Memoria Práctica 1

OpenMp

Computación Paralela

Alejandro Romero Pacho Andrés Trigueros Vega

Índice

1. Pasos dados junto con modificaciones realizadas

Para empezar la práctica, como no sabíamos por donde atacar el código, comenzamos por medir los tiempos de los diferentes bucles y ver cuáles eran los de más peso. De un primer vistazo, vimos que el que más tiempo empleaba en ejecución era el 4.8, pero no mucho después, descubrimos que era por los datos de entrada que habíamos seleccionado y vimos que, tanto el 4.1 como el 4.3, también tenían un tiempo grande y de hecho, con algunos datos de entrada, mayor que el del 4.8. Una vez vistos los bucles que más tiempo empleaban, buscamos los que nosotros creíamos que eran más fáciles de paralelizar. Empezamos por el 4.8, ya que aunque suene un poco ridículo, sin tener ni idea parecía el más fácil de paralelizar por su tamaño, y pese a que nuestra manera de decidirlo no fue la correcta, acertamos, ya que logramos obtener en no mucho tiempo, una mejora bastante significativa con la paralelización de ese bucle. Para la parelización de ese bucle solo fue necesario realizar un "pragma parallel for reduction(max: var)". Lo único que había que hacer un poco difícil era el reduction.

Tras esta paralelización estuvimos un día entero en el que solo desechábamos ideas y obteníamos mejoras inapreciables o que, dependiendo de los datos de entrada, no eran ni siquiera mejoras. Durante este día también descubrimos que las mejoras no se reflejaban igual en nuestras máquinas que en la del tablón. Como no sabíamos cual era la razón de esto, preguntamos a Arturo, y nos dijo que eso era posiblemente debido a la versión del compilador, que al ser un procesador amd trabajaba diferente a un intel. Estas "mejorasçonsistieron en la paralelización de los bucles 3.1, 4.2 y 4.5. Para esto, bastaba con añadir un "pragma parallel for"para su paralelización. También, empleamos gran parte del día intentando paralelizar el bucle 4.4, pero cuando al fin lo logramos, habíamos sacado tantas cosas fuera del bucle para evitar condiciones de carrera, que habíamos acabado incrementando el tiempo de ejecución del bucle. Un poco después ese mismo día, logramos paralelizar el bucle 4.3 con ayuda de un "pragma atomic". Puesto que estos cambios ya habían reducido bastante el programa, decidimos enviarlo al tablón, pero aun así, no logramos entrar.
Teníamos pensado dejarlo así hasta el próximo día, pero se nos ocurrió quitar los schedule que habíamos añadido, y al enviamos el código al tablón, logramos entrar por primera vez con un tiempo de 58s.
Ahora que teníamos paralelizados el 4.3 y el 4.8 decidimos ponernos con los bucles 4.4 y 4.6, y decidimos dejar el 4.1 para el final, puesto que después del análasis inicial del código, sabíamos que dicho bucle era el más dificil de paralelizar. En estos dos bucles, tras muchos intentos, acabamos descubriendo que el 4.6 no era posible paralelizarlo y que era mejor juntarlo con el 4.5 ya que así pasabamos de tener que recorrer dos veces el bucle a solamente una, y acabamos logrando paralelizar el bucle 4.4 con ayuda de una sección crítica y un par de variables auxiliares.
Con esto descubrimos también que los bucles 4.2 y 4.8 se podían hacer en un solo bucle, y en vez de en dos, que fuera desde 0 hasta rows * columns, lo que mejoraba algo más el código.
Una vez unidos el bucle 4.5 y 4.6, realizamos los cambios sobre los bucles 4.2 4.4 y 4.8 y volvimos a mandar al tablón el programa, aunque apenas mejoramos el tiempo (aunque en ejecuciones futuras sí que mejoraba el tiempo. Acabamos dando por hecho que, dependiendo de lo saturado que estuviese el tablón y de los datos que utilizara, el mismo código variaban en casi 4 segundos, por lo que decidimos continuar dando por hecho que sí que eran mejoras.

Al día siguiente tratamos de hacer diferentes pruebas para probar schedule, y con ello acabamos

descubriendo que lo mejor era en algunos casos el guided y en otros dejarlo en default. Poniendo dichos schedule logramos bajar hasta la primera referencia.
Puesto que vimos que no podíamos bajar más el tiempo a no ser que nos pusieramos con el bucle 4.1, decidimos ponernos con él. Para intentar paralelizarlo, sacamos las generaciones aleatorias y las intentamos guardar en un vector para así evitar condiciones de carrera. Haciendo esto, descubrimos que no se podía paralelizar de esa manera, pero vimos que al separar la asignación de aleatorios de la asignación del array culture[], reducía el tiempo hasta casi la mitad. Este fue nuestro último cambio importante, y con él llegamos a bajar hasta los 42 segundos.

2. Conceptos utilizados

Inlining sobre el bucle 4.4, sustituyendo las llamadas a la función cellnewdirection por el contenido de la misma.

Pragma parallel for, empleado sobre todos los bucles que se podían o hemos encontrado la manera de paralelizar. La diferencia entre unos y otros la marcaban las diferentes cláusulas, pero la directiva parallel for la hemos utilizado para todos los bucles que hemos paralelizado.

Pragma atomic, directiva utilizada para complementar el parallel for en el bucle 4.3 y así evitar condiciones de carrera sobre la línea de código de asignación del vector culture[]. Esta directiva, hemos procurado usarla solo cuando era completamente necesaria, puesto que solo afecta a asignaciones y aumenta demasiado el tiempo de ejecución del bucle.

Pragma critical, para crear secciones críticas que se ejecutan en secuencial. Al igual que la directiva atomic, esta también incrementa el tiempo de ejecución, por lo que solo la usamos en el bucle 4.4, el cuál tras muchos otros intentos, descubrimos que la única manera de paralelizarlo era con dicha directiva (ya que la variable stepnewcells iba incrementándose y se utilizaba como índice para asignaciones, por lo que no podíamos permitir que dicho incremento se hiciera en paralelo). Con la ayuda de una variable auxiliar, hicimos la asignación de la variable igualándola al incremento de stepnewcells dentro de una sección crítica.

Clausa schedule, en el bucle 4.8 la utilizamos porque acabamos viendo que fijarlo como guided en vez de dejarlo en default funcionaba mejor. El cambio era mínimo, puesto que al dejarlo en default, funcionaba en static y la diferencia entre static y guided es mínima.

Desdoblar bucles, en casos como el bucle 4.2 y 4.8, en los que el bucle se podía hacer recorriéndolo solo una vez, cambiando la i y la j (las columnnas y las filas), por tan solo la i y sustituyendo el valor de accesmat por su vector, culture[].

Juntar dos bucles que están iterando sobre lo mismo sin que haya condiciones de carrera,como hicimos con el 4.5 y 4.6.

Empleo de la cláusula reduction para variables que se iban modificando en cada iteración, como por ejemplo en el bucle 4.8, en el que utilizamos reduction(max:var) currentmaxfood para evitar condiciones de carrera.

Empleo de variables auxiliares como en el 4.4 en el que necesitábamos guardar el valor de stepnewcells, ya que este se usaba como índice en asignaciones, y para poder añadir variables que se obtienen por llamada simstat.var, puesto que esas variables no se pueden añadir a la cláusula reduction, por lo que tuvimos que igualar su valor a una variable auxiliar antes del bucle, trabajar con dicha variable y cambiar el valor nuevamente al salir del bucle.

En términos generales, pese a que en algunos bucles no lo hayamos indicado, las cláusulas usadas que "menos marcaron la diferencia"fueron: private, que se utiliza para las variables que iteran

en los bucles, firstprivate para las variables que queríamos que mantuvieran su valor global, como por ejemplo rows columns, shared para los vectores y reduction(+:var) para las variables que incrementaban su valor por cada iteración...

3. Intentos de modificación desechados

Para explicar los diferentes intentos desechados, vamos a ir hablando uno por uno sobre cada uno de los bucles de menos a más en función de las cosas que probamos sobre ellos. En esta sección solo se hablará de cambios relevantes, no mencionaremos pruebas intentando cambiar la política de reparto de los hilos. El caso del bucle 4.3, no desechamos ninguna opción, ya que nuestro mayor problema fue solamente donde poner el pragma atomic (tuvimos "suerte"de que la opción que barajamos al principio era la correcta para enfrentarnos a este bucle).

Bucle 4.2 y 4.8: En el caso de estos bucles, las dos opciones que teníamos era hacerlo en dos for o en uno, y probando con diferentes valores de entrada, descubrimos que la forma más eficiente era con un solo bucle.

Bucle 4.5 y 4.6: En el caso de estos dos (seguimos sin saber decir qué manera sería la mejor para optimizar el tiempo de ejecución), las dos opciones eran dividirlos y paralelizar el 4.5 o juntarlos y no paralelizar ninguno de los dos. Nuestra mejor marca de tiempo la obtuvimos con la segunda opción, pero en nuestras máquinas, la primera era algo mejor que la segunda, por lo que seguimos sin poder decir a ciencia cierta cuál de las dos es la más eficiente.

Bucle 4.1: Intentamos paralelizarlo, al igual que con la gran mayoría de bucles, con un parallel for, pero tras unas cuantas pruebas acabamos descubriendo que la generación de números aleatorios hacía que si varios hilos ejecutanban una instrucción en paralelo, los valores darán resultados completamente diferentes a lo que deberían dar en secuencial, por lo que siguiendo el consejo que nos dieron en clase, empezamos a probar a sacar la generación aleatoria del bucle y almacenarla en un vector de tamaño igual al número de iteraciones del bucle. Con ello, haríamos en secuencial la generación de aleatorios y luego podríamos paralelizar la asignación del bucle en el que realizaríamos la asignacion al vector culture[], pero al intentar paralelizar este bucle también obteníamos valores incorrectos.
Hasta ahora habíamos cambiado de 2 a 4 bucles. En los primeros, se guardaban en vectores los resultados de la generación aleatoria, y en los segundos se asignaban los valores generados en el bucle anterior al vector culture[]. Para nuestra sorpresa, acabamos descubriendo que había una gran mejora, incluso sin paralelizar ninguno de los bucles, desdoblándolos y haciendo el trabajo por separado, es decir, en cuatro bucles.
Con estos cambios. ya habíamos entrado en la referencia, y este bucle seguía siendo el que más tardaba, por lo que decidimos paralelizarlo utilizando secciones críticas. Esta idea fue desechada, puesto que el tiempo de ejecución casi se doblaba (ya que paralelizar un bucle y añadir todo su contenido a una sección crítica, hace que se ejecute en secuencial), puesto que esto no funcionó y el bucle con la generación aleatoria no era paralelizable, decidimos dejar de intentar mejorar más este bucle y pasamos a los siguientes.

Bucle 4.4: Este bucle es al que más tiempo le hemos dedicamos. Desde el principio vimos que el problema iba a estar en la variable que se utilizaba como índice de vectores, que a la vez estaba incrementándose en cada iteración. Nuestra primera idea fue, puesto que no vimos que se pudiera solucionar con una variable auxiliar, utilizar un vector en el que se guardaran todos los valores de stepnewcells para cada iteración de manera secuencial y luego emplear el vector de manera paralela.

Tras muchos errores, acabamos sacando casi todo el bucle fuera, y al lograr paralelizar el segundo bucle, el tiempo había incrementado a causa del primero (en el que guardábamos los valores de stepnewcells en el array), y con esto, volvimos al código secuencial.
Después de desechar tanto la opción de la variable auxiliar como la del vector, decidimos volver a probar la de la variable, la cual pensamos que quizás si lográbamos implementar, mejoraría el tiempo de ejecución. Puesto que la última vez que lo intentamos, los valores nos daban incorrectos, decidimos probar a utilizar secciones críticas (en este caso, estaba bastante claro cual debía ser la sección crítica, puesto que sabiamos cuál era la operación que no estaba dando problemas), así que decidimos meter la asignación de la variable auxiliar y el incremento en una sección crítica, logrando así mejorar el tiempo de ejecución (aunque no demasiado), y a su vez obtener los valores correctos.

4. Bibliografía

- https://pages.tacc.utexas.edu/ eijkhout/pcse/html/omp-reduction.html.
- http://www.ditec.um.es/ javiercm/curso_psba/sesion_03_openmp/PSBA_OpenMP.pdf.
- https://www.openmp.org/spec-html/5.0/openmpsu107.html.
- https://docs.microsoft.com/es-es/cpp/parallel/concrt/convert-an-openmp-loop-that-uses-a-reduction-variable?view=vs-2019.
- http://ocw.uc3m.es/ingenieria-informatica/arquitectura-de-computadores-ii/otros-recursos-1/or-f-008.-curso-de-openmp.