

Arquitectura de Sistemas

Práctica 6: Hebras II

Gustavo Romero López

Actualizado: 2 de mayo de 2018

Arquitectura y Tecnología de Computadores

- ⊙ Seguir practicando con la programación multihebra.
- ⊙ Introducción a la optimización.
- ⊙ Implementar una versión multihebra de una biblioteca de matrices que mejore la velocidad de procesamiento frente a la versión secuencial.

⊙ mejor algoritmo

- menos operaciones/operaciones más rápidas
- mejor acceso a datos e instrucciones

⊙ mejor implementación

⊙ balance mejor algoritmo/mejor optimización

- ¿paralelizar fibonacci? \Rightarrow no
- ¿quicksort para ordenar 3 elementos? \Rightarrow no

⊙ análisis de rendimiento

- mejorar todo \Rightarrow pérdida de tiempo/esfuerzo
- descubrir cuellos de botella \Rightarrow si!!!

- ⊙ **Instrumentalizar**: modificar código para evaluar su rendimiento.
 - ¿te gusta hacerlo a mano?
- ⊙ **gprof**: instrumentalización automática (gcc).
 1. compilar con la opción: `-pg`.
 2. al ejecutar se crea `gmon.out`.
 3. analizar las estadísticas con: `gprof` ejecutable.
- ⊙ **valgrind**: conjunto de herramientas para estudiar la gestión de memoria, fallos en las hebras y analizar rendimiento.
 - no requiere instrumentalización pero la ejecución es lenta.
- ⊙ **oprofile**: analizador que funciona con ayuda del núcleo.
 - no requiere instrumentalización y la ejecución es rápida.
- ⊙ **perf**: analizador que funciona con ayuda del núcleo.
 - no requiere instrumentalización y la ejecución es rápida.

- ordenar vectores:

```
bubblesort(v.begin(), v.end());  
quicksort(v.begin(), v.end());  
parallel::sort(v.begin(), v.end());
```

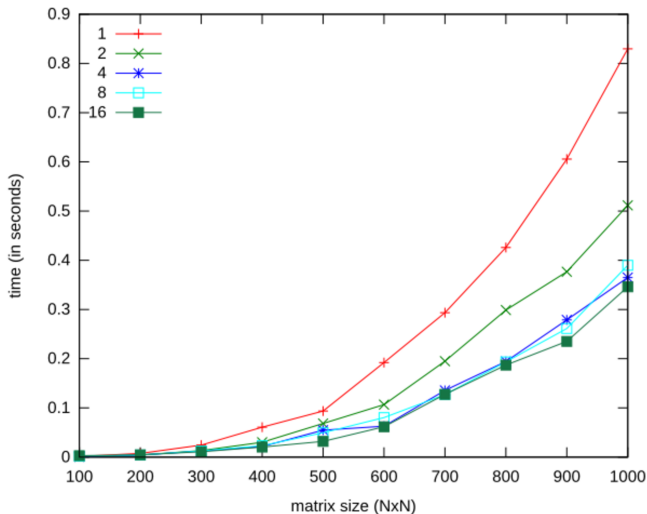
- producto escalar de 2 matrices cuadradas:

```
for (int i = 0; i < N; ++i)  
    for (int j = 0; j < N; ++j)  
        for (int k = 0; k < N; ++k)  
            s[i][j] += v1[i][k] * v2[k][j];  
  
for (int i = 0; i < N; ++i)  
    for (int k = 0; k < N; ++k)  
        for (int j = 0; j < N; ++j)  
            s[i][j] += v1[i][k] * v2[k][j];
```

Test: biblioteca multihebra para matrices cuadradas

- ⊙ Implementar una versión multihebra de los programas que puede descargar desde:
<http://pccito.ugr.es/~gustavo/as/practicas/06> 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 sólo `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 la ejecución de su programa con diferente número de procesadores.
- ⊙ Para que el `makefile` funcione correctamente en el directorio sólo debe estar él mismo junto a los ficheros `matrix.h` y `matrix.cc`.

Paralelización sobre una máquina de 4 núcleos



matrix.h |

```
//-----  
// matrix.h  
//-----  
  
#ifndef MATRIX_H  
#define MATRIX_H  
  
//-----  
  
#include <iostream>  
#include <vector>  
  
//-----  
  
extern unsigned THREAD; // number of threads from matrix.cc  
  
//-----  
  
template<typename _t>  
std::ostream& operator<<(std::ostream& os, const std::vector<_t>& v)  
{  
    os << '<';  
    for (auto i: v)  
        os << i << ' '  
    return os << '>';  
}
```



```

}

//-----

template<typename _t> class matrix: public std::vector<std::vector<_t>>
{
public:
    matrix(unsigned __size = 0, _t __t = _t()):
        std::vector<std::vector<_t>>(__size, std::vector<_t>(__size, __t)) {}

    template<class _rng> void random(_rng& __rng)
    {
        for (auto& i: *this)
            for (auto& j: i)
                j = __rng();
    }

    void transpose()
    {
        for (unsigned i = 0; i < this->size(); ++i)
            for (unsigned j = i + 1; j < this->size(); ++j)
                std::swap((*this)[i][j], (*this)[j][i]);
    }

    matrix& operator+=(const matrix& m)
    {

```

matrix.h III

```
    for (unsigned i = 0; i < this->size(); ++i)
        for (unsigned j = 0; j < this->size(); ++j)
            (*this)[i][j] += m[i][j];
    return *this;
}

matrix& operator--(const matrix& m)
{
    for (unsigned i = 0; i < this->size(); ++i)
        for (unsigned j = 0; j < this->size(); ++j)
            (*this)[i][j] -= m[i][j];
    return *this;
}

matrix& operator*=(const matrix& m)
{
    matrix r(m.size(), 0);
    for (unsigned i = 0; i < this->size(); ++i)
        for (unsigned j = 0; j < this->size(); ++j)
            for (unsigned k = 0; k < this->size(); ++k)
                r[i][j] += (*this)[i][k] * m[k][j];
    this->swap(r);
    return *this;
}

friend std::ostream& operator<<(std::ostream& os, const matrix& m)
```

matrix.h IV

```
{
    for (auto& i: m)
    {
        os << "[ ";
        for (auto& j: i)
            os << j << ' ';
        os << "]" << std::endl;
    }
    return os;
}

};

//-----

template<typename _t>
matrix<_t> operator+(const matrix<_t>& __x, const matrix<_t>& __y)
{
    matrix<_t> __r = __x;
    __r += __y;
    return __r;
}

template<typename _t>
matrix<_t> operator-(const matrix<_t>& __x, const matrix<_t>& __y)
{
    matrix<_t> __r = __x;
```

matrix.h V

```
    __r -= __y;
    return __r;
}

template<typename _t>
matrix<_t> operator*(const matrix<_t>& __x, const matrix<_t>& __y)
{
    matrix<_t> __r = __x;
    __r *= __y;
    return __r;
}

//-----

#endif // MATRIX_H

//-----

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

```
//-----  
// matrix.cc  
// info: http://www.boost.org/doc/libs/1\_55\_0/doc/html/program\_options.html  
//       http://www.radmangames.com/programming/how-to-use-boost-program\_options  
//-----  
  
#include <chrono>  
#include <functional>  
#include <iostream>  
#include <random>  
#include <sstream>  
#include <stdexcept>  
  
#include <boost/program_options.hpp>  
  
#include "matrix.h"  
  
namespace po = boost::program_options;  
  
//-----  
  
unsigned N = 10;      // number of rows and columns  
unsigned THREAD = 1;  // number of threads  
bool verbose = false; // verbose output
```

```
//-----  
  
void work()  
{  
    std::default_random_engine generator(N);  
    std::uniform_int_distribution<int> distribution(0, 9);  
    auto rng = std::bind(distribution, generator);  
  
    auto t1 = std::chrono::high_resolution_clock::now();  
  
    matrix<int> a(N);  
    a.random(rng);  
  
    if (verbose)  
        std::cout << "a = " << std::endl << a;  
  
    matrix<int> t(a);  
    t.transpose();  
  
    if (verbose)  
        std::cout << "t = " << std::endl << t;  
  
    matrix<int> s(a);  
    s *= s;  
  
    if (verbose)
```

```

        std::cout << "s = " << std::endl << s;

    matrix<int> o(N, 1);

    if (verbose)
        std::cout << "o = " << std::endl << o;

    a = a * t + s - o;

    auto t2 = std::chrono::high_resolution_clock::now();

    if (verbose)
        std::cout << "a = " << std::endl << a << std::endl;

    std::cout << "tiempo: " << std::chrono::duration_cast<std::chrono::nanoseconds>(t2
        - t1).count()
        << " ns" << std::endl;

}

//-----

void parser(int argc, char *argv[])
{
    po::options_description desc("program options");

```

```
desc.add_options()
    ("help,h", "help message")
    ("number,n", po::value<unsigned>(&N)->default_value(10), "number of rows and
        columns")
    ("threads,t", po::value<unsigned>(&THREAD)->default_value(1), "number of threads
        ")
    ("verbose,v", "verbose output")
;

po::variables_map vm;
po::store(po::parse_command_line(argc, argv, desc), vm);
po::notify(vm);

if (vm.count("help"))
    throw desc;

if (vm.count("verbose"))
    verbose = true;
else
    verbose = false;
}

//-----

int main(int argc, char *argv[])
{
```



```
try
{
    parser(argc, argv);
    work();
}
catch (po::options_description& desc)
{
    std::cout << desc;
}
catch (std::exception& e)
{
    std::cerr << argv[0] << ": " << e.what() << std::endl;
    return 1;
}
}

//-----

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