Question 1 (1.1 points)

Given the following main code of a program, where we assume that `n` is a constant defined with an integer value:

```
float a;
float A[n][n], X[n][n], Y[n][n], Z[n][n];
T1( A, X, Y, Z );
a = T2( A );
T3( a, X );
T4( a, Y );
T5( A, A, Z );
T6( X, Y, Z );
```

and given the code of the following functions:

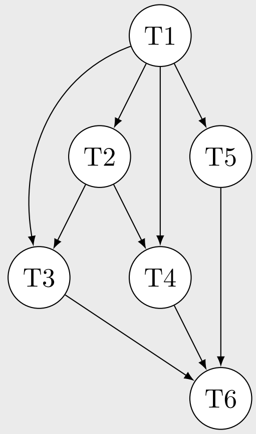
```
float T2( float A[n][n] ) {
  float a = 0.0;
  for( int i=0; i<n; i++ )
    for( int j=i; j<n; j++ )
      for( int k=i; k<n; k++ )
        a = a + A[i][k];
  return a;
}

void T5( float X[n][n], float Y[n][n], float Z[n][n] ) {
  float aux;
  for( int i=0; i<n; i++ )
    for( int j=0; j<n; j++ ) {
      aux=0;
      for ( int k=0; k<n; k++ )
        aux += X[i][k]*Y[k][j];
      Z[i][j]=aux;
    }
}
```

where we know that the function `T1` modifies all its arguments and has a cost of n^3 flops, while the functions `T3`, `T4` and `T6` modify only their last argument and also have a cost of n^3 flops. The function `T2` has a cost of $\frac{n^3}{3}$ and the function `T5` has a cost of $2n^3$.

0.3 p. (a) Draw the graph of data dependencies between tasks.

Solution:



0.6 p. (b) Implement a parallel version with OpenMP using a single parallel region.

Solution:

```
float a;
float A[n][n], X[n][n], Y[n][n], Z[n][n];
T1( A, X, Y, Z );
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    a = T2( A );
    #pragma omp section
    T5( A, A, Z );
  }
  #pragma omp sections
  {
    #pragma omp section
    T3( a, X );
    #pragma omp section
    T4( a, Y );
  }
} /* End of parallel */
T6( X, Y, Z );
```

0.2 p. (c) Get the speedup of the parallel version of the previous section for 2 processors.

Solution:

Sequential execution time:

$$t(n) = n^3 + \frac{n^3}{3} + n^3 + n^3 + 2n^3 + n^3 = \left(6 + \frac{1}{3}\right) n^3 = \frac{19}{3} n^3 \text{ flops}$$

Parallel execution time for $p = 2$:

$$t(n,p) = n^3 + \max\left(\frac{n^3}{3}, 2n^3\right) + \max(n^3, n^3) + n^3 = 5n^3 \text{ flops}$$

Speedup:

$$S = \frac{\frac{19}{3}n^3}{5n^3} = \frac{19}{15} = 1.27$$

Question 2 (1.2 points)

We want to manage a contest of nP participants, by a jury composed of nM members, numbered from position 0 onwards. Each jury casts two votes, one of 10 points for one participant and another of 5 for another. To do so, the function manage_votes receives the vector votes_jury, of nM*2 components, with the two votes cast by each member of the jury and completes the vector pts_participants, of nP elements, with the score achieved by each participant. In addition to calculating which participant has won the contest (maximum score), the function completes the vectors m10pts and m5pts with the identifiers of the members of the jury who have awarded 10 points or 5 points, respectively, to the participant indicated as argument of the function. At the end, it obtains in the variable nPZeroPts the number of participants without any vote.

```
void manage_votes(int votes_jury[], int participant) {
  int i, p10pts, p5pts, pts_participants[nP];
  int m10pts[nM], m5pts[nM];
  int nM10pts=0, nM5pts=0, nPZeroPts=0;
  int pts_max=0, winner;
  ...
  for (i=0;i<nM;i++) {
    // Accumulate points to participants
    p10pts=votes_jury[i*2];
    p5pts=votes_jury[i*2+1];
    pts_participants[p10pts]+=10;
    pts_participants[p5pts]+=5;
    // Obtain the members that have voted the indicated participant
    if (p10pts==participant) {
      m10pts[nM10pts]=i;
      nM10pts++;
    }
    else if (p5pts==participant) {
      m5pts[nM5pts]=i;
      nM5pts++;
    }
  }
  // Calculate the winner and the number of participants with 0 points
  for (i=0;i<nP;i++) {
    if (pts_participants[i]>pts_max) {
      pts_max=pts_participants[i];
      winner=i;
    }
    else if (pts_participants[i]==0)
      nPZeroPts++;
  }
  ...
}
```

Parallelize the voting management in the most efficient way possible, using a single parallel region.

Solution:

```
void manage_votes(int votes_jury[], int participant) {
  int i, p10pts, p5pts, pts_participants[nP];
  int m10pts[nM], m5pts[nM];
  int nM10pts=0, nM5pts=0, nPZeroPts=0;
  int pts_max=0, winner;
  ...
  #pragma omp parallel
  {
    #pragma omp for private(p10pts, p5pts)
```

```
    for (i=0;i<nM;i++) {
      // Accumulate points to participants
      p10pts=votes_jury[i*2];
      p5pts=votes_jury[i*2+1];
      #pragma omp atomic
      pts_participants[p10pts]+=10;
      #pragma omp atomic
      pts_participants[p5pts]+=5;
      // Obtain the members that have voted the indicated participant
      if (p10pts==participant) {
        #pragma omp critical (pts10)
        {
          m10pts[nM10pts]=i;
          nM10pts++;
        }
      }
      else if (p5pts==participant) {
        #pragma omp critical (pts5)
        {
          m5pts[nM5pts]=i;
          nM5pts++;
        }
      }
    }
    // Calculate the winner and the number of participants with 0 points
    #pragma omp for reduction(+:nPZeroPts)
    for (i=0;i<nP;i++) {
      if (pts_participants[i]>pts_max) {
        #pragma omp critical
        if (pts_participants[i]>pts_max) {
          pts_max=pts_participants[i];
          winner=i;
        }
      }
      else if (pts_participants[i]==0)
        nPZeroPts++;
    }
  }
  ...
}
```

Question 3 (1.2 points)

Given the following function:

```
void normalize_mat(double A[N][N]){
  int i, j;
  double s, norm1=0, norm2=0;
  for (i = 0; i < N; i++){
      s = 0;
      for (j=0; j<N; j++){
          norm1+=A[i][j];
          s+= fabs(A[i][j]);
      }
      if (s>norm2)
         norm2= s;
  }
  norm1=sqrt(norm1)*norm2;
```

```
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            A[i][j]*=norm1;
  }
```

0.9 p. (a) Parallelize the previous function with OpenMP, using a single parallel region.

Solution:

```
#pragma omp parallel
{
  #pragma omp for private(j,s) reduction(+:norm1) reduction(max:norm2)
   for (i = 0; i < N; i++){
       ...
   }
  #pragma omp single
  norm1=sqrt(norm1)*norm2;
  #pragma omp for private(j)
   for (i = 0; i < N; i++)
       for (j = 0; j < N; j++)
           A[i][j]*=norm1;
}
```

0.2 p. (b) Calculate the sequential and parallel cost, assuming that N is divisible by the number of threads p and that the cost of the functions `fabs` and `sqrt` is 1 flop. Indicate the intermediate computations done to reach the solution.

Solution: Sequential cost:

$$t(N) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} 3 + 2 + \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} 1 = 3N^2 + 2 + N^2 \approx 4N^2 flops.$$

Parallel cost:

$$t(N,p) = \sum_{i=0}^{N/p-1}\sum_{j=0}^{N-1} 3 + 2 + \sum_{i=0}^{N/p-1}\sum_{j=0}^{N-1} 1 = \frac{3N^2}{p} + 2 + \frac{N^2}{p} \approx \frac{4N^2}{p} flops.$$

0.1 p. (c) Calculate the speedup and the efficiency.

Solution:

$$S(n,p) = \frac{4N^2}{\frac{4N^2}{p}} = p.$$

$$E(n,p) = \frac{S(n,p)}{p} = 1.$$