Informe: Examen Paralela 2022-2

Integrantes: Nahuel Espinoza Fuentes

Profesor: Sebastián Salazar Molina

Asignatura: Computación Paralela y Distribuida

Fecha de entrega: Martes 20 de Diciembre, 18:00 horas

Índice

Contenido

Introducción

La computación paralela es una técnica de programación que permite dividir un problema grande en varias partes más pequeñas, que se pueden resolver de manera simultánea en diferentes procesadores o máquinas. Esto puede ser muy útil cuando se trata de tareas que requieren muchos cálculos o que son muy tiempo-consumidoras, ya que la paralelización permite acelerar el proceso y obtener resultados más rápidamente.

La predicción matemática, por otra parte, es el uso de modelos matemáticos y estadísticos para hacer predicciones sobre el comportamiento de ciertos fenómenos o procesos. Por ejemplo, la predicción matemática se puede utilizar para predecir el comportamiento del mercado financiero, la demanda de un producto en una empresa o el rendimiento académico de un estudiante.

La computación paralela y la predicción matemática se pueden utilizar juntas para resolver problemas de predicción de manera más eficiente y rápida. Por ejemplo, si tenemos que hacer predicciones sobre el comportamiento de un gran conjunto de datos, podemos dividir los datos en varias partes y utilizar diferentes procesadores para entrenar modelos matemáticos en paralelo, lo que nos permitiría obtener resultados más rápidamente que si utilizáramos un solo procesador.

Problema Planteado

Se solicita desarrollar una aplicación que, dada una fecha futura, pueda predecir la cantidad de accidentes esperados para el día de entrada. Esta aplicación debe implementar un modelo matemático, que use los datos históricos, esté codificada en C o C++ y use OpenMP o MPI para maximizar el rendimiento. Para el análisis, se adjunta un archvo CSV (archivo con valores separados por comas) con los datos históricos, desde que se iniciaron las operaciones (06/10/2017) hasta la fecha de corte (10/12/2022):

El código debe considerar los siguientes alcances:

Entrada.

La aplicación debe leer la fecha como el primer argumento de la línea de comandos, el formato de fecha corresponde al ISO 8601 (año/mes/día).

Resultado.

La salida del programa debe mostrar en pantalla:

- La fecha (en formato ISO 8601).
- La cantidad de accidentes esperados.
- Un salto de línea.

Una vez entregue el resultado, el programa debe terminar.

Desarrollo del problema

Para desarrollar el código solicitado, se escogió el lenguaje de programación C++, la API OpenMP para la programación paralela, las diversas librerías de C++ para trabajar con archivos y los cálculos matemáticos y de tiempo, el compilador Code Blocks, y el sistema operativo Windows 10. Para trabajar con los datos, se utilizó la técnica de análisis regresión lineal

Regresión Lineal

La regresión lineal es un método utilizado en estadística y análisis de datos para entender la relación entre dos variables. Se llama "lineal" porque se asume que hay una relación lineal entre la variable independiente y la variable dependiente.

Por ejemplo, supongamos que queremos entender cómo varía el precio de una casa en función de su tamaño. El tamaño de la casa sería la variable independiente y el precio sería la variable dependiente. Podríamos recopilar datos sobre el tamaño y el precio de un conjunto de casas y luego ajustar un modelo de regresión lineal para predecir el precio de una casa dado su tamaño.

Para hacer esto, se ajusta una línea recta a los datos, utilizando un algoritmo que minimiza la suma de los cuadrados de los errores (también conocida como el método de mínimos cuadrados). La línea recta se puede utilizar para hacer predicciones sobre el precio de una casa dado un tamaño específico.

En resumen, la regresión lineal se utiliza para predecir el valor de una variable dependiente a partir del valor de una o más variables independientes, utilizando una línea recta ajustada a los datos.

OpenMp

OpenMP es una interfaz de programación para programación paralela en C, C++ y Fortran. Proporciona un conjunto de directivas de compilador, rutinas de biblioteca y variables de entorno que permiten a los desarrolladores especificar cómo se debe paralelizar un programa y controlar el número de hilos utilizados para la ejecución paralela. OpenMP está diseñado para ser utilizado con sistemas de memoria compartida, donde múltiples hilos pueden acceder a las mismas ubicaciones de memoria. Para utilizar OpenMP en un programa, es necesario incluir los encabezados adecuados y utilizar directivas de OpenMP para especificar las regiones de código que deben ser paralelizadas. Las directivas suelen escribirse como comentarios especiales en el código fuente, por lo que son ignoradas por los compiladores que no admiten OpenMP.

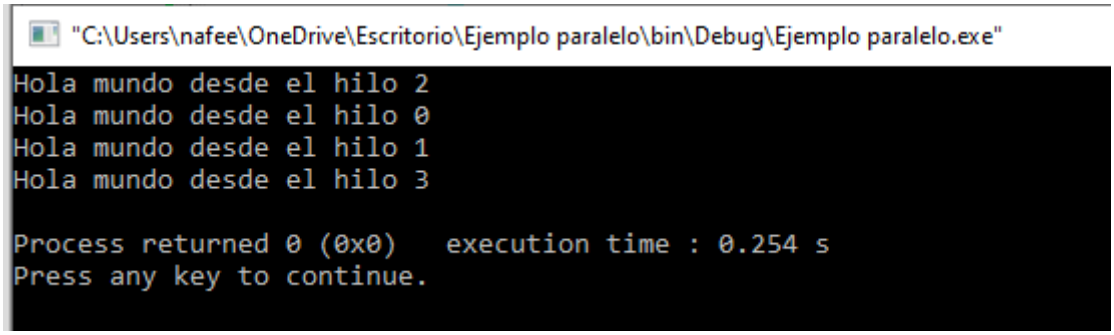
Paralelizar un Codigo

Paralelizar un código significa dividirlo en varias partes que pueden ser ejecutadas de manera simultánea en diferentes procesadores o núcleos de procesamiento. Esto se hace con el fin de acelerar la ejecución del código, ya que permite utilizar varios procesadores o núcleos a la vez en lugar de tener que ejecutar el código secuencialmente en un solo procesador o núcleo. La paralelización del código puede ser útil en situaciones en las que se necesita un gran poder de procesamiento, como en el análisis de grandes conjuntos de datos o en la simulación de procesos complejos. Sin embargo, paralelizar un código puede ser un desafío, ya que requiere dividir el código de manera adecuada y asegurar que las partes paralelizadas puedan trabajar de manera independiente y sin interferir entre sí. También puede ser necesario realizar cambios en el código para asegurar que los resultados finales sean correctos y consistentes. Aquí hay un ejemplo de codigo paralelo en C++ usando Code Blocks y la librería omp.h de OpenMp

```
#include <iostream>

#include <omp.h>

int main()

{

   #pragma omp parallel

   {

      std::cout << "Hola mundo desde el hilo " << omp_get_thread_num() << std::endl;
```

```
   }
 return 0;
}
```

Este codigo imprime:

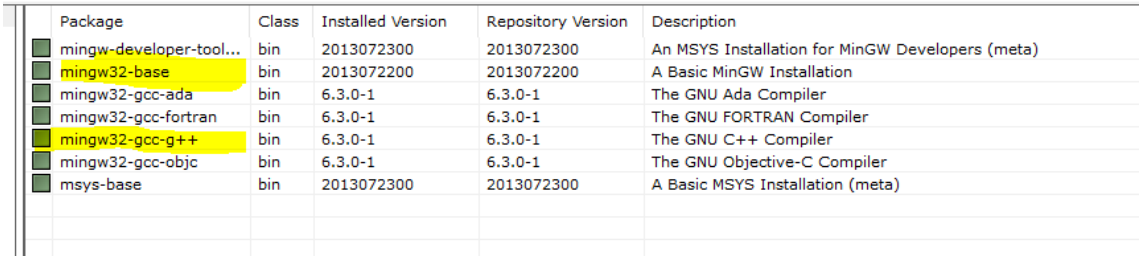


La directiva #pragma omp parallel indica a OpenMP que debe crear un grupo de hilos y ejecutar el bloque de código siguiente en paralelo. La función omp_get_thread_num() se utiliza para obtener el número de hilo en el que se está ejecutando actualmente.

Para compilar este programa, debe utilizar un compilador que soporte OpenMP y habilitar la opción de compilación correspondiente. Por ejemplo, en GCC, puede usar la opción -fopenmp:

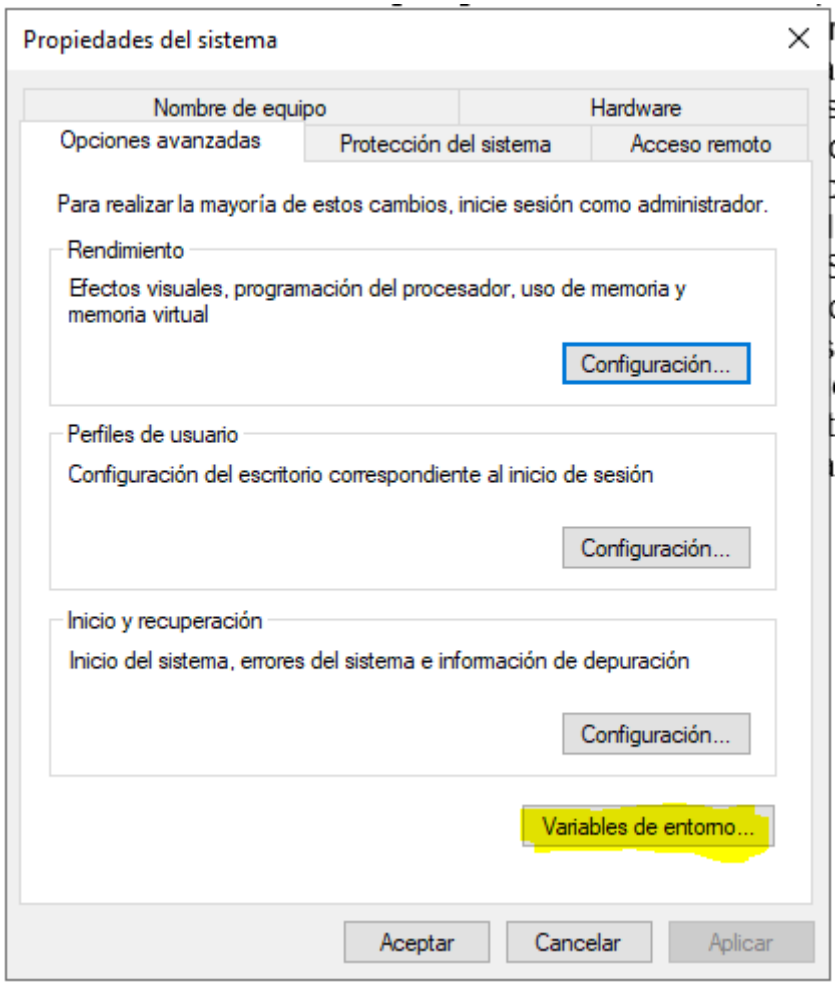
Funcionamiento del codigo

Para ejecutar el codigo, es necesario tener instalado Code blocks o similar (20.03), se puede descargar del siguiente link (https://sourceforge.net/projects/codeblocks/files/Binaries/20.03/Windows/codeblocks-20.03mingw-setup.exe/download), luego de esto, es necesario instalar MinGW desde el siguiente link (https://sourceforge.net/projects/mingw/), se deben instalar las siguiente librerías (las que estan marcadas con amarillo)

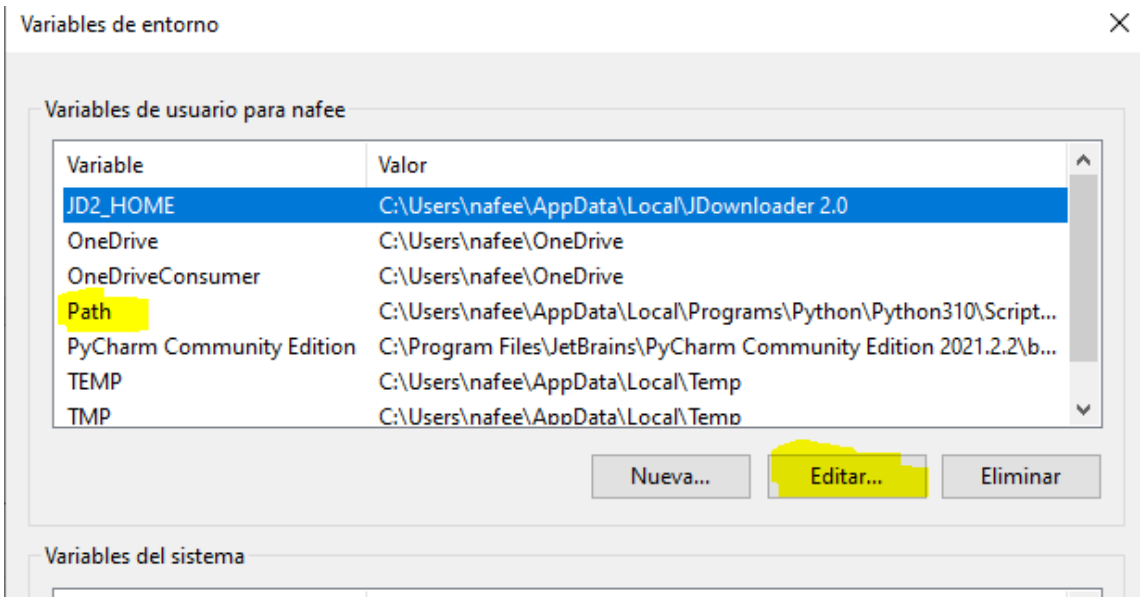


Package	Class	Installed Version	Repository Version	Description
mingw-developer-tool...	bin	2013072300	2013072300	An MSYS Installation for MinGW Developers (meta)
mingw32-base	bin	2013072200	2013072200	A Basic MinGW Installation
mingw32-gcc-ada	bin	6.3.0-1	6.3.0-1	The GNU Ada Compiler
mingw32-gcc-fortran	bin	6.3.0-1	6.3.0-1	The GNU FORTRAN Compiler
mingw32-gcc-g++	bin	6.3.0-1	6.3.0-1	The GNU C++ Compiler
mingw32-gcc-objc	bin	6.3.0-1	6.3.0-1	The GNU Objective-C Compiler
msys-base	bin	2013072300	2013072300	A Basic MSYS Installation (meta)

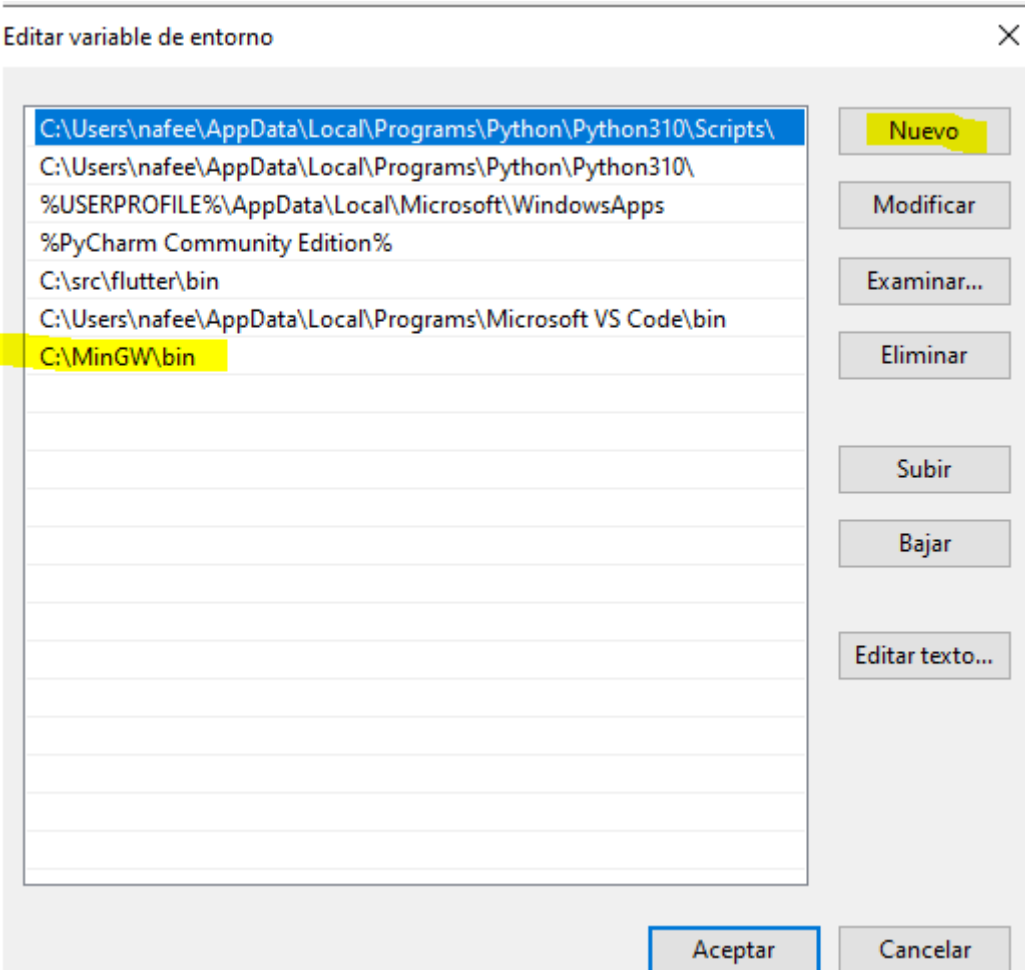
Una vez descargadas e instaladas las librerías debemos añadir al Path en las variables de entorno de Windows, entrando en la opción marcada con amarillo



Luego de ingresar seleccionamos Path y seleccionamos Editar



Luego de esto seleccionamos en nuevo y escribimos la dirección de la carpeta bin del MinGW, por defecto esta en la carpeta raíz del disco principal (C:)

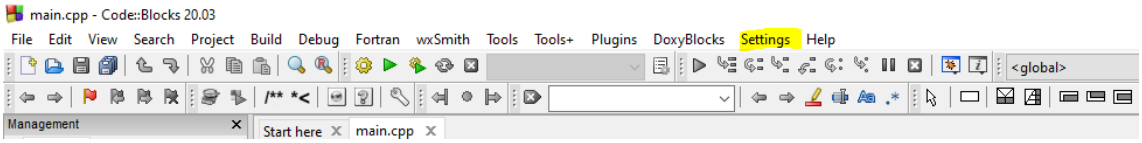


Podemos comprobar que esta operación de hizo correctamente, ingresando al cmd de Windows 10 y escribiendo g++ -v, como se ve en la imagen, vemos en amarillo la versión del gcc.

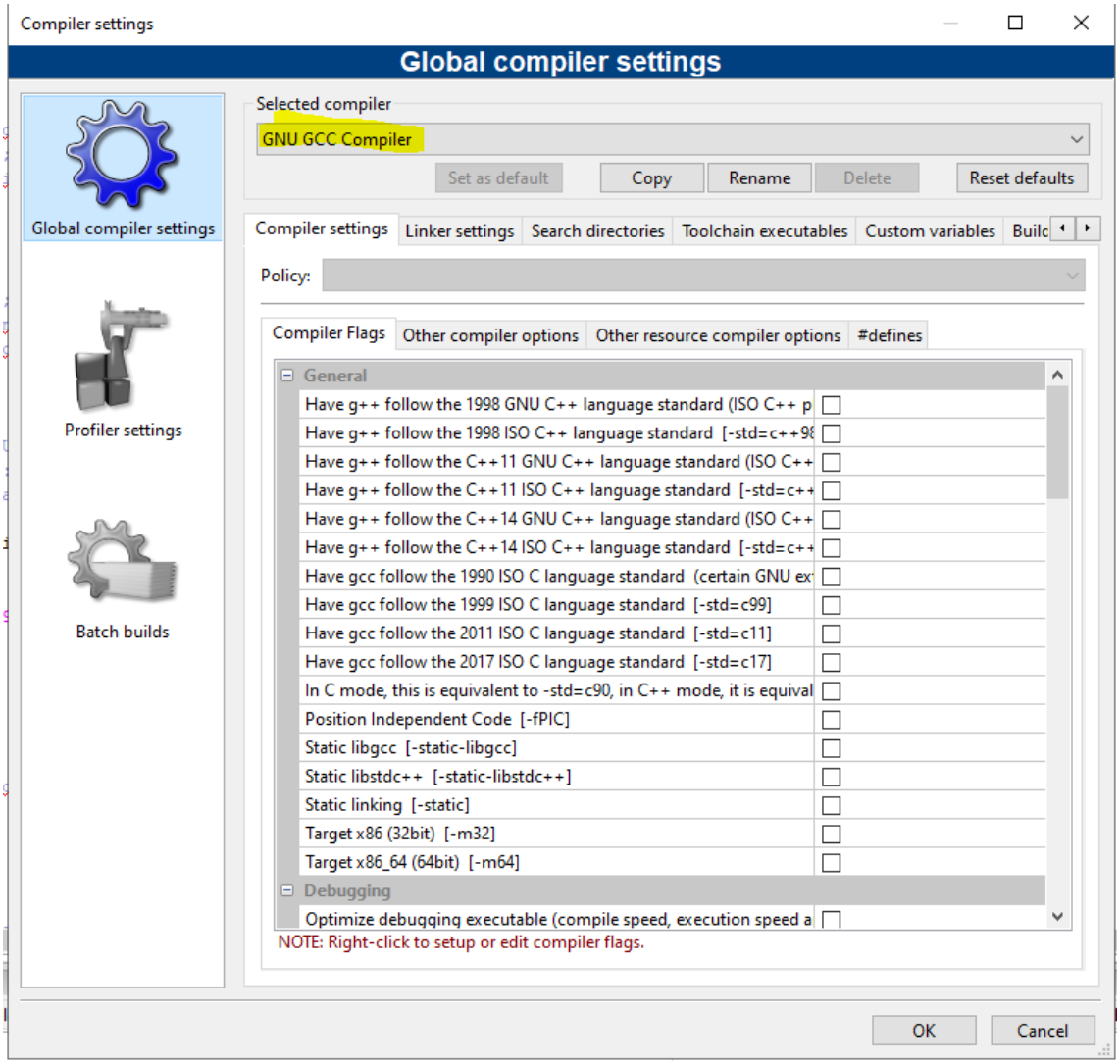
Para instalar en Ubuntu, debemos abrir una terminal y escribir <sudo apt update>, es posible que se nos solicite la contraseña de inicio, luego escribimos <sudo apt install build-essential> y con el comando gcc –version comprobamos la instalación

```
Microsoft Windows [Versión 10.0.19044.2364]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\nafee>g++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=c:/mingw/bin/../libexec/gcc/mingw32/6.3.0/lto-wrapper.exe
Target: mingw32
Configured with: ../src/gcc-6.3.0/configure --build=x86_64-pc-linux-gnu --host=mingw32 --with-gmp=/mingw --with-mpfr=/mi
ngw --with-mpc=/mingw --with-isl=/mingw --prefix=/mingw --disable-win32-registry --target=mingw32 --with-arch=i586 --ena
ble-languages=c,c++,objc,obj-c++,fortran,ada --with-pkgversion='MinGW.org GCC-6.3.0-1' --enable-static --enable-shared -
-enable-threads --with-dwarf2 --disable-sjlj-exceptions --enable-version-specific-runtime-libs --with-libiconv-prefix=/m
ingw --with-libintl-prefix=/mingw --enable-libstdcxx-debug --with-tune=generic --enable-libgomp --disable-libvtv --enabl
e-nls
Thread model: win32
gcc version 6.3.0 (MinGW.org GCC-6.3.0-1)

C:\Users\nafee>
```

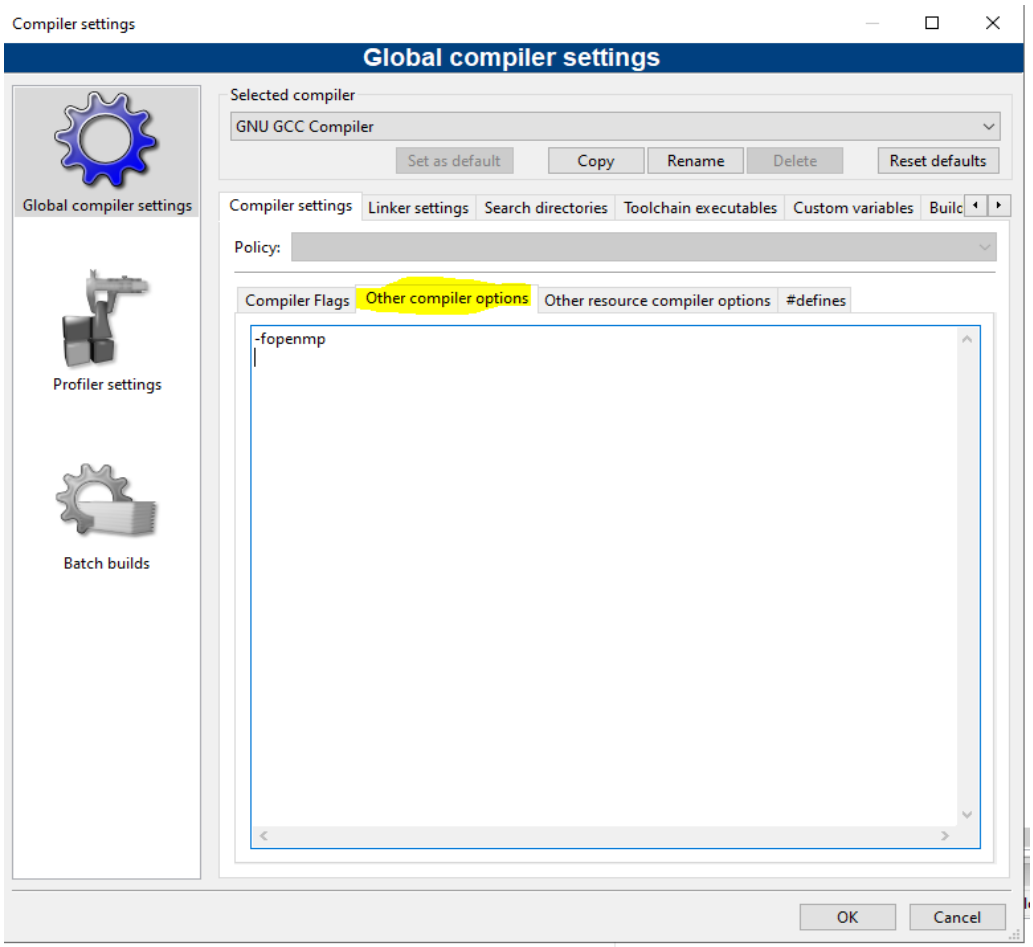
Ahora debemos el archivo main.cpp e ir al apartado de settings



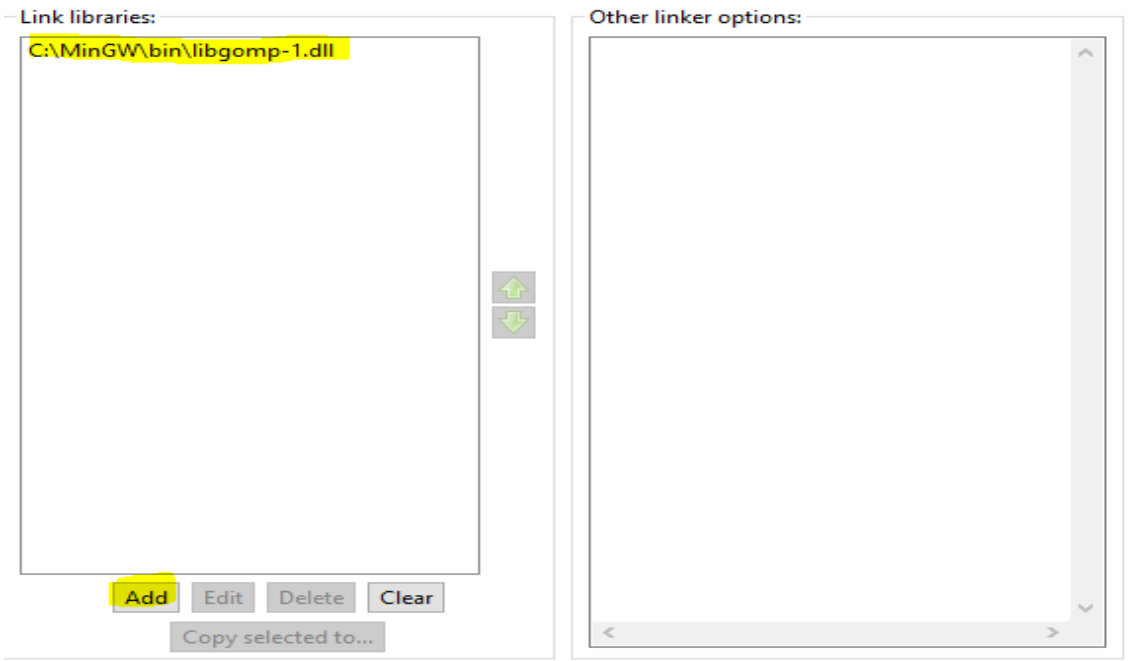
Ahí buscamos la opción Compiler, se nos abrirá la ventana que se ve en la imagen de mas abajo y debemos corroborar que el compilador sea GNU GCC Compiler



Luego vamos a la ventana Other compiler options y escribimos -fopenmp



Luego nos vamos al apartado Linker settings y debemos añadir el archivo libgomp-1.dll que esta en la carpeta bin de MinGW.



Habiendo eso esto, el compilador ya reconoce la instrucción #pragma omp parallel, ya que sin esto el programa compilaría, pero se saltaría esta instrucción.

Funciones del codigo

Esta estructura corresponde a la biblioteca ctime, y se utiliza para convertir la fecha en formato DD-MM-YYYY en un valor que pueda ser utilizado para la regresión lineal

```
double fecha_a_segundos(char* anio, char* mes, char* dia) {
   // Crear una estrcutura tm con la fecha
   tm fecha;
   fecha.tm_year = atoi(anio) - 1900;;
   fecha.tm_mon = atoi(mes) - 1;
   fecha.tm_mday = atoi(dia);

   // Convertir el objeto tm a un objeto time_t
   time_t t = mktime(&fecha);

   // Obtener el número de segundos transcurridos desde la madrugada del 1 de enero de 1970
   return t;
}
```

En esta parte se declara el Archivo, se abre y se definen las variables a utilizar inicializadas en 0, además se le indica a la función que salte la primera línea del codigo que corresponde al nombre de las variables (fecha/accidente)

```
    // Establece el número de hilos a usar
    ifstream Archivo;

    Archivo.open("datos_examen.csv");

    string linea = "";

    float aux = 0;
    float sumA = 0;
    float sumY = 0;
    float sumF = 0;
    float sumxY = 0;
    float sumXX = 0;
    float sec;

    getline(Archivo,linea);// salta la primera linea
```

En esta parte del codigo, se usa un for para trabajar las fechas y convertirlas en un dato que podamos usar, adicionalmente se operan las variables necesarias para la regresión lineal.

```
    omp_set_num_threads(4); //establece el numero de hilos en 4

    #pragma omp parallel //for reduction (+:sumA,sumxY,sumXX,sumY)
//Lo ideal hubiese sido utilizar esta funcion para paralelizar pero hay que cambiar el for
    for(string linea; getline(Archivo,linea);){
        string fecha;
        int vdependiente; //variable dependiente corresponde a los accidentes

        string tempString = "";

        stringstream inputString(linea);

        getline(inputString, fecha, ';');
        //Dividimos la fecha en componentes utilizando strtok()
        char* date_str = strdup(fecha.c_str());
        char* day_str = strtok(date_str, "-");
        char* month_str = strtok(NULL, "-");
        char* year_str = strtok(NULL, "-");


        sec =  sec = fecha_a_segundos(year_str , month_str, day_str);

        sumF = sumF + sec;


        getline(inputString,tempString,';');
        vdependiente = atoi(tempString.c_str());


        aux = aux + 1;//contador de datos debe dar 1892
        //cout << segundos <<endl;Mensaje de prueba
        sumA = sumA + vdependiente;
        sumxY = sumxY +(sec * vdependiente);
        sumXX = sumXX + (sec*sec);
        sumY = sumY + (vdependiente*vdependiente);
        linea = "";
}
```

En esta parte se calculan los valores necesarios de la regresión línea, siendo la sumatoria de las variables dependientes e independientes, sus productos, cuadrados y cocientes.

```
}
    float promx = sumA/aux;
    float promeFecha = sumF/aux;
    float nxy = aux * promeFecha * promx;
    float nx2 = aux* promeFecha * promeFecha;
    float pendiente = (sumxY - nxy)/(sumXX - nx2);
    float y = promx - pendiente * promeFecha;
```

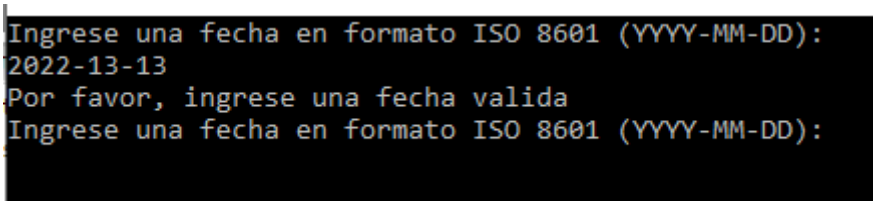
En esta función es donde se pide el dato al usuario, y además, se valida la fecha ingresada, por ejemplo, si se ingresa una fecha con mes 13 el programa dirá que es incorrecta y pedirá una nueva, el formato debe ser Año-Mes-Dia (con guiones según el formato ISO 8601), un ejemplo de fecha para meses como Enero y Febrero, es 2024-02-02, si se ingresa 2024-2-2, el programa pedirá otra fecha. Una vez ingresada la fecha, imprimirá la predicción para ese día y terminara.

```
string input; //input corresponde a los datos ingresados por el usuario
while (true){ //Este ciclo es para asegurar la fecha y validarla, si se ingresa una fecha con el mes 13 por ejemplo, pedira una nueva fecha
cout << "Ingrese una fecha en formato ISO 8601 (YYYY-MM-DD): " << endl;
cin >> input;

// Verifica si la cadena de caracteres tiene el formato correcto
if (input.size() != 10 || input[4] != '-' || input[7] != '-') {
  cout << "Por favor, ingrese una fecha valida" << endl;
  continue;
}

// Verifica si los componentes de la fecha son números válidos
int year, month, day;
if (!(istringstream(input.substr(0, 4)) >> year) ||
    !(istringstream(input.substr(5, 2)) >> month) ||
    !(istringstream(input.substr(8, 2)) >> day)) {
  cout << "Por favor, ingrese una fecha valida" << endl;
  continue;
}
    if (month < 1 || month > 12) {
  cout << "Por favor, ingrese una fecha valida" << endl;
  continue;
    }
// Crea una estructura tm con los valores de la fecha
struct tm date = { 0 };
date.tm_year = year - 1900;
date.tm_mon = month - 1;
date.tm_mday = day;

// Convierte la fecha a un valor de tiempo
time_t t = mktime(&date);

if (t != -1) {
  // La fecha es válida
  break;
}else {
  // La fecha es inválida
  cout << "Por favor, ingrese una fecha valida" << endl;
    //Aqui termina la validacion
}
}
```

Ejemplo de funcionamiento

```
Ingrese una fecha en formato ISO 8601 (YYYY-MM-DD):
2022-12-12
La prediccion de accidentes para la fecha 2022-12-12 es: 2.

Process returned 0 (0x0)   execution time : 5.801 s
Press any key to continue.
```

Ejemplo de fecha mal ingresada



Uso de la paralelización

Para aumentar el rendimiento del codigo, se utilizó la función #pragma omp parallel en el for y se asigno con el comando omp_set_num_threads(4); el número de hilos, escogí esta función, porque, a mi criterio es la parte mas demandante del codigo, aunque originalmente buscaba utilizar una forma más especifica que era (for reduction (+:sumA,sumxY,sumXX,sumY)), esto no fue posible debido a que estas tareas debían ser sincronizadas, algo que no puede realizar y no fui capaz de resolver el error que me presentaba el codigo.

Conclusión

Crear un código paralelo en C++ puede ser un desafío debido a varios factores. Algunas de las dificultades más comunes son las siguientes:

Diseño del código: para que un código pueda ser ejecutado de manera paralela, es necesario dividir el trabajo en un conjunto de tareas que puedan ser realizadas de manera independiente. Esto puede ser difícil de lograr en ciertos casos, ya que algunos problemas tienen dependencias entre sus tareas y no pueden ser divididos de manera efectiva.

Sincronización de tareas: cuando varias tareas están siendo ejecutadas de manera concurrente, es importante asegurar que no haya conflictos de acceso a recursos compartidos. Esto puede requerir la implementación de medidas de sincronización como semáforos o bloqueos de mutex, lo que puede ser complejo de implementar y puede afectar el rendimiento del código.

Debugging: depurar código paralelo puede ser más difícil que depurar código secuencial debido a la complejidad adicional introducida por la ejecución concurrente de varias tareas. Esto puede hacer que sea más difícil determinar la causa de errores o problemas de rendimiento.

Herramientas de desarrollo: aunque existen herramientas de depuración y perfilado específicas para código paralelo, estas pueden ser más difíciles de usar y menos estables que sus contrapartes secuenciales.

En resumen, crear un código paralelo en C++ puede ser un desafío debido a la necesidad de diseñar el código de manera adecuada para su ejecución paralela, la implementación de medidas de sincronización y la dificultad añadida en la depuración y perfilado del código.

Para este codigo en particular, uno de los grandes desafíos y problemas a su vez, fue hacer énfasis en las funciones del codigo en sí, pero dejar para el final las acciones que debía tomar para poder hacer el codigo paralelo, algo de lo que me di cuenta tarde y donde concluí, con mis conocimientos, el que debí haber priorizado el cómo resolvería este problema haciendo énfasis en su ejecución paralela y no en el resultado de las operaciones matemáticas que debía realizar.

Anexos

Problemas presentados en la ejecución:

En algunas ejecuciones, una misma fecha puede dar dos predicciones distintas

Al ejecutar el codigo de manera paralela, en algunas ocasiones el programa terminaba de manera repentina, e imprimía un codigo de error

```
terminate called recursively
terminate called after throwing an instance of 'std::length_error'
  what():  basic_string::_M_create

Process returned 3 (0x3)   execution time : 2.313 s
Press any key to continue.
```

Al ejecutar el codigo en Ubuntu, podía dar como resultado violación de segmento <core> generado, debido a que no podía acceder al archivo csv, este error también se daba en Windows, usando Code Blocks, pero al ejecutar el codigo este daba como resultado -1 en las fechas en vez de transformarlas en otros valores

Bibliografía

-Validar la entrada del usuario en C++

https://www.delftstack.com/es/howto/cpp/cpp-input-validation/

-Regression Analysis and the Best Fitting Line using C++

https://www.geeksforgeeks.org/regression-analysis-and-the-best-fitting-line-using-c/

-Ejercicios de programación paralela con OpenMP y MPI

https://drive.google.com/file/d/1iHxyr_T8g4c3mA0Zf9fmEiP66OC2gMeJ/view