Programación de Arquitecturas Emergentes [G4012452] [2024/2025]

Lab 1 - Setup e Introducción de algoritmos paralelos

Resumen

El objetivo de esta práctica es conocer la configuración del FinisTerrae III, el clúster de Computación de Altas Prestaciones del Centro de Supercomputación de Galicia (CESGA), implementar varios algoritmos de introducción a la programación paralela y preparar los scripts para ejecutar nuestros programas en uno de los nodos.

1. Acceso al CESGA

En la primera sesión de prácticas se asignarán las cuentas de usuario para acceder al CESGA y usar los nodos del FinisTerrae III. Es importante usar la cuenta de acceso de esta asignatura.

1.1. Conexión al FinisTerrae III

La conexión al CESGA se realiza mediante `ssh cursoNNN@ft3.cesga.es`, siendo `cursoNNN` la cuenta de usuario asignada. Por motivos de seguridad, el acceso está restringido a centros autorizados, como la USC. El acceso desde fuera de la USC debe realizarse mediante una VPN. En el campus virtual tienes un enlace a la Guía de configuración de la VPN.

☞ El correo electrónico a utilizar en la configuración VPN es `cursoNNN@cesga.es`.

1.2. Solicitar un nodo de cómputo

Los nodos del CESGA están agrupados en base a diferentes características como el número de cores, memoria RAM, aceleradores o capacidad de almacenamiento en disco. Los nodos se asignan según los recursos que necesitamos. Cada nodo tiene 2 procesadores Intel Xeon Ice Lake 8352Y con 32 cores cada uno (64 cores por nodo). Al conectar al FT3 lo hacemos a un nodo de login. Para trabajar de forma interactiva usamos el comando `compute`.

☞ Conecta a un nodo de forma interactiva y estudia la organización del procesador con el comando: `lstopo --no-io`.

Puedes guardar la salida en una imagen añadiendo un nombre de fichero y la extensión `.png`, por ejemplo: `lstopo --no-io IceLake8352Y.png`. Estudia en detalle la arquitectura del procesador.

1.3. Compilar código y enviar trabajos a un nodo

El CESGA tiene varias herramientas de desarrollo. Tenemos que indicar el compilador que queremos usar cargando el módulo correspondiente. Por ejemplo, para compilar un código escrito en lenguaje C usando el compilador GNU.

```
module load gcc
gcc hello_world.c -o hello_world
```

El envío de trabajos se hace mediante un sistema de colas. Aunque se puede hacer desde línea de comandos usando `srun`, lo habitual y recomendado por el CESGA es usar un `script` para solicitar los recursos necesarios, por ejemplo:

```
cat job.sh

    #!/bin/bash
    #SBATCH -N 1         # 1 nodo en total
    #SBATCH -c 1         # 1 core por tarea
    #SBATCH -t 00:02:00 # 2 minutos Run time (hh:mm:ss)
    # Programa a ejecutar
    ./hello_world
```

> ☞ CONSEJO: Para que te asignen antes un nodo y ejecutar tus programas, ajusta el tiempo necesario y solicita lo menos posible.

Para enviar el trabajo a la cola debes usar el comando `sbatch`:

```
sbatch job.sh
```

En las prácticas de programación paralela con OpenMP y CUDA pediremos más recursos (threads y GPUs), pero siempre limitados a un nodo.

2. Algoritmos de introducción a la programación paralela

Implementa los siguientes algoritmos en lenguaje C, de forma secuencial, y ejecútalos en un nodo del Finisterra III. Las versiones paralelas se discutirán en las próximas sesiones.

1. Implementa la operación $Y = \alpha X + Y$, donde α es un *escalar* y X, Y son vectores de tamaño N que contienen números en formato de coma flotante de doble precisión. Esta operación es conocida como `DAXPY` *Double-Precision* $A \cdot X$ *Plus* Y. Utiliza por defecto un tamaño de array N de `6GiB`.

   ```
   $ Use ./DAXPY alpha N
   ```

2. Realiza la implementación de la **distancia euclidiana** entre dos vectores X e Y de dimensión N. La distancia euclidiana entre X e Y se establece como:

   $$d(X,Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}$$

   Usa un tipo de datos *float* y arrays de `2,5GiB` por defecto.

   ```
   $ Use ./distancia N
   ```

3. Desarrolla el **histograma** para una imagen en tonos de gris. Este histograma revela cómo se distribuyen las intensidades de los píxeles en la imagen. En este caso, el histograma está compuesto por 256 valores dentro del intervalo [0..255].

   ```
   $ Use ./histograma image.png
   ```

4. Ejecuta una **convolución** bidimensional sobre una imagen X con dimensiones $M \times N$. Esta operación genera cada elemento de salida como una suma ponderada de ciertos elementos de la imagen X. Los coeficientes o pesos en la suma están determinados por una máscara de entrada K, comúnmente llamada kernel de la convolución. Los kernels suelen ser de forma cuadrada. Desarrolla la convolución de manera que sea compatible con cualquier tamaño de kernel y realiza pruebas usando el siguiente kernel de tamaño 3×3:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

```
$ Use ./conv2d image.png
```

3. Especificaciones

- Compila los programas con optimizaciones `-O2`.
- Desarrolla un script simple para compilar y ejecutar tus programas en el CESGA. El script debe ejecutar el programa 10 veces y presentar un resumen de los tiempos de ejecución (promedio de las 10 ejecuciones). Puedes revisar ejemplos de scripts en `/opt/cesga/job-scripts-examples-ft3`.
- Recuerda enviar los trabajos a la cola de procesos para medir los tiempos de ejecución.
- Encapsula cada implementación en una función. Usa la plantilla `pae_template.c` disponible en el campus virtual para cada ejercicio.
- Determina el tiempo de ejecución `wall time` en los nodos del FinisTerrae III, diferenciando siempre que sea posible el tiempo dedicado a la reserva de memoria del empleado en la computación.
- Para resolver los ejercicios 3 y 4, utiliza las imágenes que están en `Lab1_material.zip`.
- Emplea la función `loadPGMu8` en el ejercicio 3 y `loadPGM32` en el ejercicio 4, para leer los datos de entrada. Las imágenes se almacenan en memoria en forma de un array unidimensional.
- Puedes encontrar un ejemplo de uso en el archivo `pgmio_demo.c`. Consulta con el profesor en caso de dudas.
- Compara resultados numéricos con el resto de los compañeros/as.

4. Formato y fecha de entrega

1. Sube al campus virtual el código de cada algoritmo en un único archivo `.zip`..
2. Sube al campus virtual los **tiempos de ejecución** de cada algoritmo al Excel '*Hall of Fame*' disponible en la sección de prácticas:
3. Para los algoritmos de `histograma` y `conv2d` calcula la media con las 3 imágenes.

> ☞ **Fecha límite de entrega**: 17 de febrero de 2025, a las 9:00 horas.

5. Observaciones

En las siguientes sesiones prácticas, algunos de estos algoritmos se implementarán en paralelo empleando OpenMP y CUDA. Estos algoritmos presentan diversas estructuras fundamentales usadas en programación paralela. Es crucial contar con una implementación secuencial precisa para comparar tanto los resultados numéricos como los tiempos de ejecución de ambas versiones.