Qüestió 1 (1.3 punts)

Donada la següent funció:

```
void computev(double v[N]) {
   double A[N][N], B[N][N], C[N][N];
   readm1(A);      // T1: llig una matriu A (cost N^2 flops)
   readm2(B);      // T2: llig una matriu B (cost N^2 flops)
   readv(v);       // T3: llig un vector v  (cost  N flops)
   p2Mat(A,C);     // T4
   pMatVec(B,v);   // T5
   pMatVec(C,v);   // T6
   pMatVec(B,v);   // T7
}
```

siendo

```
void p2Mat(double A[N][N],double B[N][N]){
   int i, j, k;
   for (i= 0; i< N; i++)
      for (j = 0; j < N; j++) {
          B[i][j]=0.0;
          for (k= 0; k< N; k++)
             B[i][j]+=A[i][k]*A[k][j];
      }
}

void pMatVec(double A[N][N],double v[N]){
   int i, j, k;
   double aux;
   for(k=0;k<N;k++)
      for (i= 0; i< N; i++){
         aux=v[i];
         for (j = 0; j < N; j++)
             aux+=A[i][j]*v[j]+2.0;
         v[i]=aux;
      }
}
```

0.1 p. (a) Calcula el cost computacional de les funcions `p2Mat` i `pMatVec`.

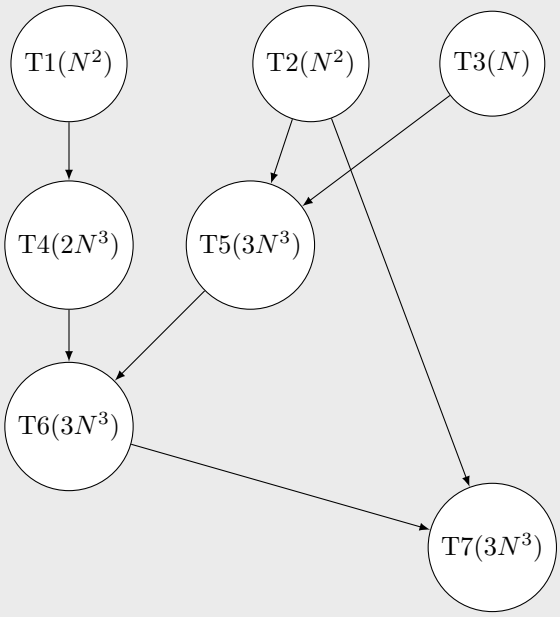
Solució:

$$\text{p2Mat}: \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1} 2 = 2N^3 \text{ flops.}$$

$$\text{pMatVec}: \sum_{k=0}^{N-1}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} 3 = 3N^3 \text{ flops.}$$

0.3 p. (b) Dibuixa el graf de dependències.

Solució:



0.7 p. (c) Implementa una versió paral·lela amb MPI utilitzant només dos processos, tenint en compte que el vector v obtingut deu quedar emmagatzemat en la memòria local del procés P0 i que l'assignació de tasques a processos maximitze el paral·lelisme i minimitze el cost de comunicacions.

Solució: Una assignació que maximitza el paral·lelisme i minimitza el cost de comunicacions consisteix en que les tasques T2, T3, T5 i T7 s'assignen al procés P0 i les tasques T1, T4 i T6 s'assignen al procés P1.

```
void computevp(double v[N]) {
   double A[N][N], B[N][N], C[N][N];
   int rank, p;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   if(rank==0) {
      readm2(B);      // T2: llig una matriu B (cost N^2 flops)
      readv(v);       // T3: llig un vector v  (cost  N flops)
      pMatVec(B,v);   // T5 (cost 3N^3)
      MPI_Send(v, N, MPI_DOUBLE, 1, 100,MPI_COMM_WORLD);
      MPI_Recv(v, N, MPI_DOUBLE, 1, 200, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      pMatVec(B,v); // T7 (coste 3N^3)
   }
   else if(rank==1){
      readm1(A);      // T1: llig una matriu A (cost N^2 flops)
      p2Mat(A,C);     // T4 (cost 2N^3)
      MPI_Recv(v, N, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      pMatVec(C,v);   // T6  (cost 3N^3)
      MPI_Send(v, N, MPI_DOUBLE, 0, 200,MPI_COMM_WORLD);
   }
}
```

0.2 p. (d) Calcula el cost seqüencial i el cost paral·lel.

Solució:

Cost seqüencial:

$$t_s(N) = N^2 + N^2 + N + 2N^3 + 3N^3 + 3N^3 + 3N^3 \approx 11N^3 \text{flops}.$$

Cost paral·lel:

$$t_a(N,2) = N + N^2 + 3N^3 + 3N^3 + 3N^3 \approx 9N^3 \text{flops},$$
$$t_c(N,2) = 2(t_s + Nt_w),$$
$$t(N,2) = t_a(N,2) + t_c(N,2) \approx 9N^3 \text{flops} + 2(t_s + Nt_w).$$

Qüestió 2 (1.1 punts)

Donada una matriu `A`, de `F` files i `C` columnes, i un vector `v` de dos elements, la següent funció calcula el vector `x`, de `F` elements.

```
void calcula(double A[F][C], double v[2], double x[F]) {
  int i, j;
  double sum, min;
  /* Calcula el vector x
   * x[i] = suma d'elements de la fila i de A que son >=v[0] i <=v[1] */
  for (i=0; i<F; i++) {
    sum=0;
    for (j=0; j<C; j++) {
      if (A[i][j]>=v[0] && A[i][j]<=v[1])
        sum += A[i][j];
    }
    x[i] = sum;
  }
  /* Calcular el mínim element de x, i restar-lo a tots els elements */
  min=x[0];
  for (i=1; i<F; i++)
    if (x[i]<min) min = x[i];
  for (i=0; i<F; i++)
    x[i] = x[i]-min;
}
```

0.8 p. (a) Fes una versió paral·lela de la funció anterior, on els càlculs es repartisquen de forma equilibrada entre tots els processos i les comunicacions es facen mitjançant operacions col·lectives. Inicialment, tant la matriu `A` com el vector `v` estan disponibles únicament en el procés 0, i es requerix que, al finalitzar la funció, el vector `x` estiga disponible en tots els processos. Es pot suposar que `F` és divisible entre el nombre de processos.

La capçalera de la funció serà la següent, on `Aloc` i `xloc` poden ser utilitzades per a que cada procés guarde la seua part local de `A` i `x`, respectivament.

```
void calculap(double A[F][C],double v[2],double x[F],
              double Aloc[][C],double xloc[])
```

Solució:

```
void calculap(double A[F][C],double v[2],double x[F],
              double Aloc[][C],double xloc[]) {
  int i, j, p;
  double sum, min, minloc;
  MPI_Comm_size(MPI_COMM_WORLD, &p);
  MPI_Scatter(A,F/p*C,MPI_DOUBLE,Aloc,F/p*C,MPI_DOUBLE,0,MPI_COMM_WORLD);
```

```
    MPI_Bcast(v,2,MPI_DOUBLE,0,MPI_COMM_WORLD);
    /* Calcula el vector x
     * x[i] = suma d'elements de la fila i de A que son >=v[0] i <=v[1] */
    for (i=0; i<F/p; i++) {
      sum=0;
      for (j=0; j<C; j++) {
        if (Aloc[i][j]>=v[0] && Aloc[i][j]<=v[1])
          sum += Aloc[i][j];
      }
      xloc[i] = sum;
    }
    /* Calcular el mínim element de x, i restar-lo a tots els elements */
    minloc=x[0];
    for (i=1; i<F/p; i++)
      if (xloc[i]<minloc) minloc = xloc[i];
    MPI_Allreduce(&minloc,&min,1,MPI_DOUBLE,MPI_MIN,MPI_COMM_WORLD);
    for (i=0; i<F/p; i++)
      xloc[i] = xloc[i]-min;
    MPI_Allgather(xloc,F/p,MPI_DOUBLE,x,F/p,MPI_DOUBLE,MPI_COMM_WORLD);
  }
```

0.3 p. (b) Indica el cost de comunicacions de dues operacions de comunicació diferents que hages usat en l'apartat anterior, suposant una implementació senzilla de les comunicacions.

Solució: Encara que es demanen només dues, s'indica el cost de totes:

- Scatter: $t_c = (p-1)\left(t_s + \frac{F\,C}{p} t_w\right) \approx p t_s + F\,C t_w$
- Bcast: $t_c = (p-1)(t_s + 2t_w) \approx p t_s + 2p t_w$
- Allreduce (equival a Reduce+Bcast): $t_c = 2(p-1)(t_s + t_w) \approx 2p t_s + 2p t_w$
- Allgather (equival a Gather+Bcast):
  $t_c = (p-1)\left(t_s + \frac{F}{p} t_w\right) + (p-1)\,(t_s + F t_w) \approx 2p t_s + pF t_w$

Qüestió 3 (1.1 punts)

La següent funció calcula l'arrel quadrada de la suma dels elements d'una matriu triangular inferior elevats al quadrat:

```
double arrel_sumaquad_triang_inf(double A[M][N]) {
  int i,j;
  double suma, arrel;
  suma=0;
  for (i=0;i<M;i++) {
     for (j=0;j<=i;j++) {
         suma+=A[i][j]*A[i][j];
     }
  }
  arrel=sqrt(suma);
  return arrel;
}
```

Es demana paral·lelitzar el codi de forma que es realitze un repartiment cíclic de les files de la matriu utilitzant

tipus de dades derivats, per tal d'equilibrar la càrrega i reduïr el nombre de missatges. Es suposa que:

- El procés 0 disposarà inicialment de la matriu A completa i tindrà que repartir-la entre la resta de processos per a dur a terme els càlculs oportuns en paral·lel. Cada procés emmagatzemarà les files que li corresponen en una matriu A local, amb espai només per al nombre de files que li toquen a cada procés, la qual es declararà com a paràmetre d'entrada i eixida, junt a la matriu A, en la funció a implementar.
- Per a que siga més senzill, la matriu A es distribuirà entre els processos sense tindre en consideració el que siga triangular inferior (no hi ha que preocupar-se pel fet d'enviar els elements iguals a zero situats a la dreta de la diagonal, encara que cap procés deurà operar amb eixos elements).
- El valor de retorn de la funció deurà ser vàlid en tots els processos.
- Suposarem que el nombre M de files de la matriu sempre serà múltiple del nombre de processos emprats i que M i N són constants conegudes.

Solució:

```
double arrel_sumaquad_triang_inf(double A[M][N],double Alocal[][N]) {
  int i,j,id,np,k;
  double suma, suma_local,arrel;
  MPI_Datatype files_ciclic;

  MPI_Comm_size(MPI_COMM_WORLD,&np);
  MPI_Comm_rank(MPI_COMM_WORLD,&id);
  k=M/np;
  MPI_Type_vector(k,N,np*N,MPI_DOUBLE,&files_ciclic);
  MPI_Type_commit(&files_ciclic);
  if (id==0) {
      MPI_Sendrecv(A,1,files_ciclic,0,0,Alocal,k*N,MPI_DOUBLE,
                       0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      for (i=1;i<np;i++)
             MPI_Send(&A[i][0],1,files_ciclic,i,0,MPI_COMM_WORLD);
  }
  else
     MPI_Recv(Alocal,k*N,MPI_DOUBLE,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  suma_local=0;
  for (i=0;i<k;i++) {
     for (j=0;j<=np*i+id;j++) {
        suma_local+=Alocal[i][j]*Alocal[i][j];
     }
  }
  MPI_Allreduce(&suma_local,&suma,1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
  arrel=sqrt(suma);
  MPI_Type_free(&files_ciclic);
  return arrel;
}
```