Computación Paralela y Distribuída
Proyecto

Prof. José Fiestas
jfiestas@utec.edu.pe
UTEC

Instrucciones para desarrollar el proyecto:

- grupos de 3 estudiantes
- se presentan dos tipos de proyecto: **(A)** Desarrollo de código en paralelo y **(B)** Análisis de Performance
- debe escoger uno de los 6 proyectos propuestos. También puede proponer un nuevo proyecto de su interés. La calificación será sobre 20
- la entrega consiste en: el proyecto programado en C/C++, y un informe en LaTeX
- el informe escrito debe contener capítulos correspondientes a: **Introducción** (con la descripción del problema), **Método** (descripción del procedimiento), **Resultados** (presentación de los logros del proyecto) y **Conclusiones** (resumen y análisis de resultados.)
- la presentación oral consistirá de 3 minutos por integrante, que escogerá algún tema del proyecto, utilizando una presentación o el mismo informe, y una pregunta final a cada estudiante.
- Entrega: miércoles 21.07, por Gradescope

Se utilizará la siguiente rúbrica para le calificación del proyecto

Criterio	Puntaje
Desarrollo de código/PRAM	6.0
Optimización	4.0
Presentación escrita (informe)	6.0
Presentación oral	4.0
Total	20.0

---

1 RngStreams: generador random (Proyecto tipo B)

Uno de los principales objetivos de la generación de números random en paralelo, es lograr la independencia de procesos al generar éstos. Esto colind con el paralelismo ideal, pero es uno de los fundamentos principales de modelos estocásticos.

Los distintos métodos incluyen:
a) **Random spacing**: donde los procesos se inicializan en forma random a través de distintos seeds (E.g. Mersenne Twister (MT) generator)
b) **Sequence splitting**: dividiendo una secuencia en bloques que no se superponen
c) **Cycle division**: donde el período de un generador se divide en segmentos en forma determinística (Combined Multiple recursive generator (CMRG) with cycle division: RngStreams package)
d) **Parametrización**: los parámetetros de un generador se modifican para producir distintos streams (Multiplicative lagged Fibonacci generator with parametrization: SPRNG package)

El método MRG garantiza que los streams no coincidan. Este puede usarse en forma combinada (CMRG), con mayores ventajas en la generación de random. Son muy estables y estadísticamente seguros. El paquete RngStreams puede ser encontrado aqui [1]
El desarrollo de la solución del problema debe contener lo siguiente:
a) Elegir un caso de aplicación de las presentadas en el paper adjunto y elaborar un PRAM del mismo
b) Determinar la complejidad teórica del código y compararla con mediciones de tiempo, para distinto número de procesos y tamaño del problema
c) Medir la velocidad del algoritmo, y representarla en gráficas. Analizar la escalabilidad del software
d) **Optimización:** Desarrollar un software de análisis de performance que se adapte al código utilizado y permita obtener métricas que faciliten el análisis de escalabilidad

Bibliografía
[1] http://www.iro.umontreal.ca/ lecuyer/myftp/streams00/
[2] https://arxiv.org/pdf/1403.7645.pdf

2 Expresiones regulares (Proyecto tipo A)

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 de la solución del problema debe contener lo siguiente:
El desarrollo de la solución del problema debe contener lo siguiente:
a) Desarrollar un código en C++, correspondiente al paradigma apropiado de paralelismo
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
d) **Optimización:** Desarrolle un software que realice el análisis de tiempos en forma automática

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

3 TSP (Proyecto tipo A)

Resuelva el problema del agente viajero en paralelo utilizando el método *branch and bound*
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.
Según el método de búsqueda primero en profundidad (DFS en inglés), se consideran las posibles ciudades que se pueden visitar desde una ciudad de partida. Asi sucesivamente, y calcular el camino mínimo de todas las posibilidades. Por ello, para n ciudades, obtenemos una cantidad total de caminos de (n-1)!
El algoritmo recursivo correspondiente es el siguiente:

```
DFS(camino){
if (camino tiene longitud n) {
.       if (camino es el mejor camino) {
.               mejor_camino = camino
.       }
.       else: {
.               for (para todos los caminos aún no recorridos i)
.                       DFS(camino ∪ i)
.               }
.      }
}
```

En el que **camino** es una estructura que contiene las ciudades a visitar.
En **branch and bound** se verifica si el camino encontrado hasta el momento, es mayor que el mejor camino encontrado. En caso lo sea, se detiene la búsqueda por ese camino.

```
BB(camino){
if (camino tiene longitud n) {
.       if (camino es el nuevo mejor camino) {
.               mejor_camino = camino
.       }
.       else: {
.               for (para todos los caminos aún no recorridos i)
.                       if (camino ∪ i es menor que el actual mejor camino )
.                               BB(camino ∪ i)
.               }
.      }
}
```

En su versión no-recursiva el algoritmo realiza lo siguiente:

```
camino={0}
push(camino)
while pop(camino){
if (camino tiene longitud n) {
.          if (camino es el nuevo mejor camino) {
.                    mejor_camino = camino
.          }
.          else: {
.                    for (para todos los caminos aún no recorridos i)
.                              if (camino ∪ i es menor que el actual mejor camino )
.                                        push(camino ∪ i)
.                    }
.        }
}
```

Para su implementación se utiliza una pila (stack), que se inicializa con la ciudad **0**. En cada iteración se retira la ciudad del tope de la pila. Si el camino tiene longitud n se evalua, de lo contrario se añaden ciudades si la longitud del camino no es mayor que el del mejor actual. El programa termina cuando la pila está vacía.

Considere el siguiente caso:
Una agencia de transporte de materiales, quiere optimizar sus operaciones en Lima. La agencia trabaja en los siguientes distritos: Lima Centro, Lince, Miraflores, Barranco, Rimac, Los Olivos, La Molina, La Victoria, Magdalena, San Borja.
Se necesita elaborar un software en paralelo que resuelva el problema TSP entre estos distritos de Lima. Para ello debe elaborar una matriz con distancias entre estos. Puede tomar distancias referenciales (e.g. desde Google Maps).

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

El desarrollo de la solución del problema debe contener lo siguiente:
a) Desarrollar un código en C++, correspondiente al paradigma apropiado de paralelismo
b) Registro del desarrollo del código en por lo menos 3 pasos (versiones beta). En caso utilice una fuente externa, mostrar la validación del código
c) Utilice la matriz de distritos de Lima para implementar el problema y añada un factor adicional de costo, e.g. combustible, a los caminos trazados.
d) **Optimización:** Permita que el software sea interactivo para poder ingresar dos o mas ciudades y encontrar el camino óptimo entre ellas

4 Colisión galáctica (Proyecto tipo B)

Simulaciones de colisión geláctica son unas de las más importantes para poder predecir teóricamente el comportamiento de dos galaxias en cercanía y con probabilidad de colisión en el futuro.

El modelo requiere conjuntos de objetos (estrellas) de una cantidad muy grande, billones de estrellas en la realidad, que exigen una simulación lo más realistica posible, pero cuyos resultados sean accesibles en un tiempo mesurado.

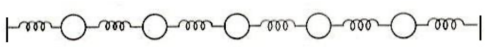
Esta propuesta se basa en el trabajo hecho por el grupo Astrofísica de la Universidad de Heidelberg, Alemania. Para ello, se cuenta con el software PhiGPU [1].

El desarrollo de la solución del problema debe contener lo siguiente:
a) Elegir un caso de aplicación de las presentadas en el paper adjunto y elaborar un PRAM del mismo
b) Determinar la complejidad teórica del código y compararla con mediciones de tiempo, para distinto número de procesos y tamaño del problema
c) Medir la velocidad del algoritmo, y representarla en gráficas. Analizar la escalabilidad del software
d) **Optimización:** Desarrollar un software de análisis de performance que se adapte al código utilizado y permita obtener métricas que faciliten el análisis de escalabilidad

Bibliografía
[1] phiGPU code
[2] Paper2011

5 Oszilador Armónico acoplado (Proyecto tipo A)

Analice la dinámica de una cadena que representa un oscilador armónico acoplado de N partículas (átomos). Cada átomo está acoplado a su átomo vecino por una constante elástica acoplada D.



La ecuación clásica de movimiento es:
$mx.. = -D(x_i - x_{i+1}) - D(x_i - x_{i-1}) = -D(2x_i - x_{i+1} - x_{i-1})$

Donde x_i representa la desviación de la partícula i de su posición de reposo. Los extremos de la cadena estan fijos a $x_0 = x_n =$ constante, con las relaciones $\sqrt{m}x_i(t) = v_i exp(iwt)$

El cálculo de las vibraciones del sistema, se reduce al cálculo de los valores propios y vectores propios de la matriz tridiagonal
$M_{ij} = \frac{D}{\sqrt{m_i m_j}}(2\delta_{ij} - \delta_{i,j+1} - \delta_{i,j-1})$

1. Analice el caso de un defecto en la cadena, en el que $m_i =$ m = constante, para $n \neq n_0$, y $m_n = 100$ m para $n = n_0$
Escoja valores de tiempo y unidades de longitud adecuadas, para que m = D = 1
Escriba un programa en paralelo que resuelva la matriz y calcule los valores y vectores propios para N=99, n_0=60 y n_0=30 con ayuda de la librería tqli de Numerical Recipees [1]

2. Escoja condiciones iniciales $x_i(0)$=1 (i=n_0) y $x_i(0)$=0 ($i \neq n_0$) y grafique $x_i(t)(i = n_0)$ para un intervalo adecuado de tiempo.
Nota: las condiciones iniciales están representadas como una superposición de las vibraciones individuales.
$y_i = \sum_{n=1}^{N-1} a_n v_n e^{iw_i t} + b_n v_n e^{-iw_i t}$
O en valores reales
$y_i = \sum_{n=1}^{N-1} a_n v_n cos(w_i t) + b_n v_n sin(w_i t)$
El desarrollo de la solución del problema debe contener lo siguiente:
a) Desarrollar un código en C++, correspondiente al paradigma apropiado de paralelismo. Desarrolle puntos 1 y 2 de la descripción. Elevar la dimensión de la cadena lo suficiente para obtener mediciones significativas de tiempos
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
d) **Optimización:** Desarrolle un software que realice el análisis de tiempos en forma automática

Bibliografía

[1] https://www.cec.uchile.cl/cinetica/pcordero/MClibros/NumericalRecipesinC.pdf

6 Simulación de plasma (Proyecto tipo B)

En este proyecto se trata de analizar el performance de un código Particle in Cell (PIC) que es usado para simulaciones del plasma en la ionosfera y sus posibles efectos en el clima. Los modelos teóricos se basan en la solución de un sistema de ecuaciones que describen la función de distribución del sistema, conocida como la ecuación de Vlasov Esta ecuacion describe la dinámica del plasma, considerando

$$\left(\partial_t + \frac{\mathbf{p}}{m_s\gamma}\cdot\nabla + \mathbf{F}_L\cdot\nabla_{\mathbf{p}}\right) f_s = 0\,,$$

efectos eléctricos y magnéticos, de acuerdo a sus densidades Como resultado,

$$\begin{aligned}\rho(t,\mathbf{x}) &= \sum_s q_s \int d^3p f_s(t,\mathbf{x},\mathbf{p})\,,\\ \mathbf{J}(t,\mathbf{x}) &= \sum_s q_s \int d^3p\, \mathbf{v} f_s(t,\mathbf{x},\mathbf{p})\,.\end{aligned}$$

modelos físicos como el mencionado, logran elaborar mapas de la ionosfera como el siguiente Permitiendo realizar pronósticos de efectos en la evolución del clima

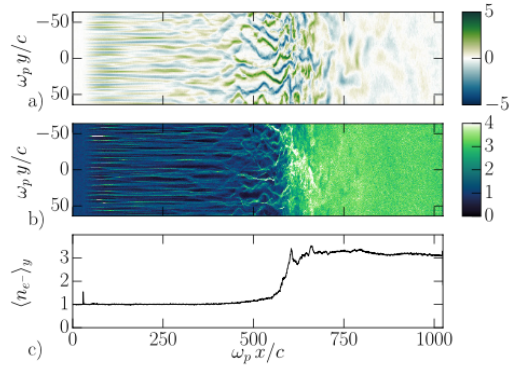


Figure 20: Snapshot at $t = 1000\,\omega_p^{-1}$. a) Weibel generated magnetic field B_z in units of $B_0 = m_e\,\omega_p/e$. b) Electron density in units of n_0. c) Electron density (in units of n_0) averaged along the y-direction.

de una región dependiendo de su latitud y condiciones propias del lugar.

El proyecto no requiere el entendimiento de la física detras de estos fenómenos. Se trata de analizar el performance y escalabilidad del código PIC Smilei, adjunto, cuyo diseño se ilustra en el siguiente gráfico

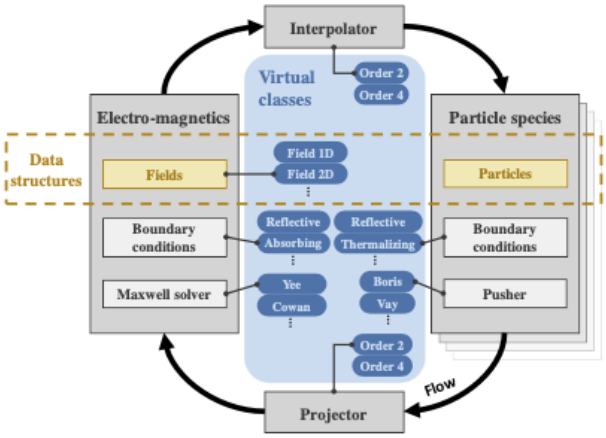


Figure 2: C++ flow, classes and data structure in SMILEI.

El desarrollo de la solución del problema debe contener lo siguiente:
a) Elegir un caso de aplicación de las presentadas en el paper adjunto y elaborar un PRAM del mismo
b) Determinar la complejidad teórica del código y compararla con mediciones de tiempo, para distinto número de procesos y tamaño del problema
c) Medir la velocidad del algoritmo, y representarla en gráficas. Analizar la escalabilidad del software
d) **Optimización:** Desarrollar un software de análisis de performance que se adapte al código utilizado y permita obtener métricas que faciliten el análisis de escalabilidad

Bibliografía

[1] https://smileipic.github.io/Smilei/index.html
[2] https://www.sciencedirect.com/science/article/pii/S0010465517303314?via%3Dihub