Optymalizacja Kodu Na Różne Architektury
Ćwiczenie 1

1 Wprowadzenie

Rozważmy następującą procedurę uruchamiającą mnożenie macierzy

Listing 1: Bazowa wersja kodu

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


//subroutine for optimization
int mm(double ** first, double ** second, double ** multiply, int SIZE)
```

```
{
  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
      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;
      sum = 0;
    }
  }
  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;

  int SIZE = 1500;

//allocate blocks of continous memory
  first_ = (double*) malloc(SIZE*SIZE*sizeof(double));
  second_ = (double*) malloc(SIZE*SIZE*sizeof(double));
  multiply_ = (double*) malloc(SIZE*SIZE*sizeof(double));

//allocate 2D matrices
  first = (double**) malloc(SIZE*sizeof(double*));
  second = (double**) malloc(SIZE*sizeof(double*));
  multiply = (double**) malloc(SIZE*sizeof(double*));

//set pointers to continous blocks
  for (i = 0; i < SIZE; i++) {
```

```
    first[i] = first_ + i*SIZE;
    second[i] = second_ + i*SIZE;
    multiply[i] = multiply_ + i*SIZE;
  }

//fill matrices with test data
  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;
    }
  }

//measure mm subroutine computation time
  dtime = dclock();
  iret = mm(first,second,multiply,SIZE);
  dtime = dclock()-dtime;
  printf( "Time: %le \n", dtime);


  fflush( stdout );

//cleanup
  free(first_);
  free(second_);
  free(multiply_);

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

  return iret;
}
```

2 Ćwiczenia

Przy kolejnych krokach proszę wykonywać kopie aktualnych wersji stanu kodu. Będą potrzebne przy ostatnim punkcie oraz na kolejnych zajęciach. Proszę na bieżąco notować czasy wykonywania się obliczeń.

2.1

Proszę skompilować procedurę bez optymalizacji.

```
maciekw@Banach:~/studenci/OORA/skrypty/lab1> gcc mm1.c
maciekw@Banach:~/studenci/OORA/skrypty/lab1> ./a.out
Time: 8.436804e+00
maciekw@Banach:~/studenci/OORA/skrypty/lab1>
```

2.2

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab1> cp mm1.c mm2.c
```

oraz dokonać w mm2.c pierwszej próby optymalizacji procedury mm poprzez umieszczenie zmiennych kierujących pętlami w rejestrach

```
register unsigned int i,j,k;
```

Proszę skompilować procedurę z pierwszą optymalizacją.
Jaki jest czas działania?

2.3

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab1> cp mm2.c mm3.c
```

oraz dokonać w mm3.c drugiej próby optymalizacji procedury mm poprzez wykonanie lokalnej kopii oraz umieszczenie rozmiaru pętli w rejestrach

```
register unsigned int local_size;
```

Proszę skompilować procedurę z optymalizacją.
Jaki jest czas działania?

2.4

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab1> cp mm3.c mm4.c
```

oraz dokonać w mm4.c trzeciej próby optymalizacji procedury mm poprzez zamianę pętli z krokiem +1

```
for (i = 0; i < SIZE; i++) {
  for (j = 0; j < SIZE; j++) {
    for (k = 0; k < SIZE; k++){
      ...
    }
  }
}
```

na pętle kończące się zerowaniem licznika.

```
for (i = local_size; i-- ; )
```

Proszę skompilować procedurę z optymalizacją.
Jaki jest czas działania?

2.5

Proszę wykonać kopię procedury (UWAGA - patrz na wersje plików)

```
maciekw@Banach:~/studenci/OORA/skrypty/lab1> cp mm3.c mm5_4.c
maciekw@Banach:~/studenci/OORA/skrypty/lab1> cp mm3.c mm5_8.c
```

oraz dokonać w mm5_*.c czwartej próby optymalizacji procedury mm poprzez zamianę pętli względem k z krokiem +1

```
for (i = 0; i < SIZE; i++) {
  for (j = 0; j < SIZE; j++) {
    for (k = 0; k < SIZE; k++){
      sum = sum + _first_(i,k)*_second_(k,j);
    }
    _multiply_(i,j) = sum;
    sum = 0;
  }
}
```

na pętle względem k z krokiem +4 (mm5_4.c) i +8 (mm5_8.c).

```
for (i = 0; i < local_size; i++) {
  for (j = 0; j < local_size; j++) {
    for (k = 0; k < local_size; ) {
      if(k<local_size-4) {
        sum = sum + first[i][k]*second[k][j];
        sum = sum + first[i][k+1]*second[k+1][j];
        //PROSZE UZUPELNIC
        k=k+4;
      }
      else {
```

```
            sum = sum + first[i][k]*second[k][j];
            k++;
          }
        }
        multiply[i][j] = sum;
        sum = 0;
      }
    }
```

Proszę skompilować procedurę z optymalizacją.
Jaki jest czas działania?

2.6

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab1> cp mm5_4.c mm6.c
```

oraz dokonać w mm6.c piątej próby optymalizacji procedury mm poprzez umieszczenie wspólnie używanych zmiennych wewnątrz pętli w rejestrach

```
register double XXX;
```

oraz zwinięcie ciągu sum do pojedynczego polecenia.
Proszę skompilować procedury z optymalizacją.
Jaki jest czas działania?

2.7

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab1> cp mm6.c mm7.c
```

oraz dokonać w mm7.c szóstej próby optymalizacji procedury mm poprzez zamianę wierszy i kolumn w macierzy second.
Proszę skompilować procedurę z optymalizacją.
Jaki jest czas działania?

2.8

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab1> cp mm6.c mm8.c
```

oraz dokonać w mm8.c siódmej próby optymalizacji procedury mm poprzez użycie makr i bezpośredniego dostępu do ciągłej pamięci.

```
#define _first_(i,j) first[ (j)*local_size + (i) ]
#define _second_(i,j) second[ (j)*local_size + (i) ]
#define _multiply_(i,j) multiply[ (j)*local_size + (i) ]
...
int mm(double * first, double * second, double * multiply, int SIZE)
...
sum = sum + _first_(i,k)*_second_(k,j);
...
iret = mm(first_,second_,multiply_,SIZE);
...
```

Proszę skompilować procedurę z optymalizacją.
Jaki jest czas działania?

2.9

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/COORA/skrypty/lab1> cp mm8.c mm9.c
```

oraz dokonać w mm9.c ósmej próby optymalizacji procedury mm poprzez połączenie optymalizacji z dwóch ostatnich punktów.
Proszę skompilować procedurę z optymalizacją.
Jaki jest czas działania?

2.10

Proszę wszystkie uzyskane na Laboratoriach kody skompilować z opcją optymalizacji **-O2**.

```
maciekw@Banach:~/studenci/COORA/skrypty/lab1> gcc –O2 mm1.c
maciekw@Banach:~/studenci/COORA/skrypty/lab1> ./a.out
...
```

Jaki jest ich czas działania?

3 Podsumowanie

Które metody optymalizacji dały najlepsze rezultaty?
Dlaczego? Jakie mechanizmy za nimi stały?