

Teoria Współbieżności

Ćwiczenie 8

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie studentów z programowaniem współbieżnym w OpenMP (C).

2 Wprowadzenie teoretyczne

OpenMP jest popularnym interfejsem umożliwiającym tworzenie programów współbieżnych na maszynie o pamięci współdzielonej. Jest wykorzystywany w językach C, C++ oraz Fortran. Standard ten jest przenośny oraz skalowalny. OpenMP implementuje wielowątkowość, czyli metodę zrównoleglania w której wątek programu "forkuje" się na kilka wątków potomnych wspólnie wykonujących określone zadanie. Po wykonaniu współbieżnej części wątki potomne kończą swoje działanie.

3 Plan ćwiczenia

- Proszę uwspółbieżnić algorytm mnożenia macieży (plik mm1.c, funkcja mm) za pomocą OpenMP.
- Proszę uważać na problem wyścigu. Proszę używać zmiennych prywatnych (dla wątku) oraz globalnych.
- Sumę do wspólnej zmiennej z wielu wątków wykonuje się poprzez polecenie **reduce**.
- Jak będzie się zrównoleglał algorytm (wersja optymistyczna)? Jak poprawić można poprawić ten parametr?
- Proszę zbadać skalowalność algorytmu.

4 Elementy pomocne

Zapoznaj się z metodą kompilacji kodu z OpenMP. Dla `gcc` oraz `g++` istnieje opcja kompilacji `-fopenmp`.

```
maciekw@Banach:~/studenci/TW/lab8> gcc -fopenmp mm1.c
```

Zapoznaj się ze słowami kluczowymi takimi jak *parallel*, *for*, *default*, *private*, *shared*, *collapse*, *barrier*, *reduce*.

```
#pragma omp [parallel] for [clauses]
    for (...) {
        do_sth
    }

#pragma omp parallel for default(none) private(i) shared(j)
    for (...) {
        do_sth
        #pragma omp barrier
    }

#pragma omp parallel for collapse(2)
    for (...) {
        for (...) {
            do_sth
        }
    }
```

Sprawdź jak działa opcja uruchomieniowa `OMP_NUM_THREADS`.

```
maciekw@Banach:~/studenci/TW/lab8> OMP_NUM_THREADS=2 ./a.out
```

Literatura

- [1] OpenMP web site: <https://www.openmp.org>
- [2] OpenMP 3.0 Summary Card <http://www.openmp.org/mp-documents/OpenMP3.0-SummarySpec.pdf>
- [3] Wstęp do OpenMP na stronach Intelu <https://software.intel.com/en-us/articles/getting-started-with-openmp>

Dodatki

Listing 1: Kod

```
#include <stdio.h>
#include <stdlib.h>

int SIZE;

#define _first_(i,j) first[ (j)*SIZE + (i) ]
#define _second_(i,j) second[ (j)*SIZE + (i) ]
#define _multiply_(i,j) multiply[ (j)*SIZE + (i) ]

#include <sys/time.h>
#include <time.h>

static double gtod_ref_time_sec = 0.0;

/* Adapted from the bl2_clock() routine in the BLIS library */

double dclock()
{
    double the_time, norm_sec;
    struct timeval tv;
    gettimeofday( &tv, NULL );
    if ( gtod_ref_time_sec == 0.0 )
        gtod_ref_time_sec = ( double ) tv.tv_sec;
    norm_sec = ( double ) tv.tv_sec - gtod_ref_time_sec;
    the_time = norm_sec + tv.tv_usec * 1.0e-6;
    return the_time;
}

int mm(double * first, double * second, double * multiply)
{
    register unsigned int i,j,k;
    double sum = 0;

    for (i = 0; i < SIZE; i++) { //rows in multiply
        for (j = 0; j < SIZE; j++) { //columns in multiply
            sum = 0;
```

```

        for (k = 0; k < SIZE; k++) { //columns in first and rows in second
            sum = sum + _first_(i,k)*_second_(k,j);
        }
        _multiply_(i,j) = sum;
    }
}
return 0;
}

int main( int argc, const char* argv[] )
{
    int i,j,iret;
    double ** first;
    double ** second;
    double ** multiply;

    double * first_;
    double * second_;
    double * multiply_;

    double dtime;

    SIZE = 1000;

    first_ = (double*) malloc(SIZE*SIZE*sizeof(double));
    second_ = (double*) malloc(SIZE*SIZE*sizeof(double));
    multiply_ = (double*) malloc(SIZE*SIZE*sizeof(double));

    first = (double**) malloc(SIZE*sizeof(double*));
    second = (double**) malloc(SIZE*sizeof(double*));
    multiply = (double**) malloc(SIZE*sizeof(double*));

    for (i = 0; i < SIZE; i++) {
        first[i] = first_ + i*SIZE;
        second[i] = second_ + i*SIZE;
        multiply[i] = multiply_ + i*SIZE;
    }

    for (i = 0; i < SIZE; i++) { //rows in first
        for (j = 0; j < SIZE; j++) { //columns in first
            first[i][j]=i+j;

```

```
        second[ i ][ j ]=i-j;
    }
}

dtime = dclock();

iret=mm( first_ ,second_ ,multiply_ );

dtime = dclock()-dtime;
printf( "Time: %le \n", dtime );

fflush( stdout );

free( first_ );
free( second_ );
free( multiply_ );

free( first );
free( second );
free( multiply );

return iret;
}
```