Universidad EAFIT

Ingeniería de Sistemas

ST0263: Tópicos Especiales en Telemática
Profesor: Edwin Montoya – emontoya@eafit.edu.co
Profesor: Álvaro Ospina – aespinas@eafit.edu.co

20202

Laboratorio de MPI

ENTREGABLES DEL LAB DE HPC EN OPENMP Y MPI

1. Lab de clase

2. GitHub: https://github.com/st0263eafit/hpc20202

3. EJECUTAR Y TOMAR TIEMPOS DE LOS PROGRAMAS 'SERIAL' Y 'PARALELO' EN OPENMP, PARA LOS CASOS DE MULTIPLICACIÓN DE MATRICES Y CALCULO DE INTEGRAL, EN DONDE CONOZCA LOS SPEEDUP Y EFICIENCIA PARA LOS SIGUIENTES ESCENARIOS:

TAMAÑOS DE MATRICES = 1000x1000, 4000x4000 y 20000x20000
Nro de intervalos para Integración = 100000, 1000000, 100000000, 1000000000

OMP_NUM_THREADS = 1 (SERIAL)
OMP_NUM_THREADS = 2 (2 CORES)
OMP_NUM_THREADS = 3 (3 CORES)
OMP_NUM_THREADS = 4 (3 CORES)
OMP_NUM_THREADS = 8 (8 CORES)
OMP_NUM_THREADS = 12 (12 CORES)
OMP_NUM_THREADS = 16 (16 CORES)
OMP_NUM_THREADS = 24 (16 CORES)

Los directorios de trabajo en el github son:

'serial'
'openmp'

La idea es graficar las diferentes opciones para el speedup vs eficiencia.

4. EJECUTAR Y TOMAR TIEMPOS DE LOS PROGRAMAS 'SERIAL' Y 'PARALELO' EN MPI TANTO EN C COMO EN PYTHON PARA EL CASO DE MULTIPLICACIÓN DE MATRICES Y VECTOR SUMA, EN DONDE CALCULE EL SPEEDUP Y EFICIENCIA PARA LOS SIGUIENTES ESCENARIOS:

TAMAÑOS DE MATRICES = 1000x1000, 4000x4000 y 20000x20000

TANTO PARA LA MULTIPLICACIÓN DE MATRICES COMO PARA EL VECTOR SUMA.

-np 2
-np 3
-np 3
-np 4
-np 8
-np 12
-np 16
-np 32
-np 48
-np 64
-np 70

La idea es graficar las diferentes opciones para el speedup vs eficiencia.

Los directorios de trabajo en el github son:

'serial'
'mpi'

1. CONEXIÓN AL CLUSTER:

¿Cómo nos vamos a conectar al cluster MPI?

Rpta: 1) nos conectamos a la VPN con cada usuario.

2) nos vamos a conectar vía SSH al front end del cluster, la IP para el laboratorio es: 192.168.10.40

el usuario es: user-vpn

password: pass-vpn

y están adentro.

3) si no estás dentro de la VPN, requieres pasar por una dirección IP pública, y posiblemente un puerto especial. Se darán las instrucciones debidas para el acceso por IP pública.

2. OPENMP

¿Cómo compilar programas en OpenMP?

Tenga en cuenta 2 variables de ambiente:

configurar y consultar el nro de de cores a utiliza en la ejecución de un programa en OpenMP

```
$ export OMP_NUM_THREADS=2

$ echo $OMP_NUM_THREADS
```

configurar el mostrar información del ambiente de ejecución de OMP:

```
$ export OMP_DISPLAY_ENV='true'

$ echo $OMP_DISPLAY_ENV
```

compilar un programa en openmp:

```
$ cd openmp

$ gcc openmp-mm.c -o openmp -fopenmp
```

en algunos casos puede necesitar la librería math.o

$ gcc openmp-mm.c -o openmp -fopenmp -lm

ejecutar el programa seria y tomar el tiempo:

$ cd serial

$ time ./serial-mm

Utilizar las anteriores instrucciones para realizar la actividad del laboratorio.

3. MPI

1. **DONDE ESTÁN LOS PROGRAMAS EJEMPLO MPI?**

Rpta: dentro del repo git del curso, allí es donde están los ejemplos y donde van a poner sus programas ejemplo y el trabajo que realicen.

El home de cada usuario se exporta vía nfs a los demás nodos del cluster, así que solo lo que está en este directorio se puede ejecutar en mpi.

2. **CUÁLES SON LOS EJEMPLOS QUE HAY?**

en 'example-class', 'llnl-mpi-examples' y otros dirs.

Localizar ejemplos: vecsum tanto serial como paralelos*

vecsum-serial.c

vecsum-mpi1.c

vecsum-mpi2.c

3. **QUE HACEN LOS EJEMPLOS VECSUM*?**

Rpta: básicamente ambos realizan lo siguiente:

Se tiene una matriz de X . Y . la matriz es "matrix

el problema consiste en crear un vector "vec_sum" de Y, en donde en cada posición vamos a almacenar la suma de cada fila en la matriz.

El problema trivial de procesamiento distribuido consiste en enviar a los nodos esclavos o workers, filas para que realicen la suma de cada fila y retornen el resultado.

4. **ANALIZAR EL PROBLEMA, Y DETERMINEN ESTRATEGIAS DE SOLUCIÓN DE MANERA PARALELA-DISTRIBUIDA DE ESTE PROBLEMA.**
5. **CUAL ESTRATEGIA DE SOLUCIÓN REPRESENTA EL VECSUM-MPI1?**
6. **CUAL ESTRATEGIA DE SOLUCIÓN REPRESENTA EL VECSUM-MPI2?**
7. **EVALUACIÓN DE DESEMPEÑO**

cada uno de los ejemplos imprime el tiempo de procesamiento, porque el ejemplo1 se demora más que el ejemplo2 para resolver el mismo problema?

8. **COMPILACIÓN Y EJECUCIÓN DE PROGRAMAS MPI**

a continuación se muestra como compilar y ejecutar programas en MPI.

1. para compilar, realice:

$ mpicc -O2 -w vecsum-mpi1.c -o vecsum-mpi1

$ mpicc -O2 -w vecsum-mpi2.c -o vecsum-mpi2

3. como se ejecutan?

$ mpirun -np 4 -f hosts_mpi -np 4 ./vecsum-mpi1

-np es el número de procesadores o nodos que emplearemos, como es un cluster de 4 nodos, utilizaremos -np 4

Que hay dentro de un programa MPI y como es su estructura? A continuación se explica.

9. **GUIA DE PROGRAMACIÓN RÁPIDA EN MPI en C**

¿Cuál es la estructura general de un programa en MPI?

Además del material de apoyo publicado en Contenidos de la material, se realiza una síntesis de como sería un template de un programa en MPI.

1. Se requiere incluir los encabezados de mpi

#include "mpi.h"

2. Se definen unas variables globales importantes, las cuales son:

```
#define MASTER 0
#define FROM_MASTER 1
#define FROM_WORKER 2

   int    taskId,
   numTasks,
   numWorkers;
```

explicación:

MASTER es una constante que nos represente siempre el proceso maestro o principal en MPI, siempre el proceso maestro tiene el id = 0;

FROM_MASTER y FROM_WORKER, nos sirven para identificar el origen de los mensajes, cuando vamos a realizar comunicaciones para envío y recepción de datos.

taskId, el identificador de la tarea o del proceso, esta es asignada por el cluster e indica el número de proceso que está ejecutando este programa. taskId = 0, es el nodo principal.

Todas las máquinas corren el mismo código, así que en tiempo de ejecución se determina este taskId.

si observan en el main del programa, notaran algo como:

```
if (taskId == MASTER)

//código dentro del nodo principal

....

if (taskId > MASTER)

//código dentro de cada esclavo.
```

---

numTasks, es el número de procesadores que tenemos.

numWorkers, es el número de esclavos, normalmente le restamos uno para excluir al master o fronted, y que sirva de orquestador e integrador. esto depende mucho del diseño.

3. Proceso de inicialización de MPI:

siempre antes de utilizar MPI, deben inicializarlo así:

```
MPI_Init(&argc, &argv);      // parametros del main()
MPI_Comm_rank(MPI_COMM_WORLD, &taskId);
MPI_Comm_size(MPI_COMM_WORLD, &numTasks);
numWorkers = numTasks-1;
```

4. Ahora, se presenta el proceso de comunicaciones de datos, normalmente, el master puede enviar primero datos a los workers, como lo hace?

Rpta: en este ejemplo, se envía a cada worker un dato entero

en este ejemplo, se envía a todos los workers:

int dato-a-enviar = 20;

int contador-datos = 1;

for (int i=1, i<=numWorkers, i++)

MPI_Send(&dato-a-enviar, contador-datos, MPI_INT, i, FROM_MASTER, MPI_COMM_WORLD);

---

observe:

el dato a enviar es &dato-a-enviar, es la dirección del buffer.

contador-datos, es el número de datos, en este caso 1.

MPI_INT, es el tipo de datos a enviar.

i, es el worker destino,

FROM_MASTER, se marcan estos mensajes como del master.

MPI_COMM_WORLD, dominio de comunicaciones, por ahora este es el más estándar.

5. Como recibimos datos de un worker?

para recibir datos de un worker en el master, podemos hacer:

MPI_Recv(&buffer-int, 1, MPI_INT, 3, FROM_MASTER, MPI_COMM_WORLD, &status);

estamos recibiendo datos en "buffer-int", cuantos? 1, del tipo entero, del worker = 3, mensajes marcados para el master (FROM_MASTER), etc, y una variable de status.

6. el proceso de envío y recepción de datos, se puede hacer de las siguientes formas:

* sincrónico vs asincrónico (bloqueante vs no-bloqueante)
* buffered vs non-buffered

en las presentaciones, se explica bien esto, y es pieza clave para el rendimiento de estos programas en MPI.

pregunta? en los ejemplos vecsum-mpi1.c y vecsum-mpi2.c, que tipo de comunicaciones se emplean? sync, async, bufff, non-buff?

porque la diferencia de rendimiento.

7. Todo programa MPI debe finalizar con:

MPI_Finalize();

10. GUIA DE PROGRAMACIÓN RÁPIDA EN MPI EN PYTHON

Vamos a utilizar el framework Python: mpi4py

$ sudo pip install mpi4py (como root, pero ya esta en el cluster)

1. Inicio de un programa MPI4PY

 # example1.py

```
from mpi4py import MPI

# red virtual de comunicacion entre los procesos.
comm = MPI.COMM_WORLD

# id de proceso, 0 para el master, >0 para los workers
rank = comm.Get_rank()

# numero de procesos en MPI

numworkers = comm.size
```

```
if rank == 0: # instrucciones que ejecuta el master
    print "master ", rank," trabajando... "
    data1 = " hola "
    data2 = " mundo "
    destination = 1

    #transmite los datos data1 al proceso estination= 1
    comm.send(data1, dest=destination)
    #transmite los datos data2 al proceso destination = 1
     comm.send(data2, dest=destination)

    # recibe datos de cualquier proceso
    datos3 = comm.recv()
    #recibe datos del proceso 2
    datos4 = comm.recv(source=2)
    print datos3,datos4

#// worker 1
if rank == 1:
    print "worker ", rank," trabajando... "
    datos1rx = comm.recv(source=0) #// recibe datos del
master, 0
    datos2rx = comm.recv(source=0) #// recibe datos del
master, 0
    comm.send(datos2rx, dest=2) #envia datos al worker 2
    comm.send(datos1rx, dest=2) #envia datos al worker 2
     print "worker ", rank," termino..."

#// worker 2
if rank==2:
    print "worker ", rank," trabajando... "
    datos1rx = comm.recv(source=1) #// recibe datos del
worker 1
    datos2rx = comm.recv(source=1) #// recibe datos del
worker 1
    comm.send(datos1rx, dest=0) #envia datos al master 0
    comm.send(datos2rx, dest=0) #envia datos al master 0
    print "worker ", rank," termino..."
#// otros workers
if rank > 2:
    print "worker ", rank," no hace nada "
```

2. **Ejecutar el programa:**

```
$mpirun -f hosts_mpi –np 4 python example1.py
```