Prova Scritta di Algoritmi Paralleli e Sistemi Distribuiti

Appello del 13 novembre 2023

Durata della Prova: 2,5 ore

1. (*3 punti*) Sia P(p, n) un problema parallelo di **p** processori e di dimensione **n**, caratterizzato da un overhead $\mathbf{T_o = 8\ p \log p}$ e tempo seriale $\mathbf{T_s = 2\ n}$. Siano $P_1(2, 256)$ e $P_2(8, n_2)$ due istanze di P. Affinché l'efficienza venga mantenuta al **costante** per P_1 e P_2, quale deve essere il valore di n_2 ?
    a. $n_2 = 2048$
    b. $n_2 = 3072$
    c. $n_2 = 2 * \log 256$
    d. $n_2 = 2706$

2. (*2 punti*) Dato un ipercubo di dimensione 4, sia il tempo di comunicazione punto-punto tra due nodi adiacenti uguale a **Tw**. Trascurando i tempi di computazione, il tempo impiegato da un messaggio inviato dal nodo 5 al nodo 13 è uguale **in media** a:
    a. Tw
    b. 5 * Tw
    c. Tw * Tw
    d. 13 * Tw

3. (*fino a 7 punti*) Si vuole implementare un meccanismo di "timeout" nella invocazione delle seguenti funzioni utilizzando la libreria Posix threads:

```
void doTask1(){
    ....
}

void doTask2(){
    ....
}
```

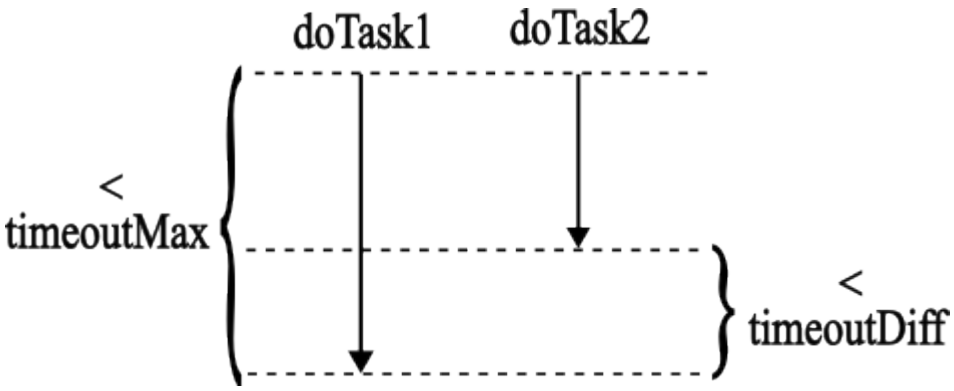
Si considerino tali funzioni come **già** implementate e si implementi la funzione:

```
void doTasksWithTimeout(int timeoutMax, int timeoutDiff){
    ....
}
```

che invochi in modo parallelo le precedenti funzioni, realizzando il seguente meccanismo di timeout:

- l'esecuzione parallela complessiva non deve durare più di `timeoutMax` secondi,
- il tempo tra la fine dell'esecuzione del più veloce e del più lento tra `doTask1()` e `doTask2()` non deve eccedere i `timeoutDiff` secondi.

Si veda, ad esempio, la seguente figura:



4. (*fino a 5 punti*) Dato la seguente porzione di codice MPI:

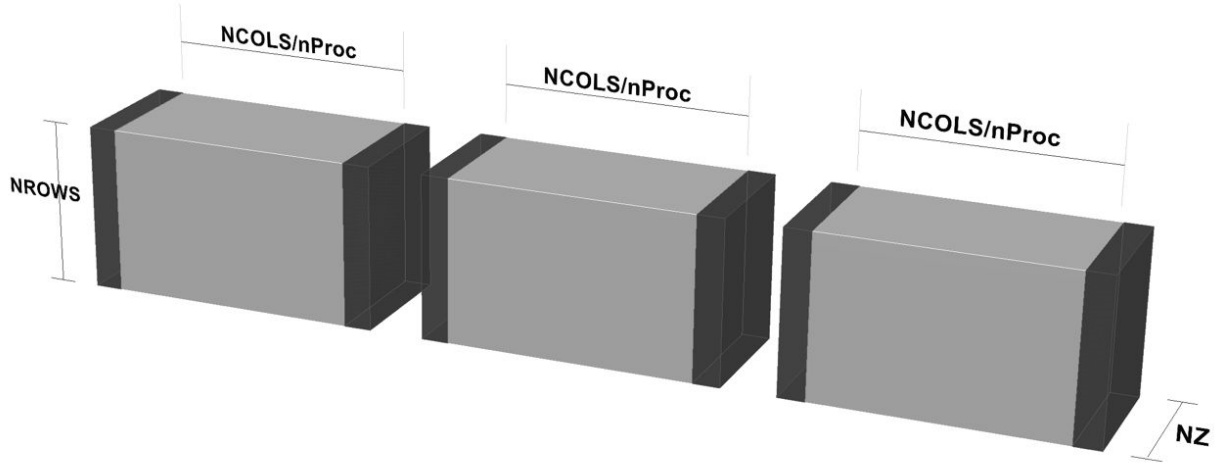
```
if (rank == 0) {
    MPI_Send(data, MAXSIZE, MPI_INT, 1, TAG, MPI_COMM_WORLD);
    MPI_Recv(data, MAXSIZE, MPI_INT, 1, TAG, MPI_COMM_WORLD, &status);
    MPI_Send(data, MAXSIZE, MPI_INT, 2, TAG, MPI_COMM_WORLD);
    MPI_Recv(data, MAXSIZE, MPI_INT, 2, TAG, MPI_COMM_WORLD, &status);

} else if (rank==1) {
    MPI_Recv(data, MAXSIZE, MPI_INT, 0, TAG, MPI_COMM_WORLD, &status);
    MPI_Send(data, MAXSIZE, MPI_INT, 0, TAG, MPI_COMM_WORLD);
    MPI_Send(data, MAXSIZE, MPI_INT, 2, TAG, MPI_COMM_WORLD);
    MPI_Recv(data, MAXSIZE, MPI_INT, 2, TAG, MPI_COMM_WORLD, &status);
} else if (rank==2) {
...
}
```

Chiamando S0 l'operazione di MPI_Send indirizzata al nodo di rank 0, S1 la MPI_Send verso il nodo di rank 1 e R0/R1 le operazioni di ricezione (MPI_Recv) dal nodo di rank 0/1, rispettivamente, indicare quale ordine di esecuzione per il rank 2 **NON esclude** la possibilità di deadlock:

a. R0-S1-R1-S0
b. R1-R0-S0-S1
c. R1-S1-R0-S0
d. R0-R1-S1-S0

5. (*fino a 7 punti*)

Si consideri l'implementazione parallela (tramite MPI) di un automa cellulare 3D di dimensione **NROWS** x **NCOLS** x **NZ**, nel quale il dominio dell'automa è partizionato in **nProc** nodi lungo le <u>colonne</u> (si veda figura seguente dove i bordi – halo cells - di sinistra e di destra sono colorati più scuri).



Si consideri che il dominio dell'automa, per ogni nodo, è memorizzato a partire dalle righe, poi per colonne e infine per z. In pratica, la macro per accedere utilizzando le 3 coordinate di riga, colonna e z è del tipo:

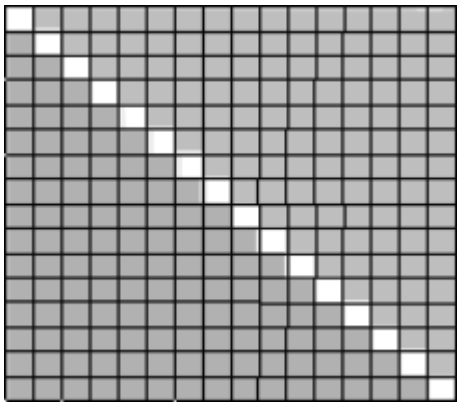
```
#define v(r,c,z) ((r)*(NCOLS/nProc+2)*NZ+(c)*NZ+(z))
```

Volendo scambiare ogni bordo con **una sola** operazione send/receive, quale delle seguenti dichiarazioni MPI_Type_vector risulta corretta?

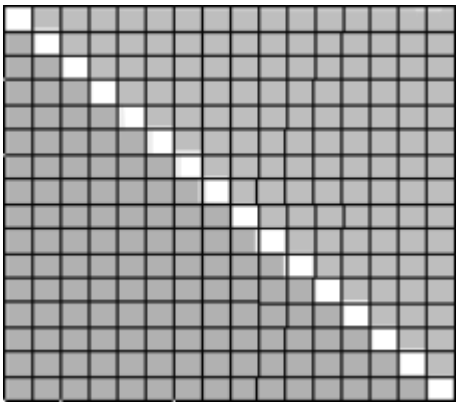
a.
```
MPI_Type_vector(NROWS, NZ*NROWS, NZ*NCOLS/nProc,
  MPI_INT, &DT);
```
b.
```
MPI_Type_vector(NROWS, NZ*NCOLS/nProc, NZ*NROWS,
  MPI_INT, &DT);
```
c.
```
MPI_Type_vector(NROWS, NZ*NROWS, NZ*NCOLS/nProc+2*NZ,
  MPI_INT, &DT);
```
d.
```
MPI_Type_vector(NROWS, NZ*(NCOLS/nProc+2), NZ*NROWS,
  MPI_INT, &DT);
```

6. (*fino a 6 punti*)

Si vuole realizzare uno scambio di dati tramite send/receive MPI tra 2 nodi entrambi ospitanti una matrice quadrata di interi, in modo da copiare il contenuto di tutta la matrice, **eccetto** la diagonale principale, dal nodo 0 al nodo 1 (si veda la seguente figura):



Nodo 0



Nodo 1

Si consideri il seguente codice:

```
#define N 200
int rank;
int mat[N][N];

void exchAllButDiagonal(){
      ...
}

int main(int argc, char *argv[]) {

      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);

      exchAllButDiagonal();

      MPI_Finalize();

      return 0;
}
```

Si implementi la funzione `exchAllButDiagonal()` in modo da realizzare lo scambio di dati tramite **una sola** operazione di send/receive utilizzando opportunamente le funzionalità di MPI_Type_vector.
N.B.: non sarà considerata valida una implementazione nella quale viene trasferita tutta la matrice inclusa la diagonale.

Signature Posix

```
//creazione thread
int pthread_create(pthread_t * thread,
                       const pthread_attr_t * attr,
                       void * (*start_routine)(void *),
                       void *arg);


// join
int pthread_join( pthread_t thread,void** value_ptr );


//mutex
int pthread_mutex_init(pthread_mutex_t *mutex,
     pthread_mutex_attr *attr);
int pthread_mutex_lock(pthread_mutex_t* mutex );
int pthread_mutex_unlock(pthread_mutex_t* mutex );
int pthread_mutex_destroy(pthread_mutex_t *mutex);

//condition
int pthread_cond_init( pthread_cond_t *cond,
     pthread_condattr_t *cond_attr )
int pthread_cond_destroy( pthread_cond_t *cond )
pthread_cond_wait(&a_c_v,&a_mutex);
pthread_cond_signal (pthread_cond_t *cond)
pthread_cond_broadcast (pthread_cond_t *cond)

void sleep(int seconds)
```

Signature MPI

```
MPI_Init (&argc,&argv);

MPI_Comm_size (comm,&size);

MPI_Comm_rank (comm,&rank);

MPI_Finalize ();

int MPI_Send( void *buf, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm );

int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int
source, int tag, MPI_Comm comm, MPI_Status *status );

MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count
);

int MPI_Isend( void *buf, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm, MPI_Request *request );
```

```
int MPI_Wait (MPI_Request  *request, MPI_Status   *status);

int MPI_Test (MPI_Request  *request, int *flag, MPI_Status   *status)

int MPI_Type_vector(int block_count, int block_length, int stride,
MPI_Datatype old_datatype, MPI_Datatype* new_datatype);

int MPI_Type_commit(MPI_Datatype* datatype);

int MPI_Type_free(MPI_Datatype* datatype);
```