Práctica 5: Hebras II

Gustavo Romero

Arquitectura y Tecnología de Computadores

19 de noviembre de 2008

Objetivos

- Poner en práctica la programación de aplicaciones multihebra con la bibliteca Pthreads.
- Implementar una versión multihebra de una biblioteca de matrices que mejore la velocidad de procesamiento de la versión secuencial.

Makefile

http://atc.ugr.es/~gustavo/aco/practicas/practica5/Makefile

```
SHELL = /bin/bash

SRC = $(wildcard *.cc)
EXE = $(basename $(SRC))

CXXFLAGS += -g3 -I. -pg -Wall
LDFLAGS += -lpthread

CPUS = 256
N    = 600

default: $(EXE)

all:    default $(EXE).eps
```


Pthreads: API

```
#include <pthread.h>
```

función	**descripción**
`pthread_create(id, attr, func, arg)`	crea una hebra mediante la ejecución de la función `func`
`pthread_exit(val)`	finaliza un hebra y devuelve un valor para otra que quiera esperarla
`pthread_join(id, val)`	espera una hebra que finaliza y recupera el valor que devuelve
`pthread_self()`	devuelve el identificador de la hebra con tipo `pthread_t`
`pthread_yield()`	ceder el procesador voluntariamente

Biblioteca de matrices multihebra

- Implementar una versión multihebra de los programas que puede descargar desde: http://atc.ugr.es/~gustavo/aco/practicas llamados `matrix.h` y `matrix.cc`.
- Los programas `matrix.h` y `matrix.cc` son una versión monohebra de una biblioteca de operaciones sobre matrices.
- Modifique `matrix.h` de forma que utilice varias hebras para acelerar los cálculos realizados en `matrix.cc`.
- No modifique `matrix.cc` para que los resultados puedan ser comparables con las versiones del resto de sus compañeros.
- Con "`make all`" generará un gráfico que comparativo de las ejecución de su programa con diferentes números de hebras.

Perfiladores de rendimiento: `gprof`, `valgrind`

- ¿Qué modificar? $\longrightarrow$ consultar a un perfilador.
- ¿Cómo utilizar `gprof`?
  1. Compilar con la opción: `CXXFLAGS = -pg`.
  2. Al ejecutar tu programa se creará el fichero `gmon.out` con estadísticas sobre su forma de ejecución.
  3. Analizar las estadísticas con `gprof ejecutable`.
- `valgrind` es mucho mejor pero también más complejo de utilizar.

matrix.h I

```
//-----------------------------------------------------------------------------
// matrix.h
//-----------------------------------------------------------------------------

#ifndef MATRIX_H
#define MATRIX_H

//-----------------------------------------------------------------------------

#include <cassert>
#include <cstdlib>
#include <pthread.h>
#include <iostream>
#include <vector>

//-----------------------------------------------------------------------------

unsigned NUM_THREADS = 2;

//-----------------------------------------------------------------------------

class matrix: public std::vector<std::vector<double> >
{
public:
  matrix(unsigned __r, unsigned __c, double __value = 0):
    std::vector<std::vector<double> >(__r, std::vector<double>(__c, __value))
  {}

  void random()
```


matrix.h II

```
  {
    for (matrix::iterator i = this->begin();
         i != this->end();
         ++i)
      for (std::vector<double>::iterator j = i->begin();
           j != i->end();
           ++j)
        *j = ::random() %9 + 1;
  }

  void transpose()
  {
    unsigned r = this->size(), c = this->front().size();
    matrix m(c, r);

    for (unsigned i = 0; i < r; ++i)
      for (unsigned j = 0; j < c; ++j)
        m[j][i] = (*this)[i][j];

    (*this) = m;
  }

  matrix& operator+=(const matrix& m)
  {
    unsigned r = this->size(), c = this->front().size();

    assert(m.size() == r && m.front().size() == c);

    for (unsigned i = 0; i < r; ++i)
```

matrix.h III

```
      for (unsigned j = 0; j < c; ++j)
        (*this)[i][j] += m[i][j];

    return *this;
  }

  matrix& operator-=(const matrix& m)
  {
    unsigned r = this->size(), c = this->front().size();

    assert(m.size() == r && m.front().size() == c);

    for (unsigned i = 0; i < r; ++i)
      for (unsigned j = 0; j < c; ++j)
        (*this)[i][j] -= m[i][j];

    return *this;
  }

  matrix& operator*=(const matrix& m)
  {
    unsigned r1 = this->size(), c1 = this->front().size();
    unsigned r2 =      m.size(), c2 =      m.front().size();

    assert(c1 == r2);

    matrix __r(r1, c2, 0);

    for (unsigned i = 0; i < r1; ++i)
```


matrix.h IV

```
      for (unsigned j = 0; j < c2; ++j)
        for (unsigned k = 0; k < c1; ++k)
          __r[i][j] += (*this)[i][k] * m[k][i];

    this->swap(__r);

    return *this;
  }

  friend std::ostream& operator<<(std::ostream& os, const matrix& m)
  {
    for (matrix::const_iterator i = m.begin();
         i != m.end();
         ++i)
      {
        for (std::vector<double>::const_iterator j = i->begin();
             j != i->end();
             ++j)
          std::cout << *j << " ";
        std::cout << std::endl;
      }
    return os;
  }
};

matrix operator+(const matrix& __x, const matrix& __y)
{
  matrix __r = __x;
  __r += __y;
```

matrix.h V

```
  return __r;
}

matrix operator-(const matrix& __x, const matrix& __y)
{
  matrix __r = __x;
  __r -= __y;
  return __r;
}

matrix operator*(const matrix& __x, const matrix& __y)
{
  matrix __r = __x;
  __r *= __y;
  return __r;
}

//-----------------------------------------------------------------------------

#endif // MATRIX_H

//-----------------------------------------------------------------------------

// Local Variables:
// mode:C++
// End:
```


matrix.cc I

```
//-----------------------------------------------------------------------------
// matrix.cc
//-----------------------------------------------------------------------------

#include <iostream>
#include <sstream>
#include <matrix.h>

using namespace std;

//-----------------------------------------------------------------------------

int main(int argc, char *argv[])
{
  const int param = 3;

  if (argc != param + 1)
    {
      cerr << "uso: " << argv[0] << " #filas #columnas #hebras" << endl;
      return EXIT_FAILURE;
    }

  unsigned v[param];

  for (int i = 0; i < param; ++i)
    {
      istringstream iss(argv[i + 1]);

      iss >> v[i];
```

matrix.cc II

```
        if (!iss)
          {
            cerr << argv[0] << ": " << argv[i + 1]
                 << " no es un numero v lido" << endl;
            return EXIT_FAILURE;
          }
      }

    NUM_THREADS = v[param - 1]; // n mero de hebras... definido en matrix.h

    srandom(v[0]);      // semilla aleatoria

    matrix a(v[0], v[1]);
    a.random();

    matrix t(a);
    t.transpose();

    matrix s(a);
    s *= s;

    matrix o(v[0], v[1], 1);

    a = a * t + s - o;
}

//-----------------------------------------------------------------------------
```


matrix.cc III

```
// Local Variables:
// mode:C++
// End:
```