UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Computación Paralela y Distribuída

Proyecto - Laboratorio
Pregrado
2025-II
Profesor José Fiestas

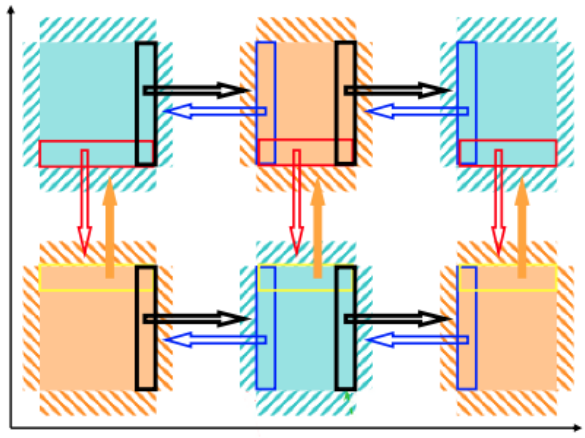
Indicaciones generales:

- Escoja **uno de los 5** proyectos mostrados
- El proyecto se desarrollará en grupos de 2 o 3 estudiantes. Los **nombres** de los integrantes deberán aparecer en el **Informe de Proyecto**
- Los archivos de proyecto se subirán directamente a Canvas
- La entrega de proyecto será hasta el **domingo 30 de noviembre**, a medianoche
- La presentación oral será:
  para la sección 1: el **lunes 01.12**, **martes 02.12** y **jueves 04.12**
  para la sección 2: el **martes 02.12**, **miércoles 03.12** y **viernes 05.12**
- La sección 8 describe **Indicaciones importantes para el proyecto**, y la sección 9, la **Rúbrica**

---

1 Ecuación de calor en 2D

En este proyecto debe paralelizar con MPI el código secuencial (adjunto)[1] que resuelve la ecuación de calor (EDP vista en clase).



El desarrollo y **análisis de resultados** debe contener lo siguiente:

a) Escoja el paradigma adecuado, elabore un PRAM y desarrolle un código en paralelo en C++

b) Registre el desarrollo del código en por lo menos 3 pasos (versiones beta). En caso utilice una fuente externa, mostrar la validación del código. Utilice descomposición del dominio considerando que n puede no ser divisible entre p. No todos los procesos tienen la misma cantidad de elementos. Utilice comunicación no-bloqueada y tipos derivados MPI

c) Realice análisis de tiempos y compare los resultados con la complejidad teórica esperada. Derive la gráfica de **speedup** de los datos anteriores y genere una gráfica de **FLOPs vs. p** (debe medir los FLOPs en el código usado).

d) Analice la escalabilidad del algoritmo con un tamaño variable del problema

Bibliografía

[1] heat.cpp

2 TSP

Resuelva el problema del agente viajero en paralelo utilizando el método *branch and bound* discutido en clase. Si utiliza otro método debe sustentarlo teóricamente.
Se trata de un vendedor que debe minimizar el recorrido entre n ciudades. Puede empezar en cualquiera de ellas y terminar en la misma luego del recorrido. Cada ciudad debe ser visitada solo una vez.

TSP se aplica a diversos problemas como DNA y genomas, conexiones de electricidad entre ciudades, conexión satelital, conexiones en fibra óptica, VLSI (Very Large System Integration)

Escoja una de las siguientes aplicaciones del algoritmo en paralelo:

- Aplique el algoritmo TSP desarrollado a un set VLSI del siguiente enlace: VLSI datasets. Escoja el set adecuado para que los resultados obtenidos sean claros. En caso los recursos disponibles (hardware) no sean suficientes para completar algunos de los casos VLSI, puede reducir la data original
- Utilice 10 puntos geográficos en sendos países del mundo y calcule las distancias entre ellos. Encuentre la ruta (aérea) más corta al recorrer los 10 puntos. Compare los resultados escogiendo ahora 20 puntos.

Puede escribir el código desde cero o utilizar algún código fuente para solucionar el problema, pero necesita validarlo, mostrando su **precisión** antes de usarlo para el caso planteado.

El desarrollo y **análisis de resultados** debe contener lo siguiente:

a) Escoja el paradigma adecuado, elabore un PRAM y desarrolle un código en paralelo en C++

b) Registre el desarrollo del código en por lo menos 3 pasos (versiones beta). En caso utilice una fuente externa, mostrar la validación del código. Si utiliza un método distinto al indicado, debe validarlo teóricamente

c) Realice análisis de tiempos y compare los resultados con la complejidad teórica esperada. Derive la gráfica de **speedup** de los datos anteriores y genere una gráfica de **FLOPs vs. p** (debe medir los FLOPs en el código usado).

d) Analice la escalabilidad del algoritmo con un tamaño variable del problema

3 Expresiones regulares

Expresiones regulares son una forma de representar un conjunto de cadenas de caracteres que satisfacen ciertas reglas de construcción (gramáticas). E.g. dado el alfabeto $\Sigma = \{0, 1\}$, la expresión regular $01(01)^*$ acepta cadenas como {01,0101,010101,...}. Expresiones regulares se usan frecuentemente en algoritmos de manipulación de strings, de análisis lexicográfico y sintáctico, entre otros. Un ejemplo de ello es NetKat, un lenguaje de programación de redes [1]

Un software de reconocimiento de expresiones regulares es de mucha utilidad, especialmente si es escalable. PAREM [2] es un ejemplo de un sotware de expresiones regulares rápido y escalable, con un speedup lineal con respecto al número de procesos. Este algoritmo particiona la cadena (input), para luego combinar los resultados parciales obtenidos (ver paper).

El desarrollo y **análisis de resultados** debe contener lo siguiente:

a) Escoja el paradigma adecuado, elabore un PRAM y desarrolle un código en paralelo en C++

b) Registro del desarrollo del código en por lo menos 3 pasos (versiones beta)

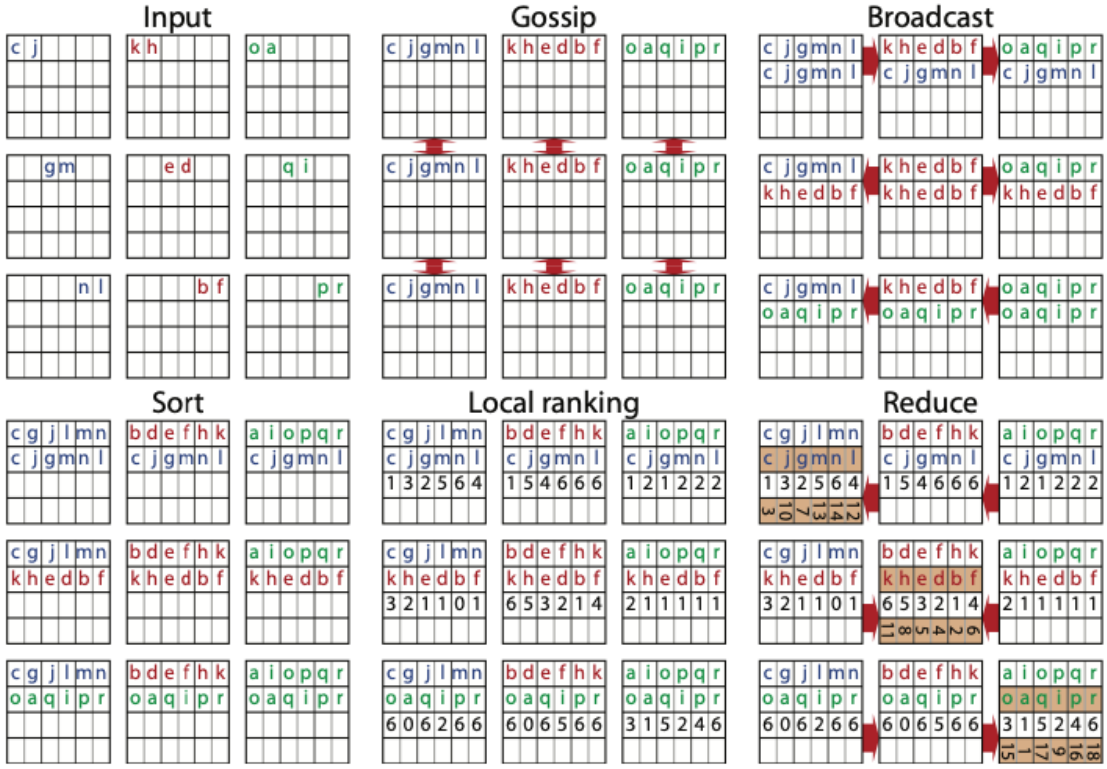
c) Realice el análisis de tiempos, comparando mediciones con la complejidad teórica del código. Para distinto número de procesos y tamaño del problema. Derive la gráfica de **speedup** de los datos anteriores y genere una gráfica de **FLOPs vs. p** (debe medir los FLOPs en el código usado).

d) Analice la escalabilidad del algoritmo con un tamaño variable del problema

Bibliografía:
[1] arxiv.org/pdf/1412.1741.pdf

4 Ordenamiento eficiente por ranking

Se pide desarrollar un código en paralelo para el algoritmo de ordenamiento por ranking visto en clase



El desarrollo y **análisis de resultados** debe contener lo siguiente:

a) Elaborar un PRAM y el código en paralelo (MPI)

b) Registro del desarrollo del código en por lo menos 3 pasos (versiones beta).

c) Mida los tiempos de **ejecución**, **cómputo** y **comunicación**, así como la velocidad del algoritmo, y representarlas en gráficas. Compare los resultados con la complejidad teórica en paralelo, para distinto número de procesos y tamaño del problema. Utilice un tamaño adecuado del array para lograr tiempos de ejecución medibles

d) Derive la gráfica de **speedup** de los datos anteriores y genere una gráfica de **FLOPs vs. p** (debe medir los FLOPs en el código usado).

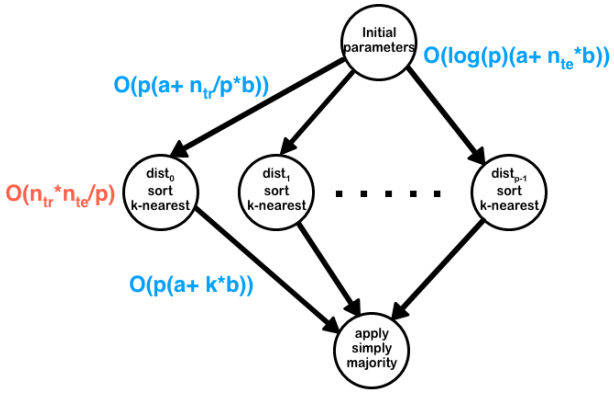
e) Compare la escalabilidad del software con Quicksort en paralelo, con un tamaño variable del problema

Bibliografía:

[1] Algoritmo similar, alternativo, en: Curso Algoritmos Paralelos, Peter Sanders, 2020/2021. Puede usarlo en vez del discutido en clase si le parece.

5 KNN

KNN es un algoritmo de clasificación supervisada (aunque también puede aplicarse a regresión) que clasifica un punto nuevo según las etiquetas de sus k vecinos más cercanos en el espacio de características. En paralelo, cada elemento de testeo puede ser calculado independientemente del resto con la fracción de la data de prueba que tiene cada proceso.



Paralelice el código adjunto *knn_digits_sec.py* [1], que utiliza la conocida librería *load_digits* para clasificar dígitos.
El desarrollo y **análisis de resultados** debe contener lo siguiente:

a) Elabore el código en Python incluyendo las directivas de comunicación (comm.bcast, comm.scatter, comm.gather)[2] de acuerdo al criterio mostrado en el DAG

b) Registre el desarrollo del código en por lo menos 3 pasos (versiones beta). En caso utilice una fuente externa, mostrar la validación del código

c) Mida los tiempos de **ejecución**, **cómputo** y **comunicación** (represéntelas gráficamente), así como la **precisión** del modelo (accuracy). Incremente tanto procesos (p) como datos (n). Utilice la complejidad teórica para **validar** el resultado obtenido. Describa como normaliza la expresión teórica para que pueda aproximar los experimentos. ¿Cuál estima, sería la cantidad **óptima de procesos**?

d) Derive la gráfica de **speedup** de los datos anteriores y genere una gráfica de **FLOPs vs. p** (FLOP por segundo). Obtenga los FLOP de la región paralelizable (entrenamiento), basada en la fórmula de la **distancia euclidiana entre dos puntos**.

e) Analice la escalabilidad del algoritmo con un tamaño variable del problema

Bibliografía:
[1] https://drive.google.com/file/d/1wTBSh7bwT1QVvce0pFMv5tcCWsg4-HeK/view?usp=sharing
[2] https://mpi4py.readthedocs.io/en/stable/

6 Marching Cubes

Marching Cubes es un algoritmo que recorre una malla 3D de valores escalares para extraer contornos/una superficie (malla triangular) correspondiente a un nivel de isovalor (isolevel). Se aplica e.g. en mapas topográficos, simulación de fluidos, tomografías médicas, reconstrucción (3D).

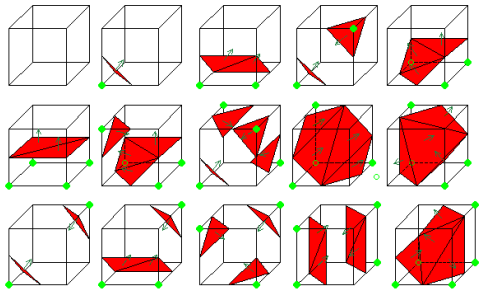


Figure 1: Configuración de Marching cubes

Implemente una versión paralela del algoritmo de Marching Cubes para acelerar la generación de contornos/iso-superficies a partir de campos escalares volumétricos
El desarrollo y **análisis de resultados** debe contener lo siguiente:

a) Escoja el paradigma adecuado, elabore un PRAM y desarrolle un código paralelo en C++

b) Registro del desarrollo del código en por lo menos 3 pasos (versiones beta)

c) Realice el análisis de tiempos, comparando mediciones con la complejidad teórica del código. Para distinto número de procesos y tamaño del problema. Utilice por lo menos matrices de 512 celdas por lado

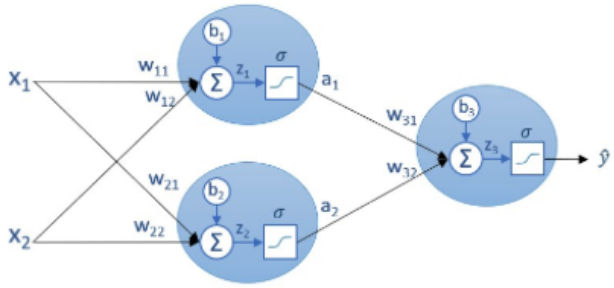
d) Derive la gráfica de **speedup** de los datos anteriores y genere una gráfica de **FLOPs vs. p** (debe medir los FLOPs en el código usado).

e) Analice la escalabilidad del algoritmo con un tamaño variable del problema

Bibliografía:
[1] https://en.wikipedia.org/wiki/Marching˙squares
[2] https://en.wikipedia.org/wiki/Marching˙cubes

7 Redes Neuronales

La multiplicación de matrices es fundamental en redes neuronales, ya que es la operación matemática central que permite:
1. Propagación hacia adelante (Forward Propagation) En cada capa de una red neuronal, los datos de entrada se transforman mediante una multiplicación de matriz con los pesos de esa capa: y=W X+b donde, X: entrada (vector o matriz), W: matriz de pesos, b: sesgo (bias), y: salida lineal
2. Entrenamiento y Backpropagation Durante el entrenamiento, el algoritmo de retropropagación también utiliza multiplicación de matrices para calcular los gradientes: $\delta = (W^T) \cdot \delta_{siguiente}$



$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix}$$

$T_s(n) = O(E \cdot n \cdot d \cdot h)$
E: número de épocas, **n:** número de muestras, **d:** número de features (entradas), **h:** número de neuronas por capa.

1) Desarrolle un código en Python basado en el PRAM mostrado para distinto número de hilos (p=1,2,4,8,16,32), y número de datos de muestra, dado por el parámetro n_samples (5000, 10000, 20000, 40000) y genere las gráficas correspondientes.

2) Registro del desarrollo del código en por lo menos 3 pasos (versiones beta)

3) Compare gráficamente los resultados con la expresión **teórica** de complejidad en paralelo. Debe superponer la curva teorica a la experimental encontrando los factores de proporcionalidad que conviertan la expresión bigO en tiempos de ejecución en segundos

4) Calcule el **speedup** y **eficiencia** y haga un análisis de **escalabilidad** basado en los resultados obtenidos. Determine experimentalmente, si E se mantiene constante

5) Calcule la velocidad en **FLOPs** necesarios para resolver el modelo planteado. Considere el calculo de FLOPs según: $FLOPs = 6 \cdot E \cdot n \sum_{l=1}^{L} n_{i-1} n_l$, donde $n_1, ..., n_{L-1}$ son las capas ocultas, $n_0 : n_{features}$, $n_L : n_{classes}$
E.g. $n_s = 200,000$, $[n_0, n_1, n_2, n_3] = [64, 128, 64, 4]$, $E = 200$
$\sum_{l=1}^{L} n_{i-1} n_l = 64 \cdot 128 + 128 \cdot 64 + 64 \cdot 4 = 16640$

FLOPs $= 6 \cdot 200,000 \cdot 200 \cdot 16640 \approx 4TFLOPs$
Proponga la cantidad de nodos en un cluster como Khipu necesarios para resolver el problema. ¿Necesita GPUs para este caso?

Algorithm 1: PRAM de entrenamiento paralelo de múltiples MLPs (Multilayer Perceptron)

Data: Datos (X, y), conjunto de semillas $S = \{0, 1, \ldots, p-1\}$, hiperparámetros $(\alpha, \eta, h, T_{\max})$
Result: Modelo M^* con mejor exactitud (accuracy)
`/* Fase 1: Distribución paralela (entrenamiento) */`
foreach $s \in S$ **do en paralelo**
  Inicializar el generador aleatorio con semilla s;
  Dividir los datos en (X_{train}, y_{train}) y (X_{test}, y_{test});
  Inicializar un modelo MLP M_s con:
  M_s = MLPClassifier(
    hidden_layer_sizes $= h$,
    learning_rate_init $= \eta$,
    regularización $= \alpha$,
    máx_iter $= T_{\max}$)

  Entrenar M_s sobre (X_{train}, y_{train});
  Evaluar $A_s \leftarrow$ accuracy$(M_s, X_{test}, y_{test})$;
  Reportar resultados locales (s, A_s) al proceso maestro;
end
`/* Fase 2: Reducción (selección del mejor modelo) */`
El proceso maestro recoge todas las parejas (s, A_s);
Selecciona la mejor semilla:

$$s^* = \arg\max_s A_s$$

Obtiene el modelo correspondiente $M^* = M_{s^*}$;
Guardar M^* en archivo (`mlp_model.joblib`);
return M^*

8 Indicaciones importantes para la entrega del proyecto

8.1 Informe de proyecto

Se pide realizar un informe de los resultados del proyecto, que contenga la siguiente información:

- **Nombre del proyecto**, y de los **integrantes**
- **Porcentaje de participación** de cada integrante en el proyecto (de 0 a 100 %, 100 es que ha contribuido en forma equitativa al resto)
- **Introducción:** Describa el proyecto elegido y la solución planteada
- **Método (punto 1 de la rúbrica):** Describa el algoritmo usado (PRAM) y/o el código
- **Resultados y análisis (punto 2 de la rúbrica):** Describa y analice los resultados obtenidos y plantee mejoras a la solución.
- **Bibliografía (punto 3 de la rúbrica):** Describa las fuentes utilizadas y referenciadas en el documento. Describa su impacto en el desarrollo del proyecto.

8.2 Entrega grupal

El proyecto se realizará en forma **grupal (2 o 3 integrantes)**
El proyecto se subirá a Canvas (Proyecto Laboratorio)
Presente el documento de proyecto en formato **.pdf**. Puede convertir un documento a pdf o usar una plantilla de Plantillas Latex . No olvide adjuntar el **código de programación** o **link a Github**.

9 Rúbrica

Se utilizará la siguiente rúbrica para la calificación del proyecto

Criterio	Logrado	Parcialmente Logrado	No Logrado
1. Desarrollo de código/PRAM	El diseño del algoritmo y código es adecuado (`4 pts`)	Existen algunos errores menores en el diseño del algoritmo o funcionamiento del código (`3 pts`)	Existen muchos errores en el diseño del algoritmo y/o funcionamiento del código (`0pts`).
2. Resultados y Análisis	Desarrolla correctamente las preguntas de análisis de resultados (`4 pts`)	Desarrolla parcialmente las preguntas de análisis de resultados (`2 pts`).	Desarrolla deficientemente las preguntas de análisis de resultados (`0pts`).
3. Uso correcto de fuentes externas	Justifica el uso de fuentes externas (publicaciones, charlas, IA) (`3 pts`)	Justifica solo parcialmente el uso de fuentes externas (`1.5 pts`).	No justifica el uso de fuentes externas (`0pts`).
4. Presentación escrita (informe) y trabajo en grupo **	Describe los puntos del informe en forma ordenada y satisfactoria (`3 pts`)	No describe en forma adecuada todos los puntos del informe (`2 pts`).	No describe en forma adecuada ningún punto del informe (`0pts`).
5. Presentación oral	Muestra dominio del contenido del proyecto durante presentación (`6 pts`)	No se demostró un dominio total del contenido del proyecto durante la presentación (`2 pts`).	No se presentó conocimiento del proyecto durante la presentación (`0pts`).

: se aplicará una **penalidad de 2 puntos si no se desarrolla el proyecto en formal grupal