Christian Flores Meléndez

# Algoritmos Paralelos

## 3 MPI

Ejercicios

3.1)

No sucede ningún cambio al probar el código, en teoría el tamaño cambió y no se debería imprimir la misma información, debería imprimir menos para el primer caso e imprimir de más para el segundo.

3.4)

```
if(my_rank == 0){
      printf("Proceso %d of %d!\n",my_rank,comm_sz);
   MPI_Send(&a,1,MPI_INT,1,0,MPI_COMM_WORLD);

}
else{

   MPI_Recv(&a,1,MPI_INT,my_rank-1,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      printf("Proceso %d of %d!\n",my_rank,comm_sz);
      if(my_rank!=comm_sz-1)
      MPI_Send(&a,1,MPI_INT,my_rank+1,0,MPI_COMM_WORLD);

}
```

3.5)

log_2(n) = h

Premisa: log_2(2n) = h + 1 (a lo que queremos llegar)

log_2(2n) = log_2(2) + log_2(n)

= 1 + h l.q.q.d

## 4 Pthreads

Programación

4.3) No veo ninguna ventaja, ya que en este caso no importa el orden de los procesos. Solo traen devesventasjas: El busy waiting obliga a los proceso a llegar en orden, y mientras los procesos esperan los mete en un bucle infinito, lo cual es un desperdicio de recursos. El semáforo tambien obligaría a que los procesos lleguen en orden, pero este ya no gasta recursos de más porque duerme a los procesos. Por último el mutex es el que mejor se comporta ya que no importa el orden de los procesos y tampoco gasta recursos innecesariamente pues el mutex también duerme los procesos mientras esperan.

Christian Flores Meléndez

5 OpenMP

Ejercicios

5.5)

999 y 910

En la computadora Bleeblon.

Primer programa:

```
float a[] = {4.0, 3.0, 3.0, 1000.0};

int i;
float sum=0.0;
for(i=0;i<4;i++)
 sum += a[i];
printf("sum = %4.1f\n”,sum);
```

Salida: 101

```
int i;
float sum=0.0;
#pragma omp parallel for num_threads(2) \reduction(+:sum)

for(i=0;i<4;i++)
 sum += a[i];
printf("sum = %4.1f\n”,sum);
```

Salida: 107