Práctica 3: Implementación distribuida de un algoritmo de equilibrado dinámco de la carga usando MPI. TSP con Branch&Bound:

UNIVERSIDAD DE GRANADA

Luis González Romero, XXXXXXXXX, luisgonromero@correo.ugr.es

Escuela Técnica Superior de Ingeniería informática y Telecomunicaciones

29 de junio de 2021

1. Programa principal

Antes del comienzo del ciclo BB, el proceso 0 lee la matriz y pone su testigo a verdadero. Después, este difunde al resto de procesos la matriz y el resto de procesos entra en equilibrado de carga esperando al reparto.

```
// el proceso 0 lee la matriz del fichero
if(id==0)
{
  LeerMatriz (argv[2], tsp0);    // lee matriz de fichero
  token_presente = true;        // el proceso 0 tiene inicialmente el testigo
}

// difusion de la matriz
MPI_Bcast(&tsp0[0][0], NCIUDADES*NCIUDADES, MPI_INT, 0, MPI_COMM_WORLD);

// guardamos los procesos relevantes para reproducir el anillo en el equilibrado
siguiente = (id+1)%size; anterior = (id-1+size)%size;
// medida de tiempo
MPI_Barrier(MPI_COMM_WORLD);
double t=MPI_Wtime();
if(id != 0)
{
  token_presente = false;
  Equilibrar_Carga(pila, fin, solucion);
  if(!fin) pila.pop(nodo);
}
```

Figura 1: Antes del ciclo BB

```
    }
    if (nueva_U) pila.acotar(U); // eliminar n
    Equilibrar_Carga(pila, fin, solucion);
    if(!fin) pila.pop(nodo);
    iteraciones++;
  }
```

Figura 2: Equilibrado de la carga dentro del ciclo BB

2. Equilibrado de carga

2.1. Comportamiento del nodo pedigüeño

Al tener la pila vacía, manda una solicitud de carga de trabajo al siguiente proceso y realiza un sondeo de mensajes. Dependiendo del tipo de mensaje que se reciba se ejecutará una parte del código concreta.

```
if(pila.vacia())
{
  MPI_Send(&id, 1, MPI_INT, siguiente, PETICION, comunicadorCarga); // enviar peticion al proceso id+1%p
  while(pila.vacia() && !fin)
  {
    MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, comunicadorCarga, &estado); // esperar mensaje de otro proceso
    switch (estado.MPI_TAG) {
```

Figura 3: Entrada y sondeo de mensajes

Para la parte de peticiones de trabajo, el nodo al no tener trabajo disponible reenviará esta al siguiente nodo, pasando el testigo al proceso anterior para controlar la detección de fin.

```
    case PETICION:  // peticion de trabajo
      MPI_Recv(&proceso_solicitante, 1, MPI_INT, anterior, PETICION, comunicadorCarga, &estado);
      MPI_Send(&proceso_solicitante, 1, MPI_INT, siguiente, PETICION, comunicadorCarga);
      if(proceso_solicitante == id) // iniciar posible deteccion de fin
      {
        estado_p = PASIVO;
        if(token_presente)
        {
          color_token = (color == NEGRO) ? NEGRO : BLANCO;

          MPI_Send(&color_token, 1, MPI_INT, anterior, TOKEN, comunicadorCarga);
          token_presente = false;
          color = BLANCO;
        }
      }
      break;
```

Figura 4: Tratamiento de peticiones

Para poder recibir la carga de trabajo solicitada, el proceso obtiene el número de trabajos a recibir y pasa a recibirlos y almacenarlos en su pila. Pasando a ser un nodo con trabajo.

```
    case NODOS: // recepcion del trabajo donado
      MPI_Get_count(&estado, MPI_INT, &count);
      MPI_Recv(pila.nodos, count, MPI_INT, estado.MPI_SOURCE, NODOS, comunicadorCarga, &estado);
      pila.tope = count;
      estado_p = ACTIVO;
      break;
```

Figura 5: Recepción de trabajo

Con la parte de token, comprobamos si el testigo que vuelve al proceso 0 tiene el color blanco, por lo que habría llegado al fin, si no es así se sigue enviando el testigo ya

que aún puede haber nodos con trabajo. Si se llegase al fin, se haría una reducción en anillo de la solución.

Si no es así, simplemente se pasa el testigo con el color correspondiente.

```
case TOKEN: // mecanismo de deteccion de fin
  MPI_Recv(&color_token, 1, MPI_INT, siguiente, TOKEN, comunicadorCarga, &estado);
  token_presente = true;
  if(estado_p == PASIVO) // si el proceso no tiene trabajo
  {
    if(id == 0 && color == BLANCO && color_token == BLANCO) // si el token vuelve al proceso 0 y permane
    {
      fin = true;
      MPI_Send(solucion.datos, 2*NCIUDADES, MPI_INT, siguiente, FIN, comunicadorCarga); // enviamos fin
      MPI_Recv(nueva_solucion.datos, 2*NCIUDADES, MPI_INT, anterior, FIN, comunicadorCarga, &estado); //
      if(nueva_solucion.ci() < solucion.ci()) CopiaNodo(&nueva_solucion, &solucion);
    }else
    {
      color_token = (color == NEGRO) ? NEGRO : BLANCO;

      MPI_Send(&color_token, 1, MPI_INT, anterior, TOKEN, comunicadorCarga);
      token_presente = false;
      color = BLANCO;
    }
  }
  break;
```

Figura 6: Detección de fin

Cuando se recibe en el sondeo que se ha llegado al fin, se hace la reducción en anillo.

```
case FIN: // todos los procesos se quedan sin trabajo
  MPI_Recv(nueva_solucion.datos, 2*NCIUDADES, MPI_INT, anterior, FIN, comunicadorCarga, &estado);
  fin = true;
  if(nueva_solucion.ci() < solucion.ci()) CopiaNodo(&nueva_solucion, &solucion);
  MPI_Send(solucion.datos, 2*NCIUDADES, MPI_INT, siguiente, FIN, comunicadorCarga);
  break;
```

Figura 7: Reducción en anillo

2.2. Comportamiento del nodo solidario

Cuando entra, lo primero que hace es un sondeo no bloqueante para ver si hay algún mensaje disponible, y si hay pues ejecuta la parte que corresponda.

```
if(!fin) // el proceso tiene nodos para trabajar
{
  MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, comunicadorCarga, &flag, &estado);
  while(flag==true) // mientras hay mensajes
  {
    switch (estado.MPI_TAG) {
```

Figura 8: Sondeo no bloqueante

En la parte de petición en este caso, se obtiene la petición del proceso solicitante y se le envía la parte de trabajo correspondiente si procede, si no se reenvía la solicitud al siguiente nodo.

Si puede enviar parte de su trabajo al nodo solicitante se calcula el color del nodo para controlar la detección de fin.

```
case PETICION:
  MPI_Recv(&proceso_solicitante, 1, MPI_INT, anterior, PETICION, comunicadorCarga, &estado);
  if(pila.tamanio() >= 2) // si hay suficientes nodos en la pila para ceder
  {
    tPila pila_equilibrada;
    pila.divide(pila_equilibrada);
    MPI_Send(pila_equilibrada.nodos, pila_equilibrada.tope, MPI_INT, proceso_solicitante, NODOS, comunicadorCarga);
    color = (id < proceso_solicitante) ? NEGRO : BLANCO; // el nodo vuelve a tener trabajo
  }else
    MPI_Send(&proceso_solicitante, 1, MPI_INT, siguiente, PETICION, comunicadorCarga); // reenviar peticion al sigu
  break;
```

Figura 9: Tratamiento de peticiones para el solidario

Para el token se limita a recibirlo.

```
  case TOKEN:
    MPI_Recv(&color_token, 1, MPI_INT, estado.MPI_SOURCE, TOKEN, comunicadorCarga, &estado);
    token_presente = true;
    break;
}
MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, comunicadorCarga, &flag, &estado); // sondear si hay m
```

Figura 10: token solidario

3. Resultados

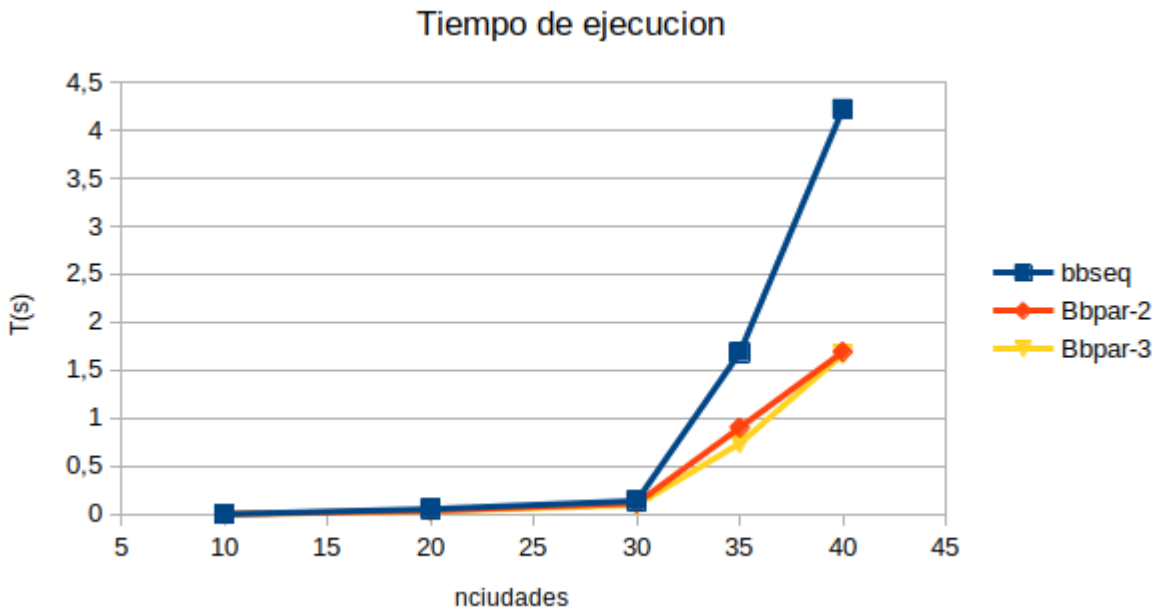

Figura 11: Tiempos de ejecución

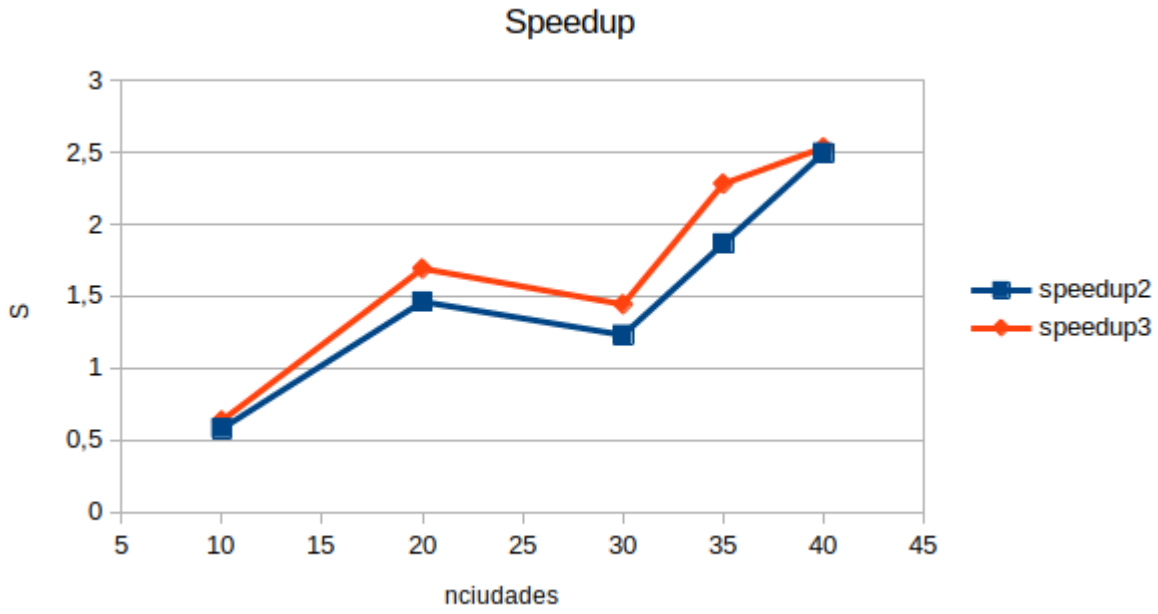

Figura 12: Ganancia obtenida

NCIUDADES	bbseq
10	207
20	3755
30	6957
35	71107
40	158556

Tabla 1: Iteraciones BB para bbseq

NCIUDADES	bbpar-2-p0	bbpar-2-p1
10	141	106
20	2388	2300
30	5573	5583
35	38208	38267
40	57286	58231

Tabla 2: Iteraciones BB para bbpar con np=2

NCIUDADES	Bbpar-3-p0	Bbpar-3-p1	Bbpar-3-p2
10	111	112	99
20	2193	2100	2080
30	4949	5107	4804
35	32303	31937	32157
40	58104	56578	57570

Tabla 3: Iteraciones BB para bbpar con np=3