the creation (opening) of the binary file differ than the creation of the text file in lab4?

Consider the The generateRandom2DArray function defined as:

```c
void generateRandom2DArray(int rows, int columns,
                           double min, double max, double **random2DArray) {
    int r, c;
    double range, div;

    // seed the random number generator
    srand( time(NULL) );
    for (r = 0; r < rows; r++) {  
        for (c = 0; c < columns; c++) {
            range = max - min;
            div = RAND_MAX / range;
            random2DArray[r][c] = min + (rand() / div);
        } // end for (c...  
    } // end for (r...  
} // end generateRandom2DArray
```

b) By seeding the random number generator using the current time in seconds (i.e., srand( time(NULL) )); then we should mostly get a different randomly generated 2D array everytime we run the program. What problem might occur if we called the generateRandom2DArray function twice within the same program to generate two difference 2D arrays, say A and B?

c) How could we fix the above problem?

d) Explain each of the three assignment statements what generate a random array element.
**Part B:** Since the output file generated is binary (i.e., not a text file), we cannot just open it in an editor to verify its correctness. To verify its correctness, un-comment the main program and write code to complete the functions:

- **read2DArray** - passed a binary file pointer opened for reading and returns the # rows, the # columns, and the 2D array of doubles. You'll need to use the fread function to read the contents of the binary file -- see the tutorial on files at: [http://www.cprogramming.com/tutorial/cfileio.html](http://www.cprogramming.com/tutorial/cfileio.html)
- **equal2DArray** - passed the # rows, # columns, two 2D arrays, and a tolerance. It returns TRUE if corresponding array elements are equal within the specified tolerance; otherwise it returns FALSE.

Re-compile and run the program with values `.\write2D 5 7 myFile.dat 5.0 9.0` to make sure that it works.

**Part C:** Matrix Multiplication is a frequently used numeric calculation that takes two matrices (i.e. 2D arrays) A (m rows x q columns) and matrix B (q rows x n columns) and produces matrix C (m x n), where \( c_{ij} \) (i.e., \( C[i][j] \)) is the dot product of the \( i^{th} \) row of A with the \( j^{th} \) column of B. In other words,

\[
q - 1 \\
\sum_{k=0}^{q-1} a_{ik} * b_{kj}
\]

For example:

\[
\begin{pmatrix}
2 & 3 & 1 \\
0 & 2 & 1 \\
2 & 2 & 1
\end{pmatrix}
\times
\begin{pmatrix}
2 & 2 \\
1 & 0 \\
2 & 1
\end{pmatrix}
=
\begin{pmatrix}
9 & 5 \\
4 & 1 \\
8 & 5
\end{pmatrix}
\]

The sequential algorithm for matrix multiplication is:

```c
for i = 0 to m-1 do
    for j = 0 to n-1 do
        c_{ij} = 0
        for k = 0 to q-1 do
            c_{ij} = c_{ij} + a_{ik} * b_{kj}
        end for k
    end for j
end for i
```

Using an editor on `student.cs.uni.edu` open the file `mmultSeqOptions.c` which contains a program that allows the user to enter an integer (# rows and # columns of square matrices A, B, and C) on the command-line, and times the calculation of the matrix multiplication of randomly generated A and B matrices. It actually calculates the product matrix C as described above (i.e., `matrixMultiplication` function) and product matrix C_alt using a slight variation by `matrixMultiplicationAlt` function.

Answer the following questions about the `mmultSeqOptions.c` program:

a) How is the time to perform the `matrixMultiplication` function determined?
b) Compile (gcc -o mmult mmultSeqOptions.c -lm) and run the program (e.g., ./mmult 1000) several times to complete the below table:

<table>
<thead>
<tr>
<th>Matrix Sizes</th>
<th>Time of <code>matrixMultiplication</code> function (in seconds)</th>
<th>Time of <code>matrixMultiplicationAlt</code> function (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500 x 500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>750 x 750</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000 x 1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1500 x 1500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000 x 2000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2500 x 2500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000 x 3000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

c) Explain why the `matrixMultiplicationAlt` function is faster than the `matrixMultiplication` function.

Submit lab5.zip containing question answers and completed programs on the eLearning system.
Learning Objectives:

- List sources of parallelization overhead
- Distinguish between task vs. data parallelism
- Apply Foster’s methodology for parallel program design to produce a design that minimizes parallel overhead.
- Apply Amdahl’s law to calculate the maximum speedup given a fraction of serial processing.
- Calculate parallel performance metrics: speedup and efficiency

1. Two types of parallelism:
   - task parallelism - split program into major tasks and solve as many tasks in parallel as possible
   - data parallelism - partition the data across the processors with each processor doing the same type of calculations on their own chunk of data

a) Which of the above approaches is more scalable (can utilize more processors) as the problems get large (i.e., lots of data)?

b) How might a combination of the two be used?

2. Let’s think about parallelism, but with real-world examples. Here think of each person as a processor. For each example, determine the “major” sequence of tasks/steps, but also explain what tasks can be “data parallelized” assuming we had many people?

a) Building a house.

b) Consider putting together a 5,000-piece jiggle-saw puzzle with different number of people:
   - 2 people
   - 5 people
   - 25 people
   - 100 people

c) Preparing food for a large (1,000 people) banquet with each meal consisting of salad, entree, side-dish, dessert
3. Chapter 2. Exercise 2.16 on page 79.

2.16. a. Suppose the run-time of a serial program is given by $T_{\text{serial}} = n^2$, where the units of the run-time are in microseconds. Suppose that a parallelization of this program has run-time $T_{\text{parallel}} = \frac{n^2}{p} + \log_2(p)$. Write a program that finds the speedups and efficiencies of this program for various values of $n$ and $p$. Run your program with $n = 10, 20, 40, \ldots, 320$, and $p = 1, 2, 4, \ldots, 128$. What happens to the speedups and efficiencies as $p$ is increased and $n$ is held fixed? What happens when $p$ is fixed and $n$ is increased?

b. Suppose that $T_{\text{parallel}} = \frac{T_{\text{serial}}}{p} + T_{\text{overhead}}$. Also suppose that we fix $p$ and increase the problem size.
   - Show that if $T_{\text{overhead}}$ grows more slowly than $T_{\text{serial}}$, the parallel efficiency will increase as we increase the problem size.
   - Show that if, on the other hand, $T_{\text{overhead}}$ grows faster than $T_{\text{serial}}$, the parallel efficiency will decrease as we increase the problem size.


2.23. In our application of Foster's methodology to the construction of a histogram, we essentially identified aggregate tasks with elements of data. An apparent alternative would be to identify aggregate tasks with elements of bin_counts, so an aggregate task would consist of all increments of bin_counts[b] and consequently all calls to Find_bin that return b. Explain why this aggregation might be a problem.
5. There is a "paradigm shift" making parallel programming conceptually different from sequential programming. A simple example is summing an array `x` containing `n` elements.

**Sequential Algorithm:**

```c
sum = 0;
for (i = 0; i < n; i++) {
    sum = sum + x[i];
} // end for
```

**Parallel Pair-Wise Summation Algorithm**

![Diagram showing parallel summation process]

a) How long would each algorithm take?
b) How many processors does the pair-wise summation algorithm utilize?

d) Which of these can you identify in the above "parallel pair-wise summation" algorithm?
(1) Task parallelization - big tasks

(2) Data parallelization, block size = \frac{\text{data count}}{P}

\[ x_0 \quad x_1 \quad x_2 \quad x_3 \]

barrier synchh.
Shared memory

Client-server architecture

Master thread \rightarrow \text{requests for work (over Internet)}

Worker thread
perform request

- disadvantage - overhead in dynamic creating threads

Pool of static threads
idle/sleeping

Master thread \rightarrow \text{request for work}
1. Consider the following table comparing different interconnection topologies for a varying number of processors.

<table>
<thead>
<tr>
<th>Topology</th>
<th>Characteristic</th>
<th>16 Processors</th>
<th>64 Processors</th>
<th>256 Processors</th>
<th>1024 Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-D Torus</td>
<td>Arrangement</td>
<td>4 x 4 grid</td>
<td>8 x 8 grid</td>
<td>16 x 16 grid</td>
<td>32 x 32 grid</td>
</tr>
<tr>
<td></td>
<td>Links per switch at each processor</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Total # of links</td>
<td>32</td>
<td>128</td>
<td>512</td>
<td>2,048</td>
</tr>
<tr>
<td></td>
<td>Bisection Bandwidth</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>Hypercube</td>
<td>Arrangement</td>
<td>4-D hypercube</td>
<td>6-D hypercube</td>
<td>8-D hypercube</td>
<td>10-D hypercube</td>
</tr>
<tr>
<td></td>
<td>Links per switch at each processor</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Total # of links</td>
<td>32</td>
<td>192</td>
<td>1,024</td>
<td>5,120</td>
</tr>
<tr>
<td></td>
<td>Bisection Bandwidth</td>
<td>8</td>
<td>32</td>
<td>128</td>
<td>512</td>
</tr>
<tr>
<td>Fully</td>
<td>Arrangement</td>
<td>15</td>
<td>63</td>
<td>255</td>
<td>1,023</td>
</tr>
<tr>
<td>Connected</td>
<td>Links per switch at each processor</td>
<td>120</td>
<td>2,016</td>
<td>32,640</td>
<td>523,776</td>
</tr>
<tr>
<td></td>
<td>Total # of links</td>
<td>64</td>
<td>1,024</td>
<td>16,384</td>
<td>262,144</td>
</tr>
<tr>
<td></td>
<td>Bisection Bandwidth</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a) Why is a fully-connected topology impractical?

b) Explain why 2-D Torus topology scales better than the hypercube topology as the number of processors increases.

2. Many shared-memory computers use a snoopy-bus protocol to solve cache-coherence (e.g., MESI protocol with cache line states of Modified, Exclusive, Shared or Invalid).

![Diagram](image)

a) If a shared variable X is contained in core 0’s cache and core 1’s cache both in the Shared state, what happens when core 0 writes 6 to X?

b) Why does cache-coherence alone not solve the problem of two threads performing the high-level language statement \( X = X + 1 \) in parallel?
3. Consider building a fence around 3 sides of a backyard with each side being 50' long and the back is 100' wide. The fence will have posts 10' apart with two horizontal boards between adjacent pairs of posts. Assume the following:

- you have access to 4 motorized post-hole diggers
- each motorized post-hole diggers require two people to operate and can dig a hole in 10 minutes
- two people can install the vertical boards between two adjacent posts in 5 minutes

a) How much “work” time (i.e., not counting breaks) would it take one pair of people to complete the fence?

b) How much “work” time would it take 4 pairs of people to complete the fence?

c) What is the speedup and efficiency of using 4 pairs of people over a single pair of people?

\[
\text{speedup} = \frac{\text{completion time of a single pair}}{\text{completion time of 4 pairs}}
\]

\[
\text{efficiency} = \frac{\text{speedup}}{(\# \text{ of pairs})}
\]

d) Identify any overhead/inefficiencies of using 8 pairs of people to build the fence.