Análisis de datos taller 1 OpenMP HPC

Presentado por:

Cristian Camilo Perilla Castaño

Daniel Villada Moncada

Luis Miguel Ortiz Alarcón

Carlos A. Vivas Reyes

Presentado a:

Ramiro Andres Barrios Valencia

Universidad Tecnológica de Pereira

Facultad de Ingenierías

High Performance Computing

Pereira

2020

**CARACTERÍSTICAS DEL COMPUTADOR**

|  |  |
| --- | --- |
| **Procesador** | AMD Ryzen R5 2600 overclockeado a 4Ghz |
| **Ram** | 2 modulos de 8gb c/u DDR4 3200MHz |
| **Disco duro** | SSD kingston UV400 hasta 550MB/s lectura y 350MB/s escritura |
| **S.O** | Ubuntu 18.04lts |
| **Número de cpu** | 6 núcleos 12 hilos |

**INTRODUCCIÓN**

Evaluar los datos de salida después de correr el código de OpenMP API Examples 4.0.1

**A Simple Parallel Loop**

El siguiente ejemplo muestra cómo paralelizar un bucle simple usando el paralelo construcción de bucle. La variable de iteración de bucle es privada de forma predeterminada, por lo que no es necesario especificarlo explícitamente en una cláusula privada.

**Example 1.1c**

void simple(int n, float \*a, float \*b)

{

int i;

#pragma omp parallel for

for (i=1; i<n; i++) /\* i is private by default \*/

b[i] = (a[i] + a[i-1]) / 2.0;

C/C++

}

**codigo completo:**

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

#include <string.h>

using namespace std;

void simple(int n, double \*a, double \*b){

int i;

#pragma omp parallel for

for (i=1; i<n; i++) /\* i is private by default \*/

b[i] = (a[i] + a[i-1]) / 2.0;

string str="";

str+=to\_string(i);

str+=" b = ";

str+='\n';

str+=to\_string(b[i]);

cout<<str;

}

int main(){

int n=1000000;

double \*a=nullptr,\*b=nullptr;

a=new double[n];

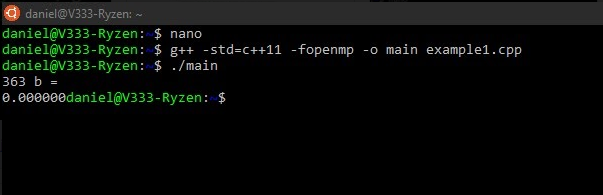
b=new double[n];

simple(n,a,b);

return 0;

}

**Resultado del ejemplo 1** **con pragma**



**The OpenMP Memory Model**

En el siguiente ejemplo, en la Impresión 1, el valor de x podría ser 2 o 5, dependiendo de

el momento de los hilos y la implementación de la asignación a x. Hay dos razones por las cuales el valor en Imprimir 1 podría no ser 5.

Primero, Imprimir 1 podría ejecutarse antes se ejecuta la asignación a x.

En segundo lugar, incluso si la Impresión 1 se ejecuta después de la asignación, no se garantiza que el valor 1 vea el valor 5 porque es posible que no se haya aplicado una descarga ejecutado por el hilo 0 desde la asignación.

La barrera después de la Impresión 1 contiene descargas implícitas en todos los hilos, así como un hilo sincronización, por lo que el programador tiene garantizado que el valor 5 será impreso por tanto Print 2 cómo Print 3.

**Example 2.1c**

**Sin pragma:**

#include <stdio.h>

#include <omp.h>

#include <iostream>

using namespace std;

int main(){

int x;

x=20;

if(omp\_get\_thread\_num()==0){

x=5;

}else{

cout<<"1: Thread #: "<<x<<" = "<<omp\_get\_thread\_num()<<endl;

}

if(omp\_get\_thread\_num()==0){

cout<<"2: Thread #: "<<x<<" = "<<omp\_get\_thread\_num()<<endl;

}else{

cout<<"3: Thread #: "<<x<<" = "<<omp\_get\_thread\_num()<<endl;

}

return 0;

}

**Con pragma:**

#include <stdio.h>

#include <omp.h>

#include <iostream>

using namespace std;

int main(){

int x;

x=20;

#pragma omp parallel num\_threads(12) shared(x)

if(omp\_get\_thread\_num()==0){

x=5;

}else{

cout<<"1: Thread #: "<<x<<" = "<<omp\_get\_thread\_num()<<endl;

}

#pragma omp barrier

if(omp\_get\_thread\_num()==0){

cout<<"2: Thread #: "<<x<<" = "<<omp\_get\_thread\_num()<<endl;

}else{

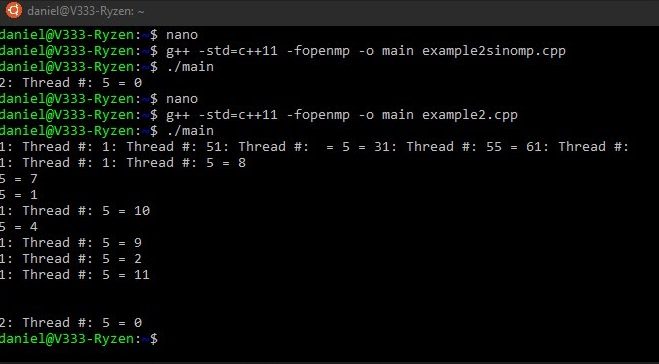
cout<<"3: Thread #: "<<x<<" = "<<omp\_get\_thread\_num()<<endl;

}

return 0;

}

**Resultado del ejemplo 2**



**CONCLUSIONES:**

**Ejemplo 2:**

En el momento de eliminar los pragmas del código, se puede observar que todo se ejecuta en un solo hilo de ejecución, imprimiendo en pantalla el número de hilo correspondiente y el valor de la variable X, en el caso con los pragmas habilitados, se evidencia que el proceso se ejecuta 12 veces más ya que fue el número de hilos asignados en el pragma y con la variable compartida X, el tiempo de ejecución en ambos casos fue prácticamente el mismo aunque en el caso con pragmas debió ser un poco más lento pero no lo suficiente para ser perceptible, sin embargo es curioso la forma de imprimir el código ya que se ve en desorden, esto se puede deber a la temporalidad de terminación por parte de cada hilo.