Informe: Examen Paralela 2023

Integrantes: Jaime Ugaz Riveros

Profesor: Sebastián Salazar Molina

Asignatura: Computación Paralela y Distribuida

Fecha de entrega: 17-07-2023

Tabla de contenido

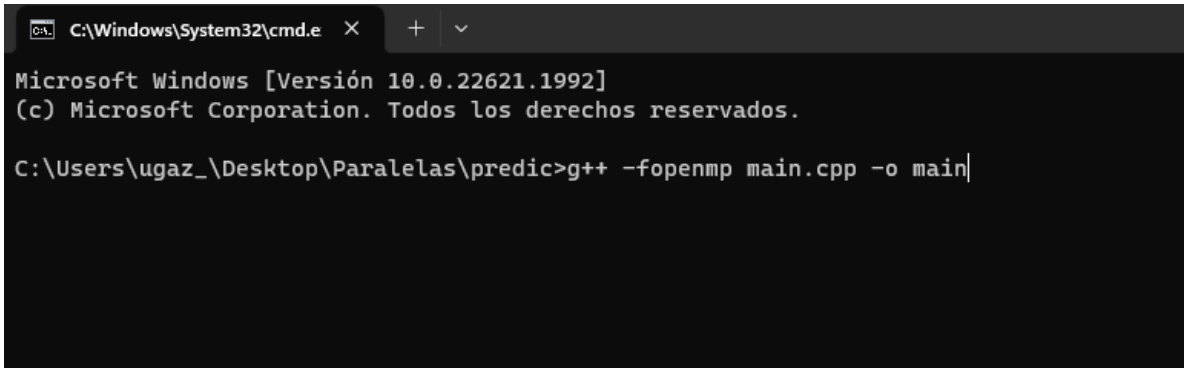
Introducción

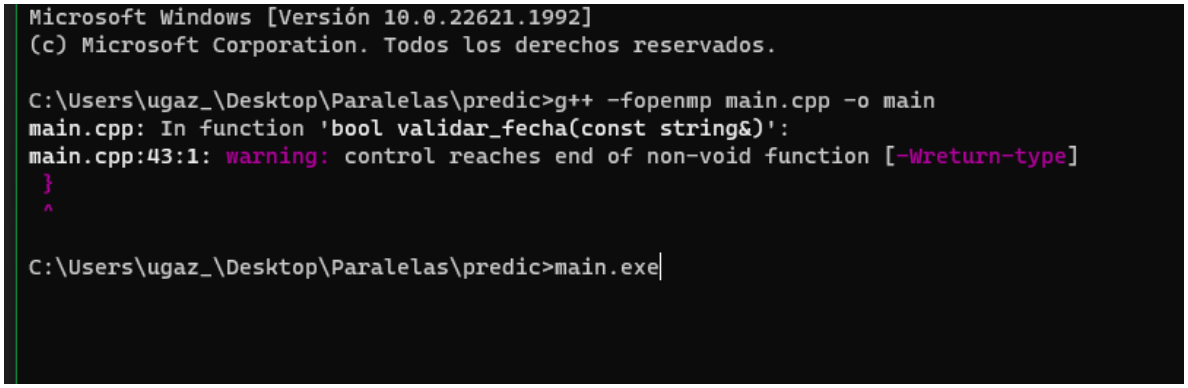
La computación paralela es una técnica de programación que permite dividir un problema grande en varias partes más pequeñas, que se pueden resolver de manera simultánea en diferentes procesadores o máquinas. Esto puede ser muy útil cuando se trata de tareas que requieren muchos cálculos o que son muy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

Ejecución del Código

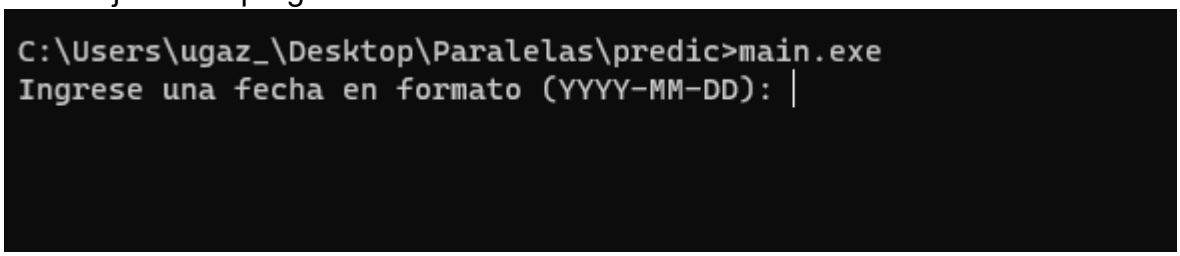
Para ejecutar el código, se puede realizar de dos formas

La primera es a través de comando, Primero ingresamos a cmd o powershell y entramos a la ruta donde se ubica el proyecto para posteriormente usar los comandos “g++ -fopenmp main.cpp -o main” y luego “main.exe”.

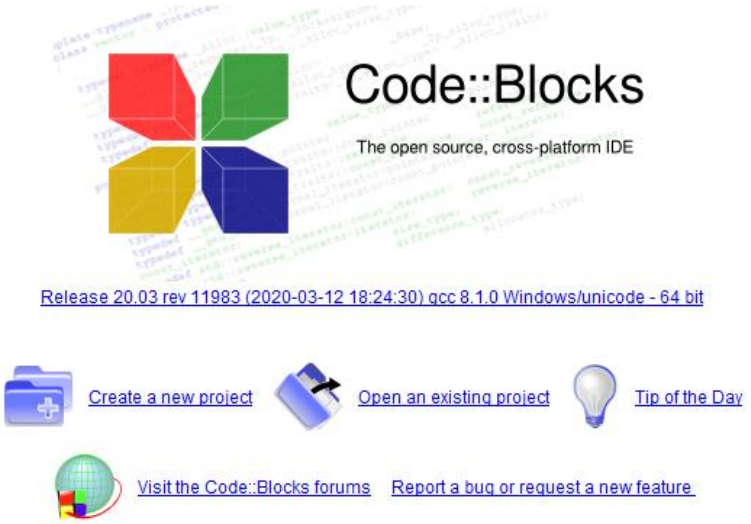




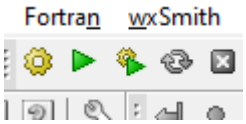
Y nos ejecuta el programa



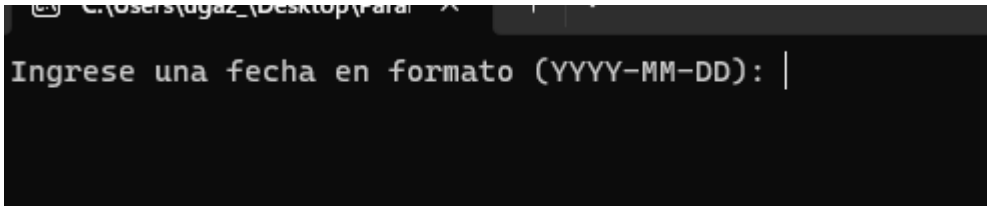
La segunda forma de ejecutar el código es a través de CodeBlocks, para comenzar abrimos el proyecto en CodeBlocks



Luego presionamos en Build and run



Para que nos ejecute el programa



Funciones del Código

```
// Función que convierte una fecha en segundos desde la época
double fecha_a_segundos(const char* anio, const char* mes, const char* dia) {
    tm fecha;
    fecha.tm_year = atoi(anio) - 1900;;
    fecha.tm_mon = atoi(mes) - 1;
    fecha.tm_mday = atoi(dia);
    time_t t = mktime(&fecha);
    return t;
}
```

La función “fecha_a_segundos” convierte una fecha representada por los componentes de año, mes y día en segundos transcurridos desde la época (1 de enero de 1970 00:00:00 UTC) hasta la fecha especificada.

La función toma como argumentos tres punteros a cadenas de caracteres que representan el año, mes y día respectivamente. Luego, crea una estructura tm que almacena los componentes de la fecha y establece los valores del año, mes y día utilizando la función atoi para convertir las cadenas de caracteres a enteros.

```
bool validar_fecha(const string& fecha){
    regex formato("\\d{4}-\\d{2}-\\d{2}$");
    if (!regex_match(fecha, formato)){
        return false;
    }

    int year, month, day;
    sscanf(fecha.c_str(), "%d-%d-%d", &year, &month, &day);

    if (year < 1900){
        return false;
    }
    if (month < 1 || month > 12){
        return false;
    }
    if (day < 1 || day > 31){
        return false;
    }
}
```

La función “validar_fecha” verifica si una cadena de caracteres representa una fecha válida en el formato ISO 8601 (YYYY-MM-DD).

La función utiliza expresiones regulares para verificar si la cadena de caracteres cumple con el patrón establecido para el formato de fecha. Si la cadena no coincide con el patrón, la función retorna `false`, indicando que la fecha no es válida.

Luego, la función utiliza la función “sscanf” para extraer los componentes de año, mes y día de la cadena de caracteres y los almacena en las variables “year”, “month” y “day”. A continuación, realiza las siguientes validaciones:

- Verifica que el año sea mayor o igual a 1900. Si el año es menor a 1900, la función retorna `false`.

- Verifica que el mes esté en el rango válido de 1 a 12. Si el mes no cumple con esta condición, la función retorna `false`.

- Verifica que el día esté en el rango válido de 1 a 31. Si el día no cumple con esta condición, la función retorna `false`.

Si la fecha pasa todas las validaciones, la función retorna `true`, indicando que la fecha es válida.

```
while (true){
    cout << "Ingrese una fecha en formato (YYYY-MM-DD): ";
    cin >> fecha_str;

    if (validar_fecha(fecha_str)){
        break;
    }else {
        cout << "Por favor, ingrese una fecha válida en formato (YYYY-MM-DD)." << endl;
    }
}

//verificamos si los componentes son validos
int year, month, day;
sscanf(fecha_str.c_str(), "%d-%d-%d", &year,&month, &day);

// Convertimos la fecha ingresada a segundos desde la época
double fecha_segundos = fecha_a_segundos(to_string(year).c_str(), to_string(month).c_str(), to_string(day).c_str());

// Obtenemos la fecha actual en segundos desde la época
time_t now = time(0);
tm* ltm = localtime(&now);
double fecha_actual_segundos = fecha_a_segundos(to_string(ltm->tm_year + 1900).c_str(), to_string(ltm->tm_mon + 1).c_str(), to_string(ltm->tm_mday).c_str());

// Calculamos la edad en años
double edad_anios = (fecha_actual_segundos - fecha_segundos) / (365.25 * 24 * 60 * 60);
```

Así se ingresa la fecha en formato ISO 8601. También se muestra la verificación y calcular la edad.

```
// Verificamos el grupo etario
string grupo_etario;
if (edad_anios < 5) {
    grupo_etario = "Bebes";
} else if (edad_anios >= 5 && edad_anios < 12) {
    grupo_etario = "Niños";
} else if (edad_anios >= 12 && edad_anios < 18) {
    grupo_etario = "Adolescentes";
} else if (edad_anios >= 18 && edad_anios < 25) {
    grupo_etario = "Jóvenes";
} else if (edad_anios >= 25 && edad_anios < 40) {
    grupo_etario = "Adultos jóvenes";
} else if (edad_anios >= 40 &&edad_anios < 55) {
    grupo_etario = "Adultos";
} else if (edad_anios >= 55 &&edad_anios < 65) {
    grupo_etario = "Adultos mayores";
} else if (edad_anios >= 65 &&edad_anios < 75) {
    grupo_etario = "Ancianos";
} else {
    grupo_etario = "Longevos";
}
```

En esta parte se muestran las condiciones para que se verifique en que grupo etario califica.

```
ifstream Archivo;
Archivo.open("datosdeprueba.csv");
string linea = "";
```

Estas líneas de código abren un archivo llamado .csv en modo de lectura utilizando el objeto Archivo de la clase ifstream. Luego, se declara una variable de tipo string llamada linea que se utilizará para almacenar cada línea leída del archivo.

```
getline(Archivo, linea); // Salta la primera línea
omp_set_num_threads(6); // Definimos el número de hilos a 6
```

La línea `getline(Archivo, linea);` se utiliza para leer la primera línea del archivo y almacenarla en la variable `linea`. Esta línea se usa comúnmente para omitir la

primera línea del archivo, ya que a menudo contiene encabezados o información no relevante para el procesamiento de los datos.

La línea `omp_set_num_threads(6);` establece el número de hilos que se utilizarán en una región paralela en OpenMP. En este caso, se está configurando el número de hilos en 6, lo que significa que se utilizarán hasta 6 hilos para realizar cálculos en paralelo. Esta configuración puede mejorar el rendimiento al distribuir la carga de trabajo entre múltiples hilos de ejecución.

```
// Bucle externo para leer lotes de líneas del archivo
while (!Archivo.eof()) {
    vector<string> lineas_lote; // Almacena las líneas leídas en el lote

    // Leer un lote de líneas del archivo
    for (int i = 0; i < BATCH_SIZE && getline(Archivo, linea); i++) {
        lineas_lote.push_back(linea);
    }
```

Se usa un bucle externo para trabajar por lotes.

Con el for se leen los lotes.

```
// Bucle interno para procesar cada línea dentro del lote
#pragma omp parallel for reduction(+:total_poblacion, bebes,
for (int i = 0; i < lineas_lote.size(); i++) {
    string linea = lineas_lote[i];

    string fecha;
    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, "-");
```

En este código, se utiliza un bucle interno para procesar cada línea dentro del lote. La directiva `#pragma omp parallel for` se encarga de paralelizar la ejecución del bucle, lo que significa que cada iteración del bucle puede ejecutarse en un hilo diferente.

Dentro del bucle, se extrae una línea del lote y se almacena en la variable `linea`. Luego, se utiliza un `stringstream` para convertir la línea en un flujo de datos que se puede leer utilizando operaciones como `getline`.

En este caso, se utiliza `getline` para extraer la fecha de la línea, utilizando `';'` como delimitador. El resultado se almacena en la variable `fecha`.

A continuación, se divide la fecha en componentes utilizando la función `strtok`. La función `strtok` toma una cadena y un delimitador, y devuelve la primera parte de la cadena hasta el delimitador. Luego, se puede llamar repetidamente a `strtok(NULL, delimitador)` para obtener las partes siguientes.

En este caso, se divide la fecha en tres componentes: día, mes y año. Los componentes se almacenan en variables separadas: `day_str`, `month_str` y `year_str`.

Una vez que se tienen los componentes de la fecha, se pueden realizar cálculos adicionales, como convertir la fecha a segundos desde la época o calcular la edad en años.

Finalmente, la reducción `#pragma omp parallel for reduction(+:total_poblacion, bebes, ninos, adolescentes, jovenes, adultos_jovenes, adultos, adultos_mayores, ancianos, longevos)` se encarga de sumar los valores parciales de las variables `total_poblacion`, `bebes`, `ninos`, `adolescentes`, `jovenes`, `adultos_jovenes`, `adultos`, `adultos_mayores`, `ancianos` y `longevos` en cada iteración del bucle. Esto asegura que los resultados finales sean correctos, incluso cuando se ejecutan en paralelo.

```
// Incrementamos el contador corr
#pragma omp critical
{
    total_poblacion++;
    if (edad < 5) {
        bebes++;
    } else if (edad < 12) {
        ninos++;
    } else if (edad < 18) {
        adolescentes++;
    } else if (edad < 25) {
        jovenes++;
    } else if (edad < 40) {
        adultos_jovenes++;
    } else if (edad < 55) {
        adultos++;
    } else if (edad < 65) {
        adultos_mayores++;
    } else if (edad < 75) {
        ancianos++;
    } else {
        longevos++;
    }
}
```

Posterior a eso incrementamos los contadores dependiendo del grupo etario cmo se muestra en la imagen

```
// Calculamos y mostramos el porcentaje de la población en cada grupo de edad
cout << "\nPorcentaje de la población en cada grupo de edad:\n" << endl;
cout << "Bebes\t\t" << (float)bebes / total_poblacion * 100 << " %\n" << endl;
cout << "Niños\t\t" << (float)ninos / total_poblacion * 100 << " %\n" << endl;
cout << "Adolescentes\t" << (float)adolescentes / total_poblacion * 100 << " %\n" << endl;
cout << "Jóvenes\t\t" << (float)jovenes / total_poblacion * 100 << " %\n" << endl;
cout << "Adultos jóvenes\t" << (float)adultos_jovenes / total_poblacion * 100 << " %\n" << endl;
cout << "Adultos\t\t" << (float)adultos / total_poblacion * 100 << " %\n" << endl;
cout << "Adultos mayores\t" << (float)adultos_mayores / total_poblacion * 100 << " %\n" << endl;
cout << "Ancianos\t" << (float)ancianos / total_poblacion * 100 << " %\n" << endl;
cout << "Longevos\t" << (float)longevos / total_poblacion * 100 << " %\n" << endl;
```

Se imprimen y calculan directamente los porcentajes por cada grupo etario.

Adicionalmente para que imprimiera correctamente caracteres como ó y ñ se uso la biblioteca ‘wchar.h’ con el comando setlocale(LC_ALL, “”);

```
setlocale(LC_ALL, ""); // del wchar.h para usar caracteres como ó
```

Conclusión

En conclusión, en este código se realiza el procesamiento de datos de un archivo CSV que contiene fechas de nacimiento de personas para determinar su grupo etario. El programa permite al usuario ingresar una fecha manualmente en formato ISO8601 para obtener el grupo etario correspondiente.

El programa utiliza la biblioteca `<fstream>` para leer el archivo CSV y la biblioteca `<regex>` para validar el formato de la fecha ingresada manualmente. También se emplea la biblioteca `<omp.h>` para paralelizar el procesamiento de las líneas del archivo utilizando OpenMP.

La función `fecha_a_segundos` convierte una fecha en segundos desde la época, y la función `validar_fecha` verifica si una fecha cumple con el formato y valores válidos.

El código implementa un bucle externo para leer lotes de líneas del archivo, y un bucle interno paralelo para procesar cada línea en paralelo. Se utilizan variables compartidas y reducciones para calcular el número de personas en cada grupo de edad.

Finalmente, se muestra al usuario el grupo etario correspondiente a la fecha ingresada manualmente, así como el porcentaje de población en cada grupo de edad basado en los datos del archivo CSV.

Este programa demuestra el uso de OpenMP para paralelizar el procesamiento de datos y ofrece una funcionalidad para calcular el grupo etario de una fecha ingresada por el usuario.

```
Ingrese una fecha en formato (YYYY-MM-DD): 2020-03-19
La fecha ingresada corresponde al grupo etario: Bebes

Resultado para la fecha 2020-03-19:
Grupo etario    Bebes

Porcentaje de la población en cada grupo de edad:

Bebes           4.1995 %

Niños           6.61292 %

Adolescentes    5.65094 %

Jóvenes         6.60941 %

Adultos jóvenes 14.142 %

Adultos         14.1214 %

Adultos mayores 9.40004 %

Ancianos        9.464 %

Longevos        29.7998 %
```