Universidad Tecnológica de Pereira

High Permormance Computing

Leonardo Castrillòn Giraldo
Cristian Camilo Sanchez

Optimización matrices

Pereira
2020

OpenMP Como paralelizaciòn de memoria compartida

se pretende con el siguiente documento hacer énfasis en la facilidad de uso en términos de ejecución de procesos en paralelo en el sistema utilizando las instrucciones OPENMP y su mejora en implementaciòn gracias a su optimización del uso de memoria compartida,

a continuaciòn se plantea un ejercicio de loop en paralelo con el uso de la instrucciòn

```
#pragma omp parallel for
```

se plantea un loop que ejecuta la compartición de memoria simultánea en cada uno de los recursos de **a** . en medio de una iteraciòn de acceso a la direcciòn previa de cada asegurada por el loop for.

```
void simple(int n, float *a, float *b){
   int i;
   clock_t start, end;
   start = clock();
   #pragma omp parallel for
   for (i=1; i<n; i++) {
   /* i is private by default */
   b[i] = (a[i] + a[i-1]) / 2.0;
   }
   end = clock()-start;
   double time_taken = ((double)end)/CLOCKS_PER_SEC;
   printf("with openMP %f\n",time_taken);
}
```

esta funciòn se aglomera dentro de un análisis de tiempo con la libreria **Time.h,** el cual nos permite tomar los tiempos en segundo obtenidos al empezar la ejecuciòn de la instrucciòn loop antes y después de ella y esta diferencia sería los tiempos en que le tomó al software utilizar los recurso de paralelización

```
#include <time.h>
```

el mismo proceso se hizo con la funciòn sin openmp

```
void whsimple(int n, float *a, float *b){
```

```
   int i;
   clock_t start, end;
   start = clock();

   for (i=1; i<n; i++) {
   /* i is private by default */
   b[i] = (a[i] + a[i-1]) / 2.0;
   }
   end = clock()-start;
   double time_taken = ((double)end)/CLOCKS_PER_SEC;
   printf("without oneMP %f\n",time_taken);

}
```

para este proceso se dieron datos base de un vector tipo float de 5 parámetros y se midieron los tiempos en la fucniòn con OPENMP (simple) y sin openMP (whsimple)

```
int main(int argcv, char * argv[]){
   int n = 5;
   float A[] = {1.0,1.0,1.0,1.0,1.0};
   float B[] = {1.0,1.0,1.0,1.0,1.0};

   simple(n,A,B);
   whsimple(n,A,B);
   return 0;
}
```

por otro lado tenemos un segundo ejemplo

```
int main(){
   int x; x = 2;
   clock_t start, end;
   start = clock();

   #pragma omp parallel num_threads(2) shared(x)
   {    if (omp_get_thread_num() == 0) {
        x = 5;
   } else {
        /* Print 1: the following read of x has a race */
        printf("1: Thread# %d: x = %d\n", omp_get_thread_num(),x );
        }

   #pragma omp barrier
   if (omp_get_thread_num() == 0) {
        /* Print 2 */
        printf("2: Thread# %d: x = %d\n", omp_get_thread_num(),x );
        } else {
            /* Print 3 */
            printf("3: Thread# %d: x = %d\n", omp_get_thread_num(),x );
                }
            }
   end = clock()-start;
   double time_taken = ((double)end)/CLOCKS_PER_SEC;
   printf("with openMP  %f\n",time_taken);

   /* La directiva parallel forma un equipo de hebras y inicia la
ejecución paralela y
   La contrucción barrier especifica de forma explicita un punto de
sincronización para todas las hebras.
   */

   return 0;
}
```

el print 1, el valor de x podría ser 2 o 5, dependiendo de la sincronización de los subprocesos y la implementación de la asignación a x. Hay dos razones por las que el valor en print1 podría no ser 5. Primero, Imprimir 1 podría ejecutarse antes de que se ejecute la asignación a x. En segundo lugar, incluso si la Impresión 1 se ejecuta después de la

asignación, el hilo 1 no garantiza que se vea el valor 5 porque es posible que el hilo 0 no haya ejecutado una descarga desde la asignación. La barrera después de la print 1 contiene descargas implícitas en todos los hilos , así como una sincronización de subprocesos, por lo que el programador tiene garantizado que el valor 5 se imprimirá tanto en print 2 como print 3.

Conclusiones

ejemplo 1

estos resultados nos muestra, ya que los valores son muy pequeños para ver la verdadera funcionalidad de openMP, lo mismo que ocurría con los procesos, sucede aquì. nos muestra la implementaciòn serial màs eficiente que la paralela.

with openMP 0.011426
without oneMP 0.000002

ejemplo 2

1: Thread# 1: x = 5
2: Thread# 0: x = 5
3: Thread# 1: x = 5
with openMP 0.000400

en este ejemplo podemos ver que el barrer en omp barrier nos garantiza que las hebras se sincronizará siempre adoptando el valor predispuesto para ello que en este caso seria 5.