Metodología de la Programación Paralela
Curso 2024/25

Práctica 2:

Paralelización con MPI

En esta práctica se va a paralelizar el algoritmo evolutivo proporcionado en la Práctica 0 mediante MPI, estándar *de facto* para la programación paralela mediante paso de mensajes en sistemas con memoria distribuida. Para ello, se proponen una serie de cuestiones que se han de realizar de forma progresiva analizando en cada momento la mejora que se obtiene en el rendimiento. Esta práctica representa el 15% de la calificación de la asignatura.

La fecha de entrega de la práctica será el **5 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 y los ficheros *Makefile* y *run.sh* para su compilación y ejecución. 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**.

- Un documento en formato *.pdf* que describa cómo se han resuelto las cuestiones propuestas, justificando las decisiones de paralelización tomadas en cada caso y analizando los resultados obtenidos. 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 se entregue solo lo imprescindible reduciendo su tamaño en MB al mínimo.

Describir el sistema computacional utilizado 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. El formato de cada tabla podría ser el siguiente:

n	*m*	*n_gen*	*tam_pob*	*n_procesos*	*t_ejec*	*speed-up*	*eficiencia*

Se recomienda la creación de gráficos 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 procesos.

Algoritmo con paso de mensajes. El algoritmo paralelo se implementará siguiendo un modelo de islas (Figura 1). Cada proceso (incluido el maestro) trabajará con su propia isla (subconjunto de la población inicial) y llevará a cabo de manera independiente la evolución de su subpoblación. Se hará uso de una variable, NGM, que determinará cada cuántas generaciones se ha de producir la migración de NEM individuos de cada isla, la mezcla en el *proceso 0*, su ordenación y el posterior envío de parte de la población (los mejores, por ejemplo) a cada isla para continuar el proceso evolutivo.

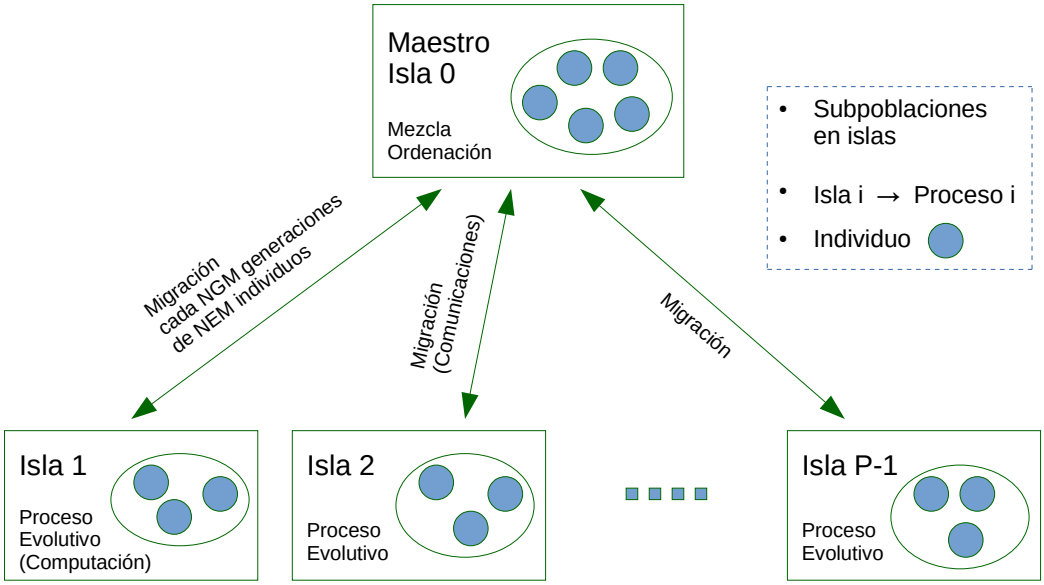

Figura 1: Modelo de islas.

A continuación, se muestran las cuestiones a realizar indicando de forma orientativa su distribución en cada semana.

Hay que tener en cuenta que **se proporciona**, anexo, **un fichero *derivados_mpi.c*** con la función **que permite crear el tipo de datos derivado MPI** para el envío y recepción de datos **de tipo *Individuo***, que será el **usado en las comunicaciones implementadas en las diferentes cuestiones**, **salvo que se indique otra cosa**. Para que MPI pueda utilizar el tipo *Individuo*, es necesario modificar el campo *array_int* del tipo de datos definido en el fichero *mh.h* para que sea un array de longitud estática (con tamaño máximo suficiente para los problemas concretos sobre los que se aplique el algoritmo).

1ª Semana

Cuestión 1.

a) Indicar qué pasos habría que llevar a cabo para paralelizar el algoritmo con MPI (no es necesario indicar las primitivas de comunicación). Incluir un esquema en pseudocódigo que refleje el patrón de computación y comunicaciones entre procesos en el modelo de islas, desde la lectura inicial de parámetros de entrada y su distribución, el intercambio de información entre procesos durante el desarrollo del algoritmo, la computación en paralelo, y la comunicación final de resultados.

b) Las primitivas como *MPI_Send* para comunicaciones con paso de mensajes pueden enviar arrays de datos del mismo tipo a condición de que estos estén contiguos en memoria. El código actual implementa la población como un array de punteros a *Individuo* con lo que los datos pueden estar en posiciones no contiguas de memoria. Como consecuencia se puede producir un error en el envío de poblaciones o subpoblaciones de individuos. Modifica el código como consideres mejor para solucionar este inconveniente, teniendo en cuenta que puedes cambiar de población de **Individuo* a *Individuo* o reservar memoria dinámica auxiliar contigua y apuntar a ella para crear la población.

Cuestión 2. Paralelizar el algoritmo evolutivo utilizando primitivas de **comunicación síncrona**. Hacer uso de las constantes *MPI_ANY_SOURCE* o *MPI_STATUS_IGNORE*, si se considera conveniente, para evitar que el *proceso 0* quede ocioso durante la recepción de mensajes del resto de procesos. Analizar las prestaciones que se obtienen al variar el número de procesos. Tener en cuenta el punto (3) en Observaciones para determinar los valores de NEM y NGM.

2ª Semana

Cuestión 3. Modificar la cuestión anterior para que el envío y la recepción de datos se realice mediante el uso de las funciones *MPI_Pack* y *MPI_Unpack* para **empaquetamiento de datos**, explicando si se debe usar el tipo derivado *Individuo*. Presentar también resultados de los experimentos realizados.

Cuestión 4. Paralelizar el algoritmo evolutivo utilizando primitivas de **comunicación asíncrona**. Analizar las prestaciones obtenidas al variar el número de procesos.

3ª Semana

Cuestión 5. Paralelizar el algoritmo evolutivo utilizando primitivas de **comunicación colectiva**. Analizar las prestaciones obtenidas al variar el número de procesos.

Cuestión 6. Elegir la mejor configuración de parámetros y tipo de comunicaciones para paralelizar la totalidad del algoritmo. Justificar las decisiones tomadas. De este modo, se pretende tener un código paralelo óptimo en memoria distribuida que se podrá usar para comparar con el código secuencial, y paralelo en memoria compartida e híbrido.

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

Observaciones

1) El programa principal (*main.c*) tiene que modificarse para que permita inicializar y finalizar el entorno de ejecución de MPI, así como llevar a cabo la toma de tiempos de forma adecuada, por ejemplo, si es necesario, mediante barreras de sincronización (*MPI_Barrier*).

2) El procedimiento para implementar el algoritmo de paso de mensajes queda a elección del alumno. No obstante, puede ser recomendable realizar pruebas básicas iniciales con MPI. Por ejemplo, se puede utilizar el código *hello.c*, de la Sesión 4 de Teoría, para enviar un array de enteros con comunicaciones punto a punto. Después, copiar las funciones *crear_tipo_datos* y *crear_individuo* para enviar individuos en lugar de enteros. También probar con *broadcast*, *scatter*, *gather*…

Posteriormente, comenzar la implementación del algoritmo con un código simple, por ejemplo, con solo dos procesos y una iteración, donde solo se envíe un *Individuo* con comunicaciones punto a punto. Después, aumentar progresivamente la complejidad del código, enviando un array de individuos, aumentar el número de procesos, con varias iteraciones donde solo haya migraciones al final desde cada isla al maestro. Por último, aumentar la frecuencia de las migraciones considerando los parámetros NGM y NEM, hasta implementar el algoritmo completo necesario para resolver las cuestiones de la práctica.

3) Una vez implementado el esquema de comunicaciones de la Cuestión 2, se pueden lanzar ejecuciones variando NGM y NEM para determinar qué valores permiten obtener buenos resultados (de fitness, tiempo o ambos), con el objetivo de fijar estos parámetros adecuadamente. En cualquier caso, la mejor configuración dependerá de la instancia de problema, de la implementación del algoritmo realizada (incluyendo el tipo de comunicaciones) y del sistema computacional.

   Además, se pueden hacer pruebas variando NGM y NEM para conseguir unos tiempos de computación y de comunicaciones similares (del mismo orden de magnitud) en cada iteración, pudiendo variar también, para este fin, el tamaño de la población o del individuo. Por ejemplo, si tenemos que, inicialmente, la computación es mucho mayor que las comunicaciones, podemos disminuir NGM lo que aumentaría la frecuencia de las migraciones (más comunicaciones) y si, además, disminuimos la computación en cada isla (subpoblaciones más reducidas o valor del parámetro *m* menor), haría que los tiempos tiendan a igualarse. Así, se puede ver la influencia de estos parámetros en los tiempos de ejecución del algoritmo. Una vez realizadas estas pruebas, se pueden fijar unos valores de NGM y NEM adecuados para contestar todas las cuestiones de la práctica.

   Por otro lado, también se podría considerar que NEM no es una constante a determinar inicialmente, sino que puede variar en cada ejecución y podría tener valores como: el tamaño de toda la subpoblación (puede ser lo mejor para que haya la máxima contribución de comunicaciones al tiempo total, pero no necesariamente lo mejor para el fitness), o el de un porcentaje de esta. Por supuesto, como se ha indicado previamente, también se le podría dar un valor constante, razonablemente adecuado, promedio, útil para todos los experimentos.

   Como esta parte queda abierta al criterio particular, el alumno deberá elegir la implementación que considere más adecuada, razonando y analizando los resultados obtenidos.

4) Los experimentos no tienen por qué limitarse a usar un número de procesos máximo igual al número de cores físicos. De hecho, si se supera este máximo puede llegar a observarse como se degrada el rendimiento del algoritmo. En caso de que en una máquina concreta no esté habilitado por defecto, se puede usar la opción *-oversubscribe* para forzar a MPI a usar más procesos que cores físicos disponibles. Si el número de procesos es inferior o igual al número cores físicos, esta opción no tiene efecto.

5) En comunicaciones colectivas, cuando el tamaño de la población no es divisible entre el número de procesos, existe la opción de usar *MPI_Gatherv o MPI_Scatterv*, variantes de sus respectivas primitivas habituales, que permiten a los mensajes recibidos o enviados tener diferentes longitudes. También se puede usar *MPI_Gather* o *MPI_Scatter* con la opción *MPI_IN_PLACE* para el proceso 0, y para el resto, una llamada estándar a estas primitivas.

6) Para comparar mejor las prestaciones del algoritmo entre implementaciones, 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. De esta forma, habría que tantear mejor parámetros como *NEM* o *NGM* para mejorar también el fitness. Es evidente que, si solo se desea optimizar el tiempo, lo mejor sería *NEM*=1 y *NGM = n_gen* (la menor cantidad de individuos y la menor frecuencia de migración). Otra opción más simple, si no se quiere usar un indicador, podría ser medir el fitness alcanzado fijando un tiempo de ejecución razonable, igual para todas las implementaciones.