Programación Paralela (2015-2016)
Lenguajes y Sistemas de Información

Grado en Ingeniería Informática

E. T. S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Trabajo propuesto: Tipo de schedule en parallel for de OpenMP

Francisco Javier Bolívar Lupiáñez

3 de junio de 2016

Índice

1. Planteamiento del problema

Considerar el siguiente bucle:

```
for (i = 0; i < N; i++)
        f(i);
```

en el que el tiempo de ejecución de la función f() depende de la iteración i. Paraleliza el bucle anterior con una directiva `parallel for`. Realizar ejecuciones en las que se varíen el número de threads y en las que se modifiquen el tipo de `schedule` a `static`, `dynamic`, `guided`. Comprobar cómo se modifica el tiempo de ejecución en cada caso.

2. Análisis

El problema que se plantea es el de qué reparto es el más apropiado para aquellos algoritmos en los que todas las tareas no tienen el mismo costo computacional.

En OpenMP se puede controlar este reparto de la directiva `parallel for` con los distintos tipos de `schedule` y el tamaño de bloque que se utilizará.

- `static`: El reparto se hace en tiempo de compilación y siempre será el mismo. Las iteraciones se dividen en bloques que se asignan en round-robin a las hebras: Por ejemplo si se tienen 8 iteraciones, 3 hebras y un tamaño de bloque de 2, este sería el reparto:
  - P0: 0, 1, 6
  - P1: 2, 3, 7
  - P2: 4, 5
- `dynamic`: El reparto se hace en tiempo de ejecución. Cada vez que una hebra realiza una tarea coge otra de la cola de trabajos. El tamaño de bloque por defecto es 1. El funcionamiento añade una sobrecarga a tener en cuenta.
- `guided`: Similar a `dynamic`, pero el tamaño de bloque empieza siendo grande y va reduciéndose. El tamaño de bloque se calcula como *iteraciones restantes / número de hebras*, nunca siendo más pequeño que el tamaño de bloque que se le pasa como parámetro. Al igual que `dynamic` tiene sobrecarga adicional, pero menor para el mismo tamaño de bloque.

Se espera, por tanto, que el reparto `static` con bloques de N / P unidades sea el que peor resultados de, mientras que el uso de bloques de tamaño 1 se aproximará a los resultados obtenidos con `dynamic` y `guided` que obtendrán resultados similares, aunque tal vez el `guided` sea el que obtenga los mejores por tener una sobrecarga menor que el `dynamic`.

3. Implementación

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <omp.h>

using namespace std;

/**
  * Funcion que en n iteraciones realiza cuatro operaciones:
  * - asignacion
  * - suma
  * - potencia
  * - modulo
  */
double f(int n) {
  double result = 0;
  for (int i = 0; i < n; i++) {
    result += ((int) pow(i, 2) % 37);
  }
  return result;
}

/*********************************************************************/

int main(int argc, char *argv[]) {
  int P, N, i;
  double t;

  switch(argc) {
    case 3: // se especifica el numero de procesadores
      P = atoi(argv[2]);
      break;
    case 2: // no se especifica el numero de procesadores => se usan
    todos los que tenga el equipo
      P = omp_get_num_procs();
      break;
    default: // numero incorrecto de parametros => se termina la
    ejecucion
      cerr << "Sintaxis: " << argv[0] << " <num iters> <num procs>" <<
    endl;
      return(-1);
  }

  omp_set_num_threads(P);
  N = atoi(argv[1]);

  /*****************************************************************/
  // Static por bloques

  t = omp_get_wtime();
  #pragma omp parallel for schedule(static, N / P)
```

```
49    for (i = 0; i < N; i++) {
50      f(i);
51    }
52    t = omp_get_wtime() - t;
53
54    cout << "Tiempo gastado (static por bloques) = " << t << endl;
55
56    /*********************************************************************/
57    // Static ciclico
58
59    t = omp_get_wtime();
60    #pragma omp parallel for schedule(static, 1)
61    for (i = 0; i < N; i++) {
62      f(i);
63    }
64    t = omp_get_wtime() - t;
65
66    cout << "Tiempo gastado (static ciclico) = " << t << endl;
67
68    /*********************************************************************/
69    // Static dynamic
70
71    t = omp_get_wtime();
72    #pragma omp parallel for schedule(dynamic, 1)
73    for (i = 0; i < N; i++) {
74      f(i);
75    }
76    t = omp_get_wtime() - t;
77
78    cout << "Tiempo gastado (dynamic) = " << t << endl;
79
80    /*********************************************************************/
81    // Static guided
82
83    t = omp_get_wtime();
84    #pragma omp parallel for schedule(guided, 1)
85    for (i = 0; i < N; i++) {
86      f(i);
87    }
88    t = omp_get_wtime() - t;
89
90    cout << "Tiempo gastado (guided) = " << t << endl;
91
92    return(0);
93  }
```

: codigo/main.cpp

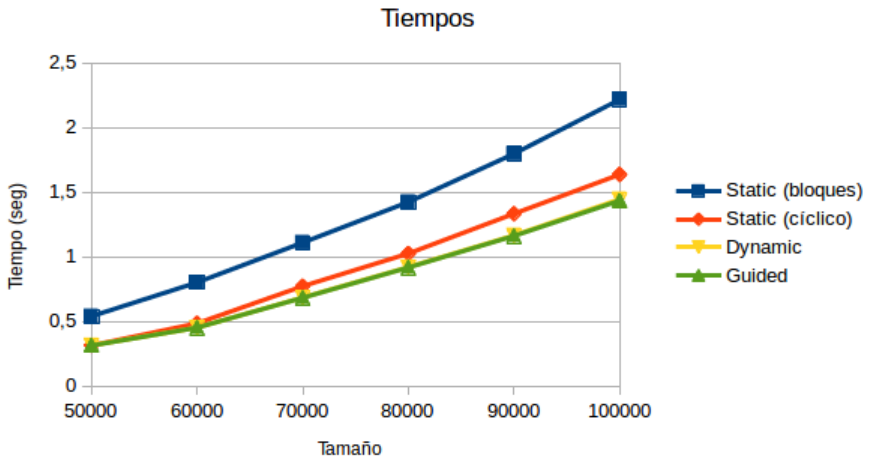
4. Resultados

Para tomar tiempos se ha utilizado un portátil *MSI CX61 2PC* con las siguientes características:

- **Procesador**: Intel® Core™ i7-4712MQ CPU @ 2.30GHz
- **Tamaño de caché**: 6MB
- **Memoria RAM**: 8GB
- **S.O.**: Linux Mint 17.3 Rosa Cinnamon (64-bit)
- **Versión g++**: 4.8.4
- **Versión OpenMP**: 3.1

Se ha lanzado el programa para cuatro hebras OpenMP y tamaños de 50.000 a 100.000 avanzando de 10.000 en 10.000.

N	Static (bloques)	Static (cíclico)	Dynamic	Guided
50000	0,540194	0,314281	0,315153	0,313882
60000	0,801774	0,485337	0,452282	0,451983
70000	1,10983	0,772775	0,685909	0,683268
80000	1,42382	1,02538	0,920239	0,917319
90000	1,79697	1,33359	1,16581	1,16131
100000	2,2165	1,63686	1,44584	1,43517



5. Conclusiones

Como se esperaba, los mejores resultados se han obtenido usando `guided`, aunque la diferencia con `dynamic` es inapreciable. El reparto estático cíclico no trabaja del todo mal si se compara con el estático con bloques de tamaño N / P, pero sigue siendo peor que las dos alternativas mencionadas anteriormente.