Parcial MPI 2024

Cuestión 2 (1.2 puntos)

Dada la siguiente función:

```
void normaliza( float v[N] ) {
  int i;
  float max = -FLT_MAX;
  for (i=0;i<N;i++)
    if( v[i] > max )
      max = v[i];
  for (i=0;i<N;i++)

     v[i] /= max;
 }
```

(a) Haz una versión paralela usando MPI, suponiendo que el vector `v` se encuentra inicialmente solo en el proceso 0 y, al finalizar, el proceso 0 debe contener el vector `v` modificado. Deberán distribuirse los datos necesarios para que todos los cálculos se repartan de forma equitativa. Nota: Se puede asumir que `N` es divisible entre el número de procesos.

(b) Calcula el coste computacional (en flops) de la versión secuencial y de la versión paralela desarrollada en el apartado anterior. Detalla el coste de las operaciones de comunicaciones empleadas. Asume que las comparaciones entre números reales cuestan 1 flop.

```
void normaliza( float v[N] ) {
  int i, p;
  float max, max_loc=-FLT_MAX;
  float vloc[N];
  MPI_Comm_size( MPI_COMM_WORLD, &p );
  MPI_Scatter( v, N/p, MPI_FLOAT, vloc, N/p, MPI_FLOAT, 0, MPI_COMM_WORLD );
  for (i=0;i<N/p;i++)
    if( vloc[i] > max_loc )
      max_loc = vloc[i];
  MPI_Allreduce( &max_loc, &max, 1, MPI_FLOAT, MPI_MAX, MPI_COMM_WORLD );
  for (i=0;i<N/p;i++)
    vloc[i] /= max;
  MPI_Gather( vloc, N/p, MPI_FLOAT, v, N/p, MPI_FLOAT, 0, MPI_COMM_WORLD );
}
```

Solución:

Coste secuencial: $t(N) = \sum_{i=0}^{N-1} 1 + \sum_{i=0}^{N-1} 1 = 2N$ flops.

Coste aritmético paralelo: $t_a(N,p) = \sum_{i=0}^{\frac{N}{p}-1} 1 + \sum_{i=0}^{\frac{N}{p}-1} 1 = \frac{2N}{p}$ flops.

Coste Scatter: $(p-1)\left(t_s + \frac{N}{p}t_w\right) \approx pt_s + Nt_w$.

Coste Allreduce (Reduce + Bcast, $p-1$ flops): $2(p-1)(t_s + t_w) + (p-1) \approx 2pt_s + 2pt_w + p$.

Coste Gather: $(p-1)\left(t_s + \frac{N}{p}t_w\right) \approx pt_s + Nt_w$.

Coste paralelo: $t(N,p) \approx 4pt_s + 2(N+p)t_w + \frac{2N}{p} + p$ flops.

Cuestión 3 (1 punto)

Se quiere implementar una función MPI para transmitir una matriz cuadrada de enteros de dimensión n, siendo n un número par, del proceso P0 al proceso P1, de manera que el proceso P1 reciba la matriz con las columnas adyacentes intercambiadas; es decir, si la matriz del proceso P0 es, por ejemplo:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

el proceso P1 recibirá la matriz:

$$B = \begin{bmatrix} 2 & 1 & 4 & 3 \\ 6 & 5 & 8 & 7 \\ 10 & 9 & 12 & 11 \\ 14 & 13 & 16 & 15 \end{bmatrix}.$$

La transmisión de la matriz debe hacerse mediante solo dos mensajes y sin realizar copias intermedias de los datos. La cabecera de la función a implementar será:

```
comunica_matriz(int A[n][n], int B[n][n], int rank)
```

donde `A` es la matriz almacenada en el proceso P0, `B` es la matriz donde se van a recibir los datos en el proceso P1 y `rank` es el identificador del proceso local (el programa puede tener más de dos procesos).

Solución: Teniendo en cuenta que las filas de las matrices se encuentran almacenadas en posiciones consecutivas de memoria y definiendo un nuevo tipo de datos derivado consistente en $n^2/2$ bloques de 1 elemento con una `stride` igual a 2, con solo dos mensajes se puede obtener la matriz B en el proceso P_1.

```
void comunica_matriz(int A[n][n], int B[n][n], int rank){
  MPI_Status stat;
  MPI_Datatype int_col;
  MPI_Type_vector(n*n/2, 1, 2, MPI_INT, &int_col);
  MPI_Type_commit(&int_col);
  if (rank==0) {
    MPI_Send(&A[0][0], 1, int_col, 1, 100, MPI_COMM_WORLD);
    MPI_Send(&A[0][1], 1, int_col, 1, 200, MPI_COMM_WORLD);
  }
  else if (rank==1) {
    MPI_Recv(&B[0][1], 1, int_col, 0, 100, MPI_COMM_WORLD, &stat);
    MPI_Recv(&B[0][0], 1, int_col, 0, 200, MPI_COMM_WORLD, &stat);
  }
  MPI_Type_free(&int_col);
}
```