Instructions.

1. Attempt all questions. Partial credit will be given.
2. Show all the steps of your work clearly.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Your Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Extra Credit</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
[Q1] (10 points):

(a) Answer the following questions using the images below:

(i) What type of computer systems are these? (1 point)  
Parallel or cluster computer systems.

(ii) Which of the machines above is a shared memory architecture? (1 point)  
Machine A, machine on left.

(iii) Describe the term "SPMD" as used in Flynn's taxonomy: (1 point)  
Single Program Multiple Data: multiple processors simultaneously execute the same program at independent points.

(iv) Define OpenMP: (1 point)  
OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran ... (Wikipedia).

(v) On which machine would OpenMP be used? (1 point)  
Machine A.

(b) Describe the terms used in In Fosters' "PCAM" Methodology:

(i) Mapping (1 point):  
Each task is assigned to a processor in a manner that attempts to satisfy the competing goals of maximizing processor utilization and minimizing communication costs.

(ii) Agglomeration (1 point):  
The task and communication structures defined in the first two stages of a design are evaluated with respect to performance requirements and implementation costs. If necessary, tasks are combined into larger tasks to improve performance or to reduce development costs.

(iii) Partitioning (1 point):  
The process of decomposing a computation that is to be performed and the data to be operated on into small tasks.

(iv) Communication (1 point):  
The communication required to coordinate task execution is determined, and appropriate communication structures and algorithms are defined.
[Q2] (15 points):
For the questions below, consider the case of a 1D finite difference problem (FD), in which there is a vector, X, of size N, which is defined as X^{(0)} at timestep t = 0, and code below which computes the vector X^{T}, where T is the total number of timesteps.

\[
X^{(t+1)}_i = \frac{X^{(t)}_{i-1} + 2X^{(t)}_i + X^{(t)}_{i+1}}{4}
\]

1D stencil.

(a) (2 points): Identify the tasks in the figure on the left and describe why they are tasks.
N=8. The tasks are defined by the elements in X. The tasks perform the work of addition and division, as shown in the equation above.

(b) (2 points): Identify the communication pattern in the figure on the left, and describe how it is determined.
The communication pattern (defined by the central difference stencil above) is shown in the bottom two figures on the left. Each element, X_i sends its data to elements on its left and right, and receives data from those neighbors.

(c) (1 points): What is the value for N?

(d) (10 points): Assume a vector, xold, is populated as shown below, using N = 4, T = 2, and xold = X^t and xnew = X^{t+1} as defined above, what is value of xnew after T time-steps?

```c
/* Initialize xold at time step t=0 */
for( i=0; i<=N-1; i++ )
    xold[i] = 2.0*i; /* x at t=0 */

/* update x at each time step*/
for( t=1; t<=T-1; t++ ) {
    /* compute border/edge cells */
    xnew[0]=(xold[0]+xold[1])/2
    xnew[N-1]=(xold[N-1]+xold[N-2])/2

    /* compute inner cells */
    for( i=1; i<N-1; i++ ) {
        xnew[i] = xold[i-1] + 2*xold[i] + xold[i+1]
    }

    xold = xnew;
}
```
Grading notes:

• 5/10 for setting up some attempt to compute / show array values at different stages
• add points when the work looks correct

```c
#include <stdio.h>
#include <math.h>
const int q=4, r=5, s=2;
const int N=4, T=2;
void main() {
    int i, j, k, t;
    double xold[N], xnew[N];
    for (i = 0; i < N-1; i++)
        xold[i] = (double)i;
    //Calculate the function
    printf("N=%d, T=%d \n", N, T);

    /* Initialize xold at time step t=0 */
    for( i = 0; i <=N-1; i++ )
        xold[i] = 2.0*i; /* x at t=0 */
    printf(" x_start[0,1,2,3]=\n");
    for( i = 0; i <=N-1; i++ )
        printf("%2.1f, ",xold[i]);
    printf("\n");

    /* update x at each time step*/
    for( t = 1; t<=T-1; t++ ) {
        /* compute border/edge cells */
        xnew[0]=(xold[0]+xold[1])/2;
        xnew[N-1]=(xold[N-1]+xold[N-2])/2;
        /* compute inner cells */
        for( i = 1; i<N-1; i++ ) {
            xnew[i] = xold[i-1] + 2*xold[i] + xold[i+1];
        }
        for( i = 0; i <=N-1; i++ )
            xold[i] = xnew[i];
    }
    printf(" x_end[0,1,2,3]=\n");
    for( i = 0; i <=N-1; i++ )
        printf("%2.1f, ",xold[i]);
    printf("\n");
}
```

`[gidget:quizzes/q1/dev] mthomas% !gcc
 gcc -W-o q1_q2 q1_q2.c
 [gidget:quizzes/q1/dev] mthomas% ./q1_q2
 N=4, T=2

 x_start[0,1,2,3] = [0, 2.0, 4.0, 6.0]
 x_end[0,1,2,3] = [1.0, 8.0, 16.0, 5.0]

 if using 1/4:
 x_end[0,1,2,3] = [1.0, 2.0, 4.0, 5.0]`
[Q3] (5 points):
The equation below is used to estimate the run-time of a computer program. Define and describe the terms in this equation. Grading Comment: 1 point each, everyone gets $T_{\text{overhead}}$:

$$T_{\text{wall}} = T_{\text{CPU}} + T_{I/O} + T_{\text{Idle}} + T_{\text{overhead}}$$

(a) $T_{\text{wall}}$: The total (or real) time that has elapsed from the start to the completion of a computer program or task.

(b) $T_{\text{CPU}}$: The amount of time for which a central processing unit (CPU) is used for processing instructions of a computer program or operating system.

(c) $T_{I/O}$: The time spent by a computer program reading/writing data to/from files such as /STDIN/STDERR, local data files, remote data services or databases.

(d) $T_{\text{Idle}}$: The time spent by a computer program waiting for execution instructions.

(e) $T_{\text{overhead}}$: The amount of time required to set up a computer program including setting up hardware, local and remote data and resources, network connections, messages.

[Q4] (5 points):
For a parallel cluster, the communication networks are customized, high-end systems. Describe the following terms.

(a) Latency:
The time that elapses between the sources beginning to transmit the data and the destinations starting to receive the first byte.

(b) Bandwidth:
The rate at which the destination receives data after it has started to receive the first byte.

(c) For a data file of size of 2 TBytes, how long would it take to transfer your data to an archival machine using the following networks (assume physical distance is not relevant):

$$T_{\text{message}} = \frac{\text{SizeOfData}}{\text{BW}} = \frac{\#\text{Bits}}{(\#\text{Bits/sec})}$$

(i) Time Warner Turbo Internet (home): 2 Mbits/s upload:

$$T_{\text{message}} = \frac{2\text{TBytes}}{2\text{Mbits/s}} = \frac{(2\times 10^{12}) \text{ bits}}{2 \times 10^6 \text{ bits/sec}(8 \text{bits/Byte})} = \frac{(16 \times 10^{12}) \text{ bits}}{(2 \times 10^6 \text{ bits/s})} = 8 \times 10^6 \text{ secs} = 92 \text{ days}!$$

(ii) InfiniBand: 14 Gbits/s

$$T_{\text{message}} = \frac{2\text{TBytes}}{14\text{Gbits/s}} = \frac{16 \times 10^{12} \text{ bits}}{14 \times 10^9 \text{ secs}} = 1.1 \times 10^3 \text{ secs} = 19 \text{ minutes}$$

(iii) CA Pacific Research Platform: 100 Gbits/sec

$$T_{\text{message}} = \frac{2\text{TBytes}}{100\text{Gbits/s}} = \frac{16 \times 10^{12} \text{ bits}}{100 \times 10^9 \text{ secs}} = 1.2 \times 10^2 \text{ secs} = 2.6 \text{ minutes}$$
[Q5] (15 points):
The serial code shown below performs operations on several vectors. It is running on a CPU that has a theoretical peak performance of 2 GigaFlops, supports the following operations: ' + ' , ' - ' , ' * ' , '=' , ';' , '/' , and each operation takes \( T_{\text{op}} = 10^{-8} \) seconds.

```c
include <stdio.h>
int n, m, p = 4, r = 5, s = 2;

main()
{
    double x[q], y[r], z[s];
    int i, j, k;

    for (i = 0; i < q; i++)
        x[i] = 1;

    for (j = 0; j < r; j++)
        y[j] = sin(j * pi / 4);

    for (k = 0; k < s; k++)
        z[k] = 0.0;

    for (k = 0; k < s; k++)
        for (j = 0; j < r; j++)
            for (i = 0; i < q; i++)
                z[k] = z[k] + x[i] * y[j];

    return;
}
```

(a) (2 points): Identify the key blocks of code that should be timed: draw and number/label the code blocks. How many blocks are there?

\( b = 4 \)

(b) (6 points): Which block will do the most work? Define your choice using a \( \vartheta (n) \) analysis to estimate the number of operations, \( N_{\text{ops}} \).

**Block 4 does the most work because it contains 3 nested loops and 3 operands:** = , + , .

\[
N_{\text{ops}} \approx (\# \text{iters in loop } 1) \times (\# \text{iters in loop } 2) \times (\# \text{iters in loop } 3) \times (\# \text{ops on line } 20)
= s \times r \times q \times 3 = 2 \times 5 \times 4 \times 3 \approx 120 \text{ ops.}
\]

(c) (7 points): For the longest block that you have identified, write an equation to estimate the total run time using your estimated value for \( N_{\text{ops}} \), and calculate the result in seconds.

The time per operation \( T_{\text{op}} = 10^{-8} \) seconds.
The estimated number of operations is \( N_{\text{ops}} = 120 \).
The total number of operations is then:

\[
T_{\text{runtime}} \approx (\text{Estimated } \# \text{ of Ops}) \times (\text{Machine Time} / \text{Op}) = (N_{\text{ops}}) \times (T_{\text{op sec/op})
= (120 \text{ ops}) \times (10^{-8} \text{ s/op})
= 1.2 \times 10^{-6} \text{ seconds}
\]