Metodología de la Programación Paralela
Curso 2024/25

Práctica 3:

Paralelización Híbrida con MPI+OpenMP

Modelado y Auto-Optimización del Algoritmo Paralelo

En esta práctica se va a realizar una versión híbrida que combine paralelismo MPI+OpenMP en el algoritmo evolutivo proporcionado en la Práctica 0. Se analizarán las prestaciones obtenidas al variar el número de procesos MPI y el número de hilos OpenMP que pone en marcha cada proceso. Para cada tipo de paralelismo se hará uso de la implementación paralela de las Prácticas 1 y 2 con la que se han obtenido las mejores prestaciones. Además, se llevará a cabo un estudio sobre el modelado y auto-optimización de algunas funciones del algoritmo evolutivo en la versión paralela para memoria compartida y mediante paso de mensajes.

Esta práctica representa el 15% de la calificación de la asignatura.

La fecha de entrega de la práctica será el **29 de noviembre a las 23:55**. Entregas posteriores no se tendrán en cuenta. La entrega se realizará a través de la tarea habilitada en el Aula Virtual, adjuntando un fichero .zip o .tgz que contenga:

- El código fuente paralelizado, los ficheros *Makefile* y *run.sh* para su compilación y ejecución, así como otros ficheros de salida o con datos, que se podrán organizar en diferentes carpetas (una para cada cuestión). En el caso de que en alguna cuestión se requiera varias implementaciones de la misma rutina, se podrá entregar un único código fuente para esa cuestión con las distintas implementaciones en forma de comentarios. Además, el código ha de incluir los comentarios necesarios para ayudar a comprender los cambios realizados. Debido a su elevado tamaño, **no se deberán incluir los ficheros de entrada generados en `/input/`**, solo **indicar en la memoria qué configuración de parámetros se ha usado en cada experimento**.

- La hoja de cálculo (*solver.ods*) donde se haya realizado el análisis sobre modelado y auto-optimización de la Parte 2 de la práctica.

- Un documento en formato .pdf que describa cómo se han resuelto las cuestiones propuestas, justificando las decisiones de paralelización tomadas en cada caso. El código que se muestre debe coincidir con el que se entregue. Asimismo, el documento ha de incluir una sección en la que se indique cómo ha sido la coordinación, el reparto del trabajo entre los miembros del grupo y el tiempo dedicado.

Se permite el uso, no obligatorio, de la herramienta *GitLab/GitHub* como repositorio software para ir almacenando los cambios realizados durante el desarrollo de la práctica. Si en alguna cuestión es necesario comparar varias versiones paralelas, se crearán diferentes ramas (una para cada versión). La rama máster contendrá la versión final del código con la que se obtienen las mejores prestaciones. Se podrán entregar carpetas con las distintas versiones del código y si se incluyen también los *.git*, habrá que comprobar su tamaño, de modo que este se reduzca al mínimo, entregando solo lo imprescindible.

El **sistema computacional** utilizado, que debe tener **como mínimo 6 núcleos físicos**, se debe describir con: `lstopo -f -p --no-legend --no-io sistema.svg`.

En aquellos casos en los que sea necesario mostrar la evolución de las prestaciones, se puede estructurar la información en tablas, incluyendo una leyenda que indique la instancia del problema sobre la que se realizan los experimentos, así como el resto de parámetros de entrada:

n_procesos | *n_hilos* | *t_ejec* | *speed-up* | *eficiencia*

Tabla 1. Leyenda para la tabla considerando n, m, n_gen, tam_pob.

Se debe indicar el algoritmo utilizado para tomar el tiempo secuencial a la hora de calcular el speed-up. Podría ser el algoritmo secuencial o el algoritmo paralelo con 1 proceso total y 1 hilo en cada función. La eficiencia se calcularía como el speed-up dividido por el número total de elementos de proceso, EP, puestos en marcha entre los dos niveles (EP = *n_procesos* · *n_hilos*).

Se recomienda la creación de gráficos (con su leyenda correspondiente) a partir de las series de valores más relevantes de las tablas, en los que se muestre la evolución de las métricas al variar el número de hilos y de procesos.

En general, para obtener unos resultados más fiables es aconsejable, en cada experimento, realizar la media de varias ejecuciones.

La práctica está dividida en dos partes: versión híbrida y modelado. En cada una de ellas se plantean una serie de cuestiones que recogen las tareas a realizar.

Parte 1. Implementación de Paralelismo Híbrido (MPI + OpenMP)

Se trabajará con una versión paralela híbrida del algoritmo, combinando el modelo de islas (con MPI) de la Práctica 2 y algunas de las directivas OpenMP estudiadas en la Práctica 1. Por tanto, se pondrán en marcha varios procesos que harán evolucionar cada subpoblación en paralelo, produciéndose migraciones periódicas según lo indicado en el modelo de islas. A su vez, cada proceso ejecutará código paralelo en memoria compartida, para lo cual se utilizarán de forma apropiada directivas OpenMP en determinadas zonas del código.

Cuestión 1. Con el fin de obtener un código más eficiente en la parte de memoria compartida y poder maximizar el proceso de mejora de la población de individuos: reemplazar las llamadas a la función *rand* por *rand_r* (que es "*thread safe*") para evitar, por un lado, que se genere la misma secuencia de valores aleatorios en diferentes hilos y, por otro, que se pierda paralelismo durante la generación de dichos valores debido a la secuencialidad que introduce la función *rand*.

NOTA: seguir el procedimiento indicado en la Práctica 1 y, si ya se ha implementado la generación aleatoria con *rand_r* en esa práctica, fusionar este apartado con la Cuestión 2.

Cuestión 2. Implementar un algoritmo híbrido, tomando como referencia las directivas OpenMP y primitivas MPI que arrojaron mejores resultados en las prácticas 1 y 2. Se deberán utilizar, por tanto, las funciones que se considere más adecuadas para obtener un código optimizado y que permita obtener las mejores prestaciones. Ejecutar el código variando el número de procesos y de hilos, y comparar los resultados obtenidos respecto a la versión secuencial utilizando las métricas de rendimiento de la *Tabla 1*.

Además, se deberá analizar la mejora obtenida en las prestaciones, considerando solo tiempo de ejecución y tiempo de ejecución + fitness (ver apartado (1) de Observaciones), respecto al algoritmo secuencial y también respecto a la mejor versión paralela obtenida en las prácticas anteriores usando el mismo sistema computacional para todas las versiones.

Si se ejecutó sobre máquinas distintas en prácticas anteriores, volver a tomar tiempos y fitness en un único sistema para comparar las distintas implementaciones.

De manera opcional, considerar las indicaciones del apartado (2) de Observaciones.

Parte 2. Modelado y Auto-Optimización del Algoritmo Paralelo

Las prácticas realizadas hasta ahora se han centrado en implementar diferentes versiones paralelas del algoritmo evolutivo propuesto en la Práctica 0 mediante la aplicación de los modelos de programación paralela existentes para memoria compartida y paso de mensajes. En los estudios experimentales correspondientes, se ha variado el valor de determinados parámetros que afectan al rendimiento (como el número de hilos OpenMP o el número de procesos MPI) con el fin de reducir el tiempo de ejecución.

Es necesario seleccionar de forma adecuada el valor de dichos parámetros si se quiere optimizar el rendimiento del algoritmo paralelo. En memoria compartida esta tarea se lleva a cabo mediante la obtención de un modelo del tiempo de ejecución para cada rutina, estableciendo el número de hilos a usar en determinados bucles y en diferentes niveles de paralelismo. En memoria distribuida, en cambio, el modelo de tiempo se extiende a todo el programa paralelo, incluyendo tanto computación como comunicaciones.

Modelo en memoria compartida.

La Figura 1 muestra un esquema del proceso de auto-optimización que se podría aplicar a rutinas en memoria compartida para obtener el mejor valor de los parámetros. Este proceso consta de tres fases:

Fase 1: Diseño. Se crea la rutina y se obtiene su modelo de tiempo de ejecución teórico. Dado que en memoria compartida se han identificado dos tipos de paralelismo, se pueden usar dos modelos para el tiempo de ejecución, uno para rutinas con un nivel de paralelismo y otro para paralelismo anidado de dos niveles.

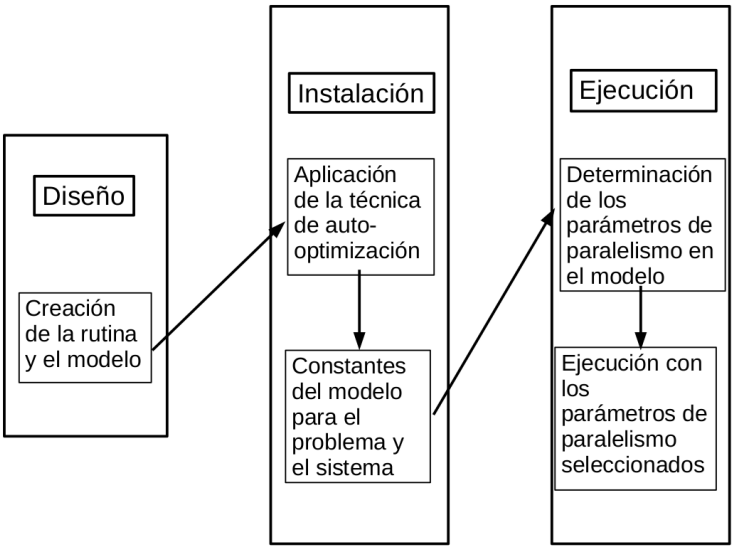

Figura 1. Fases del proceso de auto-optimización para rutinas en memoria compartida.

Por ejemplo, en el algoritmo evolutivo se podría modelar la generación de la población inicial (bucle *for* al inicio de *aplicar_mh*) con un número de elementos (individuos) *NE*:

$$t_{un\_nivel} = \frac{k_{s1} \cdot NE}{h} + k_h \cdot h \qquad \text{(ec. 1)}$$

donde k_{s1} representa el coste de generar un individuo; k_h el coste de generar un hilo; y *h* el número total de hilos.

Si en la generación inicial de individuos (dentro del mismo bucle *for*) se incluye el cálculo del fitness para *NE* individuos y un tamaño del subconjunto $|B| = m$ elementos, la rutina de dos niveles resultante se podría modelar de la siguiente forma:

$$t_{dos\_niveles} = \frac{k_{s2} \cdot NE \cdot |B|}{h_1 \cdot h_2} + k_{h,1} \cdot h_1 + k_{h,2} \cdot h_2 \qquad \text{(ec. 2)}$$

donde k_{s2} representa el coste de generar un elemento; $k_{h,1}$ y $k_{h,2}$ el coste de generar hilos en el primer y segundo nivel, respectivamente; y h_1 y h_2 el número de hilos de cada nivel.

Los modelos así obtenidos son simples, ya que no consideran aspectos relacionados con la arquitectura de la plataforma computacional, como la asignación de memoria o de hilos, facilitando su uso y permitiendo obtener resultados satisfactorios.

Fase 2: Instalación. Cuando se instala la rutina en un sistema concreto, hay que estimar el valor de los parámetros que dependen del sistema (k_{s1}, k_{s2}, k_h, $k_{h,1}$ y $k_{h,2}$). La estimación se puede hacer mediante experimentación (con cada función básica del algoritmo) con algunos valores de los parámetros metaheurísticos, del problema y de paralelismo (*NE* y *h* en la ecuación 1 para la generación inicial de elementos; y *NE*, $|B|$, y h_1, h_2 en la ecuación 2 para la generación de los elementos iniciales incluyendo el cálculo del fitness para cada individuo) y con un ajuste por mínimos cuadrados.

Fase 3: Ejecución. Durante la ejecución de la rutina, el valor óptimo de los parámetros de paralelismo (número de hilos en memoria compartida) se obtiene tras sustituir en el modelo teórico del tiempo de ejecución (ecuaciones 1 y 2) los valores de los parámetros algorítmicos con los que se está experimentando (parámetros metaheurísticos y del problema) y los valores de los parámetros del sistema estimados en la fase de instalación. El número de hilos con el que se obtiene el menor tiempo de ejecución teórico se obtendrá derivando la correspondiente ecuación tras sustituir en ella los valores de las constantes estimadas experimentalmente.

En la rutina que genera la población inicial, el número de hilos con el que se obtiene el menor tiempo de ejecución teórico (h_{opt}) se obtendría despejando *h* en la siguiente ecuación:

$$\frac{dt}{dh} = \frac{-k_{s1} \cdot NE}{h^2} + k_h = 0 \qquad \text{(ec. 3)}$$

En el caso de la generación de individuos con el cálculo del fitness dentro del mismo bucle, el número óptimo de hilos con paralelismo de dos niveles ($h_{1,opt}$ y $h_{2,opt}$) se obtendría al resolver el siguiente sistema de ecuaciones:

$$\frac{dt}{dh_1} = \frac{-k_{s2} \cdot NE \cdot |B|}{{h_1}^2 \cdot h_2} + k_{h_1} = 0 \qquad \text{(ec. 4)}$$

$$\frac{dt}{dh_2} = \frac{-k_{s2} \cdot NE \cdot |B|}{h_1 \cdot {h_2}^2} + k_{h_2} = 0 \qquad \text{(ec. 5)}$$

Una vez tenemos las herramientas necesarias, vamos a abordar el problema de modelado y auto-optimización en memoria compartida de nuestro algoritmo paralelo aplicado al problema MDP. Para ello, se plantean las siguientes cuestiones.

Cuestión 3. Modelo de Tiempo en **Memoria Compartida con 1 nivel de paralelismo**.

3.1 Obtención de Datos Experimentales

Debido al elevado tamaño de los ficheros de entrada que se necesitarían para realizar el proceso de modelado y auto-optimización de la **Parte 2 de la práctica**, se ha optado por **generar la matriz de distancias de forma directa para cada ejecución en memoria principal, sin generar ficheros de entrada**. Entonces, **se deberá sustituir el fichero *io.c* de la Práctica 0 por el nuevo fichero *io.c* disponible en la carpeta *datos/ejecucion/***. De este modo, en la Parte 2 de la práctica **solo habrá que ejecutar un *run.sh* en /src/,** especificando el tamaño de problema directamente los *makefile* correspondientes (ver NOTA al final de este apartado). **Esta modificación** en la generación de los datos de entrada se considerará **para todas las cuestiones de la Parte 2**.

a) Paraleliza el bucle *for* correspondiente a la generación inicial de individuos (*FOR_INI*) que incluye el cálculo del fitness para cada individuo dentro del mismo bucle *for*.

b) Modifica el código de la Práctica 1 para que el programa reciba dos nuevos parámetros: el número de hilos (*N_HILOS_INI*) para el bucle *FOR_INI* y el número de hilos (*N_HILOS_FIT*) para el bucle *for* (*FOR_FIT*) dentro de la función que realiza el cálculo del fitness.

c) Instrumentaliza el código para que permita medir el tiempo de ejecución del bucle *FOR_INI* al variar el número de hilos (por ejemplo, desde 1 hasta 20, con incrementos de 2: 1, 2, 4, 6, 8, …). En este caso no hay que paralelizar el bucle *FOR_FIT*.

d) Aplicar el algoritmo sobre una instancia del problema de tamaño n = 30000 y m = 28000 (u otros valores que consideres oportunos para tu sistema computacional de manera que, para una correcta visualización del gráfico, el tiempo de ejecución con 1 hilo se sitúe en el intervalo de 5 a 10 segundos). Fija el valor del parámetro *NE* en la ecuación 1, por ejemplo, *NE = 6* (igual al número de cores si el sistema es hexacore) y el número de iteraciones del algoritmo a 0 para que sólo se ejecute la parte de generación inicial de la población (para experimentar con la metodología presentada sólo nos interesa esta función, aunque el proceso real se ampliaría a todas las rutinas del algoritmo). Lanza 5 series de ejecuciones y toma la media de los tiempos experimentales obtenidos con cada número de hilos.

   NOTA: **Para realizar las ejecuciones de cada experimento se proporciona un *run.sh* y un *Makefile* dentro de la carpeta *datos/ejecucion/mc***, que puedes modificar para realizar los experimentos con otras configuraciones de parámetros adecuadas para tu sistema computacional. Consideraremos los siguientes parámetros de entrada: *n, m, n_gen, tam_pob, N_HILOS_INI, N_HILOS_FIT.* Para agilizar el proceso posterior de análisis, puedes hacer que la salida del programa sea solo una columna con los datos numéricos de tiempo.

3.2 Instalación de la Rutina (bucle *for*) con 1 Nivel de Paralelismo

a) Analiza la hoja de cálculo "*Modelo 1-Nivel Paralelismo*" del documento *datos/solver.ods*. En ella se presentan tiempos experimentales (*t_experimental*) frente al número de hilos, *h*, así como los correspondientes al modelo teórico (*t_teorico*) dado por la ecuación 1. Obviamente, debes reemplazar estos datos por tus tiempos obtenidos en el Apartado 3.1.

Como aproximación al ajuste por mínimos cuadrados se ha utilizado la herramienta "*Solucionador*" que proporciona *LibreOffice Calc*, que también permite realizar ajustes sobre ecuaciones no lineales (como es el caso de ecuación 1). Esta herramienta está accesible desde el menú *Herramientas → Solucionador (Solver)*. A continuación, dentro de *Opciones*, hay que escoger cualquier algoritmo no lineal como, por ejemplo, *DEPS Evolutionary Algorithm* (instalar en Ubuntu con: `apt-get install libreoffice-nlpsolver` si no está disponible en la versión básica de *Calc*), o el *Solver no lineal de cúmulos de LibreOffice (experimental)*.

Tras acceder a la herramienta se deben indicar dos valores:

- *Celda Objetivo*: será la suma de las diferencias entre *t_experimental* y *t_teorico* al cuadrado.
- *Cambiando Celdas*: las celdas donde se encuentran las constantes del sistema a determinar en nuestro modelo (k_{s1} y k_h)

A continuación, indicamos que el valor sea mínimo (obviamente las diferencias entre los tiempos experimentales y teóricos generados por el modelo deben ser mínimas) y hacemos clic en *Solucionar*.

En la misma hoja se puede ver un gráfico donde se representan ambos tiempos de ejecución para la generación inicial de individuos (*FOR_INI*), comprobando así la bondad del ajuste del modelo a los datos experimentales.

b) Instalar la Rutina con 1 Nivel de Paralelismo (obtener valores de constantes del sistema) a partir de los datos experimentales obtenidos en el Apartado 3.1.

3.3 Ejecución

a) Despeja el número de hilos óptimo (h_{opt}) de la ecuación 3 en función de *NE*, k_{s1} y k_h, donde los valores de k_{s1} y k_h han sido obtenidos en la instalación (Apartado 3.2)

b) Calcula el tiempo de ejecución que se obtendría con varios valores de *h* (p.ej.: 3, 5, 7, …) distintos a los usados en la obtención del modelo en los Apartados 3.1 y 3.2.

c) Realiza las correspondientes ejecuciones para ese número de hilos con *NE* = 6 (o el valor que hayas elegido para tu sistema) y comprueba la bondad del ajuste del modelo a los datos experimentales.

d) Varía el tamaño de la población dando valores a *NE* (por ejemplo 3, 10 u otros que estimes oportunos) y obtén el número de hilos óptimo (h_{opt}) en estos casos. De esta forma, al variar el tamaño del problema no necesitamos volver a determinar experimentalmente el número de hilos óptimo, sino que podemos hacer el cálculo directamente gracias al modelo de tiempos.

e) Ejecuta de nuevo la rutina *FOR_INI* con *NE = {3, 10}* y compara los resultados experimentales obtenidos con los arrojados por el modelo. ¿Son similares? ¿Ajusta bien el modelo?

Cuestión 4. Modelo de Tiempo en **Memoria Compartida con 2 niveles de paralelismo**.

En este caso hay que paralelizar los bucles *FOR_INI* y *FOR_FIT*. Recuerda hacer uso de la función *omp_set_nested(1)* que permite habilitar el uso de paralelismo anidado en OpenMP.

4.1 Instalar la Rutina con 2 Niveles de Paralelismo, es decir, obtener los valores de las constantes del sistema k_{s2}, $k_{h,1}$ y $k_{h,2}$ en la ecuación 2, donde se ha introducido el nuevo parámetro $|B| = m$ que es el tamaño del individuo. Se deben obtener datos experimentales usando la configuración de

parámetros del Apartado 3.1(d), variando el número de hilos en ambos niveles de paralelismo (h_1, h_2) como se sugiere en la hoja de cálculo "*Modelo 2-Niveles Paralelismo*" del documento s*olver.ods*.

4.2 Despeja el número de hilos óptimo $h_{1,opt}$ y $h_{2,opt}$ de las ecuaciones 4 y 5 en función de *NE*, |*B*|, k_{s2}, $k_{h,1}$ y $k_{h,2}$, donde las constantes han sido obtenidas en la instalación realizada en apartado anterior.

4.3 Calcula el tiempo de ejecución que se esperaría para varios valores de h_1 *y* h_2 (distintos a los usados para instalar el modelo) con la configuración de parámetros utilizada en 4.1.
Por otra parte, varía el tamaño de la población dando valores diferentes a *NE* y |*B*| y obtén el número de hilos óptimo $h_{1,opt}$ y $h_{2,opt}$ para cada configuración (no hay que lanzar ejecuciones).

Modelo con paso de mensajes.

Una vez descrita la metodología de modelado y auto-optimización para rutinas paralelas en memoria compartida, se va a abordar su equivalente para paso de mensajes.

Como hemos visto, para reducir el tiempo de ejecución es necesario seleccionar adecuadamente los valores de los parámetros de paralelismo. En el modelo de programación paralela mediante paso de mensajes, estos parámetros son el número de procesos, *p*, el número de elementos a migrar, *NEM*, y la frecuencia con la que migran elementos, *NGM*.

Siguiendo el esquema de la Figura 1, el primer paso (fase de diseño) consiste en obtener un modelo teórico del tiempo de ejecución total. En este caso, el tiempo se divide esencialmente en tres partes: computación (t_{cmp}), comunicaciones entre procesos (t_{cmc}) y ordenación de elementos (t_{ord}) en proceso 0:

$$t_{total} = t_{cmp} + t_{cmc} + t_{ord} \qquad \text{(ec. 6)}$$

El tiempo de computación (t_{cmp}) es inversamente proporcional al número total de procesos *p*:

$$t_{cmp} = \frac{k_{cmp}}{p} \qquad \text{(ec. 7)}$$

y k_{cmp} engloba todos los parámetros metaheurísticos y del problema.

Además, dado que habitualmente utilizan un elevado número de variables, el modelado de las comunicaciones se suele simplificar usando una función polinómica (por ejemplo, de grado 3) del número de procesos:

$$t_{cmc} = A \cdot p^3 + B \cdot p^2 + C \cdot p + D \qquad \text{(ec. 8)}$$

Un análisis de la complejidad computacional permite despreciar el término correspondiente al tiempo de ordenación t_{ord} en la ecuación 6. Si, además, suponemos que las comunicaciones no varían significativamente con el número de individuos a migrar y, por simplicidad, fijamos *NGM = 1*, tendríamos el siguiente modelo de tiempo:

$$t_{total} = \frac{k_{cmp}}{p} + A \cdot p^3 + B \cdot p^2 + C \cdot p + D \qquad \text{(ec. 9)}$$

Al igual que en memoria compartida, tras sustituir en el modelo teórico los valores de las constantes estimadas experimentalmente y por mínimos cuadrados en la fase de instalación (k_{cmp}, *A*, *B*, *C* y *D*), la obtención del número de procesos que proporciona el menor tiempo de ejecución teórico (p_{opt}) se llevaría a cabo despejando *p* de la siguiente ecuación:

$$\frac{dt_{total}}{dp} = \frac{-k_{cmp}}{p^2} + 3 \cdot A \cdot p^2 + 2 \cdot B \cdot p + C = 0 \qquad \text{(ec. 10)}$$

Ya hemos creado el modelo, ahora vamos a determinar los parámetros del mismo y a realizar la auto-optimización en memoria distribuida de nuestro algoritmo paralelo aplicado al problema MDP. Para ello, se plantea la siguiente cuestión.

Cuestión 5. Modelo de Tiempo en **Memoria Distribuida**.

5.1 Añade una nueva hoja de cálculo al documento s*olver.ods*, de forma que permita realizar los cálculos correspondientes a la fase de instalación del modelo de tiempo para paso de mensajes. Comprueba experimentalmente que el tiempo de ordenación t_{ord} es realmente despreciable en la ec. 6. Para ello, mide el tiempo de un ciclo completo del algoritmo que incluya computación paralela en las islas, una migración y una ordenación.

5.2 Obtén los valores de las constantes del sistema en la ecuación 9 (k_{cmp}, *A*, *B*, *C* y *D*) al variar el número de procesos con los datos de entrada proporcionados en el *Makefile* del directorio *datos/ejecucion/pm*. Consideraremos los siguientes parámetros de entrada: *n, m, n_gen, tam_pob.*

Para facilitar el proceso de modelado, puedes variar el número de procesos y ajustar los parámetros de entrada como consideres mejor para realizar los experimentos.

5.3 Calcula el tiempo de ejecución teórico para diferentes valores de *p* y resuelve la ecuación 10 para encontrar p_{opt} en función de las constantes del sistema (puedes usar algún método numérico para resolverla y utilizar para ello el propio *Solucionador*).

Cuestión 6. Realizar el informe final incluyendo las conclusiones generales y valoración personal sobre el trabajo realizado.

Observaciones

1) Para comparar las prestaciones del algoritmo de una forma más útil entre implementaciones, secuencial, MC, PM e Híbrida, puede ser interesante comparar no solo el tiempo, sino un indicador *I* (por ejemplo, *I = fitness / tiempo*) que englobe tiempo y fitness. Teniendo en cuenta que los resultados deben ser obtenidos al aplicar el algoritmo sobre la misma instancia de problema.

   Existen otras opciones más simples que no requieren el uso de un indicador:
   - Una, podría ser medir el fitness alcanzado fijando un tiempo de ejecución razonable, igual para todas las implementaciones.
   - Otra, ejecutar el algoritmo secuencial y obtener fitness (y tiempo) con criterio de parada por número de iteraciones o dado por porcentaje de mejora entre iteraciones para, después, comparar los tiempos obtenidos con las versiones paralelas necesarios para alcanzar el fitness dado por el algoritmo secuencial (o algún otro valor de fitness menor que se considere más adecuado y representativo).

   De cualquier forma, se pueden representar los resultados en forma de tablas o gráficas que recojan valores de variables como tipo de algoritmo (y elementos de proceso), fitness, tiempo, o indicador *I* en su caso, para cada configuración de parámetros de entrada utilizada.

2) Se podría utilizar la información obtenida en la Parte 2 de esta práctica para realizar una mejor optimización del código híbrido de la Parte 1. Así, podría resultar interesante realizar el

modelado y auto-optimización de todas y cada una de las rutinas susceptibles de ser paralelizadas en memoria compartida y, además, del algoritmo completo de paso de mensajes. De esta forma se podría comparar los resultados experimentales con los teóricos y elegir la combinación de parámetros de paralelismo (hilos y procesos) más adecuada en cada región del código y globalmente.

3) Para el modelo con dos niveles en memoria compartida, como el rango de hilos con el que se experimenta es muy grande, el modelo trata de ajustar en todos los sectores del dominio, pero a costa de un ajuste regular o malo en general.

   En este sentido, para tomar decisiones lo más fundamentadas posible, se puede considerar el estudio previo realizado sobre el bucle del fitness con *reduction* de la Práctica 1: ¿merecía la pena poner en marcha muchos hilos para este bucle? Si la respuesta es negativa, entonces procede variar h_2 solo en un intervalo reducido, con la idea de que el modelo de 2-niveles sea lo más preciso posible abarcando todo el dominio de h_1, h_2.

   Además, a la hora de determinar las constantes del modelo, también hay que tener en cuenta que, si el valor de la/s constante/s resulta negativo y no está muy alejado de cero, por ejemplo, -0.01, quizás sea suficiente con poner la restricción de valores >= 0 como *Condiciones Limitantes* en el *Solucionador*, comprobando, en todo caso, si el modelo ajusta bien de este modo.

   Otra opción para mejorar este ajuste, sería considerar la definición de dos (o más) modelos cada uno de ellos para regiones específicas del dominio de hilos h_1 y h_2. Por ejemplo, con valores reducidos de h_1 como 1, 2, o 3 y variando más ampliamente h_2. O para otra zona del dominio, donde h_2 valga 2, o 3, y variar ampliamente h_1, etc. Se trataría de ir probando combinaciones. De este modo, en lugar de un solo modelo pobre en cuanto a bondad del ajuste que lo abarca todo, se puede tener varios modelos locales (más trabajo que realizar para el diseñador) pero con ajustes específicos mejores.

4) En la Cuestión 5, existe la posibilidad de crear otro modelo del tiempo de comunicaciones diferente al polinómico de la ec. 8, con lo que habría que experimentar y justificar esta elección en base a la bondad de los ajustes alcanzados por el nuevo modelo planteado.