

Práctica Nro. 4

Programación con MPI / Programación híbrida

Información útil para compilar y ejecutar:

- Para compilar con OpenMPI, abra una consola y use *mpicc* empleando la siguiente sintaxis:
mpicc archivofuente.c -o nombreBinario
- Para compilar con OpenMPI+OpenMP, abra una consola y use *mpicc* empleando la siguiente sintaxis:
mpicc archivofuente.c -o nombreBinario -fopenmp
- Para ejecutar un binario en una máquina local, emplee la siguiente sintaxis:
mpirun -np P nombreBinario arg1 arg2 ... argN
donde *P* representa el número de procesos a generar.
- Para ejecutar en el cluster de la cátedra, siga las instrucciones detalladas en el instructivo.

Pautas generales

- Para obtener el tiempo de ejecución de todos los algoritmos se debe utilizar la función provista por la cátedra (*dwalltime*).
- Por convención sólo deberá tomarse el tiempo de ejecución del procesamiento y comunicación de datos (se recomienda medir ambos por separado). Esto significa excluir del tiempo de ejecución:
 - Reserva y liberación de memoria.
 - Inicialización de estructuras de datos.
 - Impresión y verificación de resultados.
 - Impresión en pantalla (*printf*)
- Las pruebas deben realizarse de forma aislada a la ejecución de otras aplicaciones. Se debe ejecutar desde consola, sin otras aplicaciones ejecutándose al mismo tiempo.
- Además del algoritmo paralelo, debe implementar el algoritmo secuencial en el caso que corresponda.
- Los ejercicios 4-6 deben probarse en las siguientes modalidades:
 - a) Usando 1 único nodo con 1 proceso por núcleo.
 - b) Usando 2 nodos con 1 proceso cada 2 núcleos.
 - c) Usando 2 nodos con 1 proceso por núcleo.

Por ejemplo, si el cluster dispone de nodos quad-core, entonces debe generar 4 procesos para el caso a), 4 para el caso b) y 8 para el caso c)

- Para todos los ejercicios 4-7 se debe calcular el speedup y la eficiencia del algoritmo paralelo respecto al secuencial. Además, realice un análisis de escalabilidad y del overhead de las comunicaciones.

Ejercicios

1. Revisar el código *mpi-simple.c*. Compile y ejecute el código. Modifíquelo para que los procesos se comuniquen en forma de anillo: el proceso i debe enviarle un mensaje al proceso $i+1$, a excepción del último que debe comunicarse con el 0.
2. Los códigos *blocking.c* y *non-blocking.c* siguen el patrón *master-worker*, donde los procesos *worker* le envían un mensaje de texto al *master* empleando operaciones de comunicación bloqueantes y no bloqueantes, respectivamente.
 - Compile y ejecute ambos códigos usando $P=\{4,8,16\}$ (no importa que el número de núcleos sea menor que la cantidad de procesos). ¿Cuál de los dos retorna antes el control?
 - En el caso de la versión no bloqueante, ¿qué sucede si se elimina la operación `MPI_Wait()` (línea 52)? ¿Se imprimen correctamente los mensajes enviados? ¿Por qué?
3. Los códigos *blocking-ring.c* y *non-blocking-ring.c* comunican a los procesos en forma de anillo empleando operaciones bloqueantes y no bloqueantes, respectivamente. Compile y ejecute ambos códigos empleando $P=\{4,8,16\}$ (no importa que el número de núcleos sea menor que la cantidad de procesos) y $N=\{10000000, 20000000, 40000000, \dots\}$. ¿Cuál de los dos algoritmos requiere menos tiempo de comunicación? ¿Por qué?
4. El algoritmo *mpi_matmul.c* computa una multiplicación de matrices cuadradas empleando comunicaciones punto a punto:
 - Compile y ejecute el código empleando $N=\{512,1024,2048\}$ usando todos los núcleos de 1 y 2 nodos.
 - Revise las secciones de código donde se realiza la comunicación de las matrices. Analice el patrón de comunicación y piense si es posible emplear comunicaciones colectivas en lugar de a punto a punto. En ese caso, modifique el código original, compile y ejecute la nueva versión. ¿Se mejora la legibilidad? ¿Se logra mejorar el rendimiento? ¿Por qué?
5. Desarrolle un algoritmo paralelo que resuelva la expresión $R = AB + CD + EF$, donde A, B, C, D, E y F son matrices cuadradas de $N \times N$. Ejecute para $N = \{512, 1024, 2048\}$ con $P=\{2,4,8,16\}$.

6. Desarrolle un algoritmo paralelo que dado un vector V de tamaño N obtenga el valor máximo, el valor mínimo y valor promedio de sus elementos. Ejecute para $P=\{2,4,8,16\}$ variando el valor de N .
7. Desarrolle una versión híbrida (MPI+OpenMP) de la multiplicación de matrices. Replique el análisis realizado para el algoritmo puro MPI (ejercicio 5) y compare sus rendimientos. ¿Cuál es mejor? ¿Por qué?