I - S - 2021 - RRF - Programación Paralela y Concurrente - 002

Para esta tarea cree un programa en C que calcule las sumas de Goldbach de números enteros de forma concurrente usando *Pthreads*. Esta tarea es una secuencia de la tarea01.

Correcciones a la tarea01

Si no lo ha hecho, primero corrija su solución a la tarea01 en la carpeta tareas/goldbach_serial/ para aplicar las observaciones que obtuvo durante la revisión de la misma, de manera que no se repliquen las deficiencias identificadas en la tarea02. Recuerde utilizar issues de su sistema de control de versiones:

1. Por cada observación que recibió durante la revisión (sea que las haya anotado o que visualice la grabación de la revisión), cree un issue en el sistema de alojamiento de control de versiones (git.ucr o github). Inicie el título del issue con el número de la tarea, por ejemplo "Tarea01: cambiar arreglo estático por dinámico". Describa en el issue el problema identificado puntualmente. Note que cada issue recibe un número que lo identifica.

2. Corrija un issue a la vez en su repositorio. Modifique los archivos que necesite directamente (NO cree una copia de carpeta). Para cada corrección cree un commit y en el mensaje refiera el número del issue. Si es el último commit que resuelve el issue, ciérrelo desde el mensaje de commit por ejemplo, el mensaje:

git commit -m 'Store sums in dynamic array instead of a static one. Close #13'

cerraría el issue identificado con 13. De esta forma el issue y el commit quedan ligados en el sistema de control de versiones y es fácil la rastreabilidad.

Una vez que haya corregido la tarea 01 pase a la versión concurrente.

Goldbach concurrente

En su repositorio personal copie la carpeta `tareas/goldbach_serial` y su contenido como `tareas/goldbach_pthread`). Haga un commit con este cambio.

El programa concurrente recibirá la lista de números enteros *en la entrada estándar* e imprime la cantidad de sumas de cada uno de ellos, así como las sumas en caso de negativos, de la misma forma en que la versión serial lo hace. Es decir, el programa concurrente debe pasar los mismos casos de prueba que la versión serial (se proveen casos adjuntos a este enunciado). La diferencia con la versión serial, es que la solución a esta tarea debe calcular las sumas de Goldbach de manera **concurrente** utilizando hilos que se reparten el trabajo.

Su programa concurrente debe permitir al usuario invocarlo con un número provisto como argumento de línea de comandos, el cual indica la cantidad de hilos de ejecución que realizarán el cálculo de las sumas de Goldbach. Si este número no se provee, se debe asumir la cantidad de núcleos (CPUs) disponibles en la máquina.

Su solución debe tener un buen diseño concurrente expresado en pseudocódigo o una red de Petri o ambas. El diseño debe centrarse en la lógica concurrente (lo nuevo en esta tarea) y no los detalles algorítmicos utilizados para calcular las sumas de Goldbach (lo ya resuelto en la tarea01). Por ejemplo, las subrutinas de la tarea01 que no cambian, puede invocarlas

sin tener que definirlas en el diseño. Esto ya se ha hecho con algunos ejemplos de diseño en el curso que abstraen tareas como "sentence A1" o "play_chess()".

Importante. Debe repartir el trabajo lo más equitativamente posible entre los hilos de ejecución. Haga que los hilos trabajen de forma lo más paralela posible, evitando sincronización innecesaria. Recuerde que además su solución debe imprimir los resultados en el orden esperado. Por lo tanto, debe impedir la serialización de los hilos usando una estrategia *conditionally safe*.

En esta versión NO haga optimizaciones. La versión concurrente debe implementar el mismo algoritmo que la versión serial. Simplemente debe repartir el trabajo entre los hilos de ejecución. La versión concurrente debe registrar un incremento de velocidad. Es decir, la duración de la versión concurrente con un caso de prueba mediano (ver casos adjuntos) debe ser menor o igual que la duración de la versión serial. IMPORTANTE: NO agregue los casos de prueba medianos a control de versiones (por el contrario, inclúyalos en su archivo .gitignore).

Aspectos a evaluar

Asegúrese de verificar el funcionamiento de su programa con los casos de prueba que se adjuntan. Se recomienda enfáticamente crear más casos de prueba. Recuerde que en toda tarea se evaluará:

1. El diseño de la solución concurrente (archivo .pseudo o red de Petri o ambos).
2. La corrección de la solución (pasar los casos de prueba).
3. Las buenas prácticas de programación.
4. Modularizar (dividir) subrutinas o clases largas, y modularizar los archivos fuente (varios .h y .c).

5. Apego a una convención de estilos (cpplint).
6. No generar accesos inválidos ni fugas de memoria (memcheck, asan, msan, ubsan).
7. No generar condiciones de carrera, espera activa, bloqueos mutuos, o inanición (helgrind, tsan).
8. Documentación de subrutinas y registros (doxygen), además de código no trivial.
9. Proveer mecanismos para construir la solución (Makefile), y usar carpetas.
10. Ignorar archivos generados a partir del código fuente como ejecutables o salida de Doxygen (bin/, build/, doc/).
11. Incremento del desempeño con los casos de prueba medianos.

Se recomienda subir una copia comprimida del código fuente a este enunciado a modo de respaldo.