

1 Sparc

1.1 Laborka: min, max oraz max - min

1.1.1 Funkcja w języku C

```
#include <stdio.h>
extern int minmax(int *tab, int n, int *max, int *min);

int main()
{
    int i, N, *tab;
    int max, min, span;
    scanf("%i", &N);
    if (N < 0) {
        printf("N<0!\n");
        return -1;
    }
    tab = malloc(N*sizeof(*tab));
    for(i = 0; i < N; ++i)
        scanf("%i", tab + i);
    span = minmax(tab, N, &max, &min);
    printf("min=%i , max=%i , span=%i\n",
        min, max, span);
    free(tab);
    return 0;
}
```

1.1.2 Odpowiednik w SPARCu

```
.global minmax
.proc 4
; rejestry:
; %i0 - adres do tablicy
; %i1 - ilosc liczb (N)
; %i2 - adres do max
; %i3 - adres do min
;
; %l6 - pomocnicza do przechowania przesuniecie w bajtach
; %l7 - pomocnicza do porownywania z min/max
; %l1 - max
; %l2 - min
minmax:
    save %sp, -96, %sp ; przesuniecie okienka
    ; zaladuj wartosci dla max i min, gdy n <= 0
    mov 0, %l1
    mov 0, %l2
    ; sprawdz czy n > 0
    subcc %i1, 1, %i1
    blt end
    nop
    ; zaladuj startowe max (%l1) i min(%l2) z pierwszej liczby
    ld [%i0], %l1
    mov %l1, %l2
petla:
    ; sprawdz koniec petli
    blt end
    ; wylicz adres i zaladuj kolejna liczbe
    smul %i1, 4, %l6
    ld [%i0+%l6], %l0 ; %l0 - obecna liczba

    ; update max
    subcc %l0, %l1, %l7 ; %l1 - max
    blt next
    nop
    mov %l0, %l1
next:
    ; update min
    subcc %l0, %l2, %l7 ; %l2 - min
    bgt next2
    nop
    mov %l0, %l2
next2:
    ba petla
    subcc %i1, 1, %i1
end:
    ; zapisz wynik
    st %l1, [%i2]
    st %l2, [%i3]
    sub %l1, %l2, %i0
    ret
    restore ; odtworzenie okienka
```

1.2 Treść

Funkcja zwraca $a(n)$ wyliczoną ze wzoru rekurencyjnego, pobiera dwa argumenty: n oraz k , obydwa typu *unsigned int*.

$$a(n) = a(n-1)^k + n \cdot k, \quad a(0) = 1, \quad n = 1, 2, 3, \dots$$

1.2.1 Rozwiązanie nr 1 by Dexus

```
.global _start

_start:
    MOV    0x05,    %g1                ;! g1 - K
    MOV    0x0A,    %o7                ;! rej o7 i i7 -> N (lokalne)
    MOV    %o7,     %g7                ;! N absolutne

_petla:
    SAVE    %sp,     -96,    %sp        ;! otworzenie okna

    SUBcc   %i7,     0x00,    %g0        ;! sprawdzenie, czy to dno
    BE      _nzero
    NOP

    SUB     %i7,     0x01,    %o7        ;! wykonanie rekurencji
    BA      _petla
    NOP

_nzero:
    MOV     0x00,    %i5
    MOV     0x00,    %g2                ;! g2 temp n

_petlapowrot:
    RESTORE                ;! zamknięcie koła

    MOV     %i5,     %l0                ;! obliczenia
    MOV     0x01,    %l1                ;! temp k

_petlamnoz:
    UMUL    %i5,     %l0,    %l0        ;! obliczenia zgodnie ze wzorem
    ADD     %l1,     0x01,    %l1
    SUBcc   %l1,     %g1,    %g0
    BNE     _petlamnoz

    UMUL    %g2,     %g1,    %l2
    ADD     %l1,     %l2,    %o5

    ADD     %g2,     0x01,    %g2

    SUBcc   %g2,     %g7,    %g0        ;! czy koniec odkręcania koła
    BLE     _petlapowrot
    NOP

    MOV     %i5,     %g1                ;! g1 - wynik koncowy
    NOP                                ;! koniec
```

1.2.2 Rozwiązanie nr 2 by Trimack

```
.global _start
.proc 4
;! Rejestry:
;! %i0 - n. Numer elementu ciągu, który chcemy pobrać
;! %i1 - k. Parametr równania ciągu
;! %l0 - zmienna tymczasowa do porównań
;! %l1 - licznik potęgowania
;! %l2 - wynik potęgowania
;! %l3 - wynik mnożenia n * k
;! Wzór: a(n) = a(n - 1) ^ k + n * k; a(0) = 1
;! Wartość zwracana: a(n)

_start:
    save    %sp,    -96,    %sp    ;! Przesunięcie okienka

    subcc   %i0,    1,    %o0    ;! %o0 -> n - 1. if (n == 0)
    bneg    zwroc1
    nop

    subcc   %i1,    1,    %l0    ;! if (k == 0)
    bneg    zwroc1
    nop

    mov     %i1,    %o1    ;! %o1 -> k
    call    _start
    nop

    ;! %o0 -> a(n - 1)
    mov     %i1,    %l1    ;! %l1 -> k
    mov     %o0,    %l2    ;! %l2 -> a(n - 1)

power:
    ;! dekrementacja licznika i sprawdzenie, czy skończyliśmy potęgować: if
    ;! (%l1 - 1 = 0)
    subcc   %l1,    1,    %l1
    be      powerEnd
    nop
    umul    %l2,    %o0,    %l2    ;! %l2 *= a(n - 1)
    ba      power
    nop

powerEnd:
    ;! %l2 -> a(n - 1) ^ k
    umul    %i0,    %i1,    %l3    ;! %l3 = n * k
    add     %l2,    %l3,    %i0    ;! %i0 = wynik
    ba      end
    nop

zwroc1:
    mov     1,    %i0

end:
    ret
    restore
```

1.3 Treść

Funkcja realizująca operację w języku C:

```
int f(int *tab, int n)
{
    int i, suma = 0;
    for(i = 0; i < n; i++)
    {
        suma += i*tab[i];
    }
    return suma;
}
```

1.3.1 Rozwiązanie 1

```
.global _start
_start:
;!i1 wskaźnik na pocz tablicy
;!i2 n
;!i0 - wyjściowa suma (RESTORE spowoduje że będzie to w rej.
;!wyjściowych funkcji nadrzędnej
;!l0, l1 - wskaźnik na el. tablicy, n
;!l2 - iterator
;!l3 - suma
    mov     %i1,    %l0
    mov     %i2,    %l1
    mov     0x00,   %l2
    mov     0x00,   %l3
_loop:
;!if sprawdzają czy i < n
    subcc   %l2,    %l1,    %g0
    bge     _koniec
    nop

    ld      [%l0],   %l7
    umul    %l7,    %l2,    %l7
    add     %l7,    %l3,    %l3

    add     %l2,    0x01,   %l2

    ba      _loop
    add     %l0,    0x04,   %l0
;!można zamienić miejscami i dać nop, ale tak optymalniej

_koniec:
    mov     %l3,    %i0
```

1.3.2 Rozwiązanie 2

```
.global f
.proc 4
;! Rejestry:
;! %i0 - adres tablicy wejsciowej
;! %i1 - rozmiar tablicy (n)
;! %l0 - zmienna tymczasowa do porównań
;! %l1 - suma
;! %l2 - licznik (i)
;! %l3 - i * a[i]
;! int f(int *tab, int n)

f:
    save    %sp,    -96,    %sp
    mov     0,     %l1          ;! suma = 0
    mov     0,     %l2          ;! i = 0

    subcc   %i1,    1,         %l0    ;! if (n == 0)
    bneg    koniec
    nop

petlaFor:
    subcc   %l2,    %i1,    %l0    ;! if (i >= n)
    bge     koniec
    nop

    ld      [%i0],   %l3          ;! %l3 = a[i]
    smul    %l3,     %l2,    %l3    ;! %l3 = i * a[i]

    add     %l1,     %l3,    %l1    ;! suma += i * a[i]
    add     %i0,     4,      %i0    ;! %i0 wskazuje na kolejny e. tablicy

    add     %l2,     1,      %l2    ;! i++
    ba      petlaFor
    nop

koniec:
    mov     %l1,     %i0          ;! zwrócenie wyniku
    ret
    restore
```

2 PVM

2.1 Laborki, min max z macierzy

Program przekazuje kolejne wiersze macierzy do programów potomnych, które znajdują lokalne minimum i maksimum. Program zbiera wszystkie minima i maksima do tablicy o rozmiarze wysokości macierzy. Pod koniec sam ręcznie wylicza min i max z tych dwóch tablic.

Należy pamiętać, że programy potomne muszą fizycznie znajdować się na dyskach innych komputerów w sieci PVM.

```
/* - Autorzy:
   -- Forczu Forczmański
   -- Wuda Wudecki
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "pvm3.h"
#define WYSOKOSC 5      // liczba wierszy
#define SZEROKOSC 5     // liczba kolumn
/// Program rodzica
main()
{
    // dane potrzebne do obliczeń
    int matrix[WYSOKOSC][SZEROKOSC];
    int min_result[WYSOKOSC], max_result[WYSOKOSC];
    int minimum, maksimum;
    // wypełnienie macierzy danymi
    int i, j;
    for ( i = 0; i < WYSOKOSC; ++i )
        for ( j = 0; j < SZEROKOSC; ++j )
            matrix[i][j] = rand() % 30;
    // wypisanie macierzy na konsoli
    for ( i = 0; i < WYSOKOSC; ++i )
    {
        for ( j = 0; j < SZEROKOSC; ++j )
            printf("%d_", matrix[i][j]);
        printf("\n\n");
    }
    // pobranie informacji
    int ilhost, ilarch;
    struct pvmhostinfo * info;
    pvm_config(&ilhost, &ilarch, &info);
    printf("Liczba_hostow: %d\n", ilhost);

    int id1 = 0;
    int tid;
```

```

// Dla każdego hosta - inicjujemy go
for ( i = 0; i < ilhost; i++ )
{
    pvm_spawn( "/home/pvm/pvm3/sekcja11/bin/LINUX/hello_other", 0,
        PvmTaskHost, info[i].hi_name, 1, &tid);
    if ( tid < 0 )
    {
        ilhost--;
        continue;
    }
    printf("tid: %d\n", tid);
    pvm_initsend(PvmDataDefault);
    // wysyłamy:
    // id wiersza
    pvm_pkint(&id1, 1, 1);
    // elementy wiersza
    pvm_pkint(&matrix[id1][0], SZEROKOSC, 1);
    pvm_send(tid, 100);
    id1++;
}
///// Wykonywanie programu aż do przedostatniej pętli
int bufid, child_tid, child_id1, tmp;
while ( id1 < WYSOKOSC )
{
    bufid = pvm_recv(-1, 200);
    pvm_buinfo(bufid, &tmp, &tmp, &child_tid);
    printf("recv: %d\n", child_tid);
    // pobranie id wiersza
    pvm_upkint(&child_id1, 1, 1);
    // pobranie nowych min / max
    pvm_upkint(&min_result[child_id1], 1, 1);
    pvm_upkint(&max_result[child_id1], 1, 1);
    // wysłanie nowych danych
    pvm_initsend(PvmDataDefault);
    // id kolejnego wiersza
    pvm_pkint(&id1, 1, 1);
    // nowy wiersz
    pvm_pkint(&matrix[id1][0], SZEROKOSC, 1);
    pvm_send(child_tid, 100);
    id1++;
}
///// Odebranie ostatnich danych
for ( i = 0; i < id1 - ilhost + 1; i++ )
{
    bufid = pvm_recv(-1, 200);
    pvm_buinfo(bufid, &tmp, &tmp, &child_tid);
    printf("recv: %d\n", child_tid);
    // pobranie id wiersza
    pvm_upkint(&child_id1, 1, 1);
    // pobranie nowych min / max
    pvm_upkint(&min_result[child_id1], 1, 1);
    pvm_upkint(&max_result[child_id1], 1, 1);
}

```



```

// uzyskanie minimum z wiersza
minimum = min_result[0];
maksimum = max_result[0];
for (j = 1; j < WYSOKOSC; j++)
{
    if ( max_result[j] > maksimum )
        maksimum = max_result[j];
    if ( min_result[j] < minimum )
        minimum = min_result[j];
}
printf(" Uzyskane_wartosci:\nMIN: %d, MAX: %d\n", minimum, maksimum);
pvm_exit();
return 0;

```

Program potomka

```

#include <stdio.h>
#include <math.h>
#include "pvm3.h"
#define WYSOKOSC 5 // liczba wierszy
#define SZEROKOSC 5 // liczba kolumn
/// Program potomka
int main()
{
    int masterid, id1, j, curr_row[SZEROKOSC], curr_min, curr_max;
    // pobierz id rodzica
    masterid = pvm_parent();
    if (masterid == 0)
        exit(1);
    while(1)
    {
        pvm_recv(masterid, 100);
        // pobranie wartosci:
        // id wiersza
        pvm_upkint(&id1, 1, 1);
        pvm_upkint(&curr_row[0], SZEROKOSC, 1);
        // uzyskanie minimum z wiersza
        curr_min = curr_max = curr_row[0];
        for (j = 1; j < SZEROKOSC; j++)
        {
            if ( curr_row[j] > curr_max )
                curr_max = curr_row[j];
            if ( curr_row[j] < curr_min )
                curr_min = curr_row[j];
        }
        // wysłanie nowych danych
        pvm_init send(PvmDataDefault);
        pvm_pkint(&id1, 1, 1);
        pvm_pkint(&curr_min, 1, 1);
        pvm_pkint(&curr_max, 1, 1);
        pvm_send(masterid, 200);
    }
    pvm_exit();
    return 0;
}

```

2.2 Laborki, odejmowanie macierzy

2.2.1 Treść

Odejmowanie macierzy.

```
#include <stdio.h>
#include <math.h>
#include "pvm3.h"
#define MATRIX_SIZE 20
int main()
{
    int i,j;
    int count = 0;          //licznik wierszy macierzy
    int rescount;
    int tidmaster, ilhost, ilarch, bufid, t_id, bytes, msgtag;
    struct pvmhostinfo info;

    int a[MATRIX_SIZE][MATRIX_SIZE], b[MATRIX_SIZE][MATRIX_SIZE], r[
        MATRIX_SIZE][MATRIX_SIZE];
    FILE *txt = fopen("result.txt", "w");

    for (i=0; i<MATRIX_SIZE; i++)
    {
        for (j=0; j<MATRIX_SIZE; j++)
        {
            a[i][j] = rand();
            b[i][j] = rand();
        }
    }
    fprintf(txt, "Macierz A:\n
    _____\n\n");
    for (i=0; i<MATRIX_SIZE; i++)
    {
        for (j=0; j<MATRIX_SIZE; j++)
            fprintf(txt, "%d\t", a[i][j]);
        fprintf(txt, "\n");
    }
    fprintf(txt, "Macierz B:\n
    _____\n\n");
    for (i=0; i<MATRIX_SIZE; i++)
    {
        for (j=0; j<MATRIX_SIZE; j++)
            fprintf(txt, "%d\t", b[i][j]);
        fprintf(txt, "\n");
    }
    tidmaster = pvm_mytid();
    pvm_config(&ilhost, &ilarch, &info);
    printf("%d", ilhost);
    for (i=0; i < (ilhost > MATRIX_SIZE ? MATRIX_SIZE : ilhost); i++)
    {
        pvm_spawn("/home/pvm3/pvm3/sekcja4/hello_other", 0, PvmTaskHost, info[
            i].hi_name, 1, &t_id);
        pvm_initsend(PvmDataDefault);
        pvm_pkint(&a[count][0], MATRIX_SIZE, 1);
        pvm_pkint(&b[count][0], MATRIX_SIZE, 1);
        pvm_pkint(&count, 1, 1);
        pvm_send(t_id, 100);
    }
}
```

```

        ++count;
    }
    while (count < MATRIX_SIZE)
    {
        bufid = pvm_recv(-1, 200);
        pvm_buinfo(bufid, &bytes, &msgtag, &t_id);
        pvm_upkint(&rescount, 1, 1);
        pvm_upkint(&r[rescount][0], MATRIX_SIZE, 1);
        pvm_initSend(PvmDataDefault);
        pvm_pkint(&a[count][0], MATRIX_SIZE, 1);
        pvm_pkint(&b[count][0], MATRIX_SIZE, 1);
        pvm_pkint(&count, 1, 1);
        pvm_send(t_id, 100);
        ++count;
    }
    for (i = 0; i < (ilhost > MATRIX_SIZE ? MATRIX_SIZE : ilhost); i++)
    {
        bufid = pvm_recv(-1, 200);
        pvm_buinfo(bufid, &bytes, &msgtag, &t_id);
        pvm_upkint(&rescount, 1, 1);
        pvm_upkint(&r[rescount][0], MATRIX_SIZE, 1);
        pvm_kill(t_id);
    }
    fprintf(txt, "Macierz _wynikowa:\n
    _____\n\n");
    for (i = 0; i < MATRIX_SIZE; i++)
    {
        for (j = 0; j < MATRIX_SIZE; j++)
            fprintf(txt, "%d\t", r[i][j]);
        fprintf(txt, "\n");
    }
    fclose(txt);
    exit(0);
}

```

Program potomny

```
#include <stdio.h>
#include "pvm3.h"
#define MATRIX_SIZE 20
int main()
{
    int masterid, count, i;
    double vecta[MATRIX_SIZE], vectb[MATRIX_SIZE], vectr[MATRIX_SIZE];
    masterid = pvm_parent();
    if(masterid == 0) exit(1); //zabezpieczenie przed uruchomieniem z
    //poziomu rodzica
    //OBSŁUGA OBLICZEŃ i WYSYŁANIA WYNIKÓW
    while(1)
    {
        pvm_recv(masterid, 100);
        pvm_upkdouble(&vecta[0], MATRIX_SIZE, 1);
        pvm_upkdouble(&vectb[0], MATRIX_SIZE, 1);
        pvm_upkint(&count, 1, 1);
        for(i = 0; i < MATRIX_SIZE; i++)
            vectr[i] = vecta[i] - vectb[i];
        pvm_initsend(PvmDataDefault);
        pvm_pkint(&count, 1, 1);
        pvm_pkdouble(&vectr[0], MATRIX_SIZE, 1);
        pvm_send(masterid, 200);
    }
    return(0);
}
```

2.3 Treść

Tabela A zawiera 10000 łańcuchów znakowych i tabela B zawiera 10000 wartości typu int, sum kontrolnych tabeli A. Kod rodzica porównującego checksumę z łańcuchem tabeli A.

```
#include "pvm3.h"
#include <stdio.h>
const int RECORD_NUMBER = 10000
void fill_tables(char* input[RECORD_NUMBER], int output[RECORD_NUMBER]);

int main_pvm()
{
    // dane wejściowe
    char* input[RECORD_NUMBER];
    int output[RECORD_NUMBER];
    // zakładamy, że ta funkcja wypełnia tablice jak należy
    fill_tables(input, output);
    // liczniki wystąpień
    unsigned int true_count = 0;
    // pobranie informacji
    int ilhost, ilarch;
    struct pvmhostinfo * info;
    pvm_config(&ilhost, &ilarch, &info);
    printf("Liczba hostów: %d\n", ilhost);

    int id, tid;
    id = 0;
    int i;
    // dla każdego hosta
    for (i = 0; i < ilhost; i++, id++)
    {
        pvm_spawn("/egzamin/dziecko", 0, PvmTaskHost, info[i].hi_name, 1, &tid);
        if (tid < 0)
        {
            ilhost--;
            continue;
        }
        pvm_initsend(PvmDataDefault);
        // wysyłamy:
        pvm_pkint(&id, 1, 1);           // id wiersza
        pvm_pkstr(input[id]);           // łańcuch
        pvm_pkint(&output[id], 1, 1);   // suma kontrolna
        pvm_send(tid, 100);
    }
}
```

```

{
    // ...
    int bufid, child_tid, child_id, tmp, result;
    while (id < RECORD_NUMBER)
    {
        bufid = pvm_recv(-1, 200);
        pvm_bufinfo(bufid, &tmp, &tmp, &child_tid);
        // pobranie danych
        pvm_upkint(&child_id, 1, 1);    // id wiersza
        pvm_upkint(&result, 1, 1);      // wynik
        // dziecko zwraca 0, jeśli suma nie była poprawna
        // lub 1, jeśli była poprawna
        true_count = true_count + result;
        // wysłanie nowych danych
        pvm_initsend(PvmDataDefault);
        pvm_pkint(&id, 1, 1);          // id kolejnego wiersza
        pvm_pkstr(input[id]);          // łańcuch
        pvm_pkint(&output[id], 1, 1);  // suma kontrolna
        pvm_send(child_tid, 100);
        id++;
    }
    // odbieranie ostatnich wyników od potomków
    for (i = 0; i < ilhost; i++)
    {
        bufid = pvm_recv(-1, 200);
        pvm_bufinfo(bufid, &tmp, &tmp, &child_tid);
        pvm_upkint(&child_id, 1, 1);    // pobranie id wiersza
        pvm_upkint(&result, 1, 1);      // wynik
        true_count = true_count + result;
    }
    // wypisanie wyniku
    printf("Liczba poprawnych sum kontrolnych = %d\n", true_count);
    printf("Liczba niepoprawnych sum kontrolnych = %d", RECORD_NUMBER -
        true_count);
    pvm_exit();

    return 0;
}

```

2.4 Treść

Wyliczenie silni każdego elementu tablicy T mającej 1000 elementów *INT*.

```
#include "pvm3.h"
#define ELEMENTNUMBER 1000
int main()
{
    // nasza tablica
    int T[ELEMENTNUMBER];
    int i;
    for (i = 0; i < ELEMENTNUMBER; i++)
        T[i] = i;

    int ilhost, ilarch;          //parametry z PVMA
    struct pvmhostinfo * info;
    pvm_config(&ilhost, &ilarch, &info);

    int id_komorki = 0;
    int tid;
    // Dla każdego hosta - inicjujemy go
    for (i = 0; i < ilhost; i++)
    {
        pvm_spawn("/potomek", 0, PvmTaskHost, info[i].hi_name, 1, &tid);
        //powołanie potomka
        if (tid < 0)
        {
            ilhost--;
            continue;
        }
        pvm_initsend(PvmDataDefault);
        // wysyłamy
        pvm_pkint(&T[id_komorki], 1, 1);
        pvm_send(tid, 100);
        id_komorki++;
    }
}
```

```

{
    // Wykonywanie programu aż do przedostatniej pętli
    int bufid , child_tid , child_id_komorki , child_silnia , tmp;
    while (id_komorki < ELEMENTNUMBER)
    {
        // odebranie info o zakończeniu pracy
        bufid = pvm_recv(-1, 200);
        pvm_buinfo(bufid , &tmp, &tmp, &child_tid);
        // pobranie obliczonej silni
        pvm_upkint(&child_id_komorki , 1, 1);
        pvm_upkint(&child_silnia , 1, 1);
        // napisanie wartości w tablicy
        T[child_id_komorki] = child_silnia;
        // wysłanie nowych danych
        pvm_initsend(PvmDataDefault);
        pvm_pkint(&T[id_komorki] , 1, 1);
        pvm_send(child_tid , 100);
        id_komorki++;
    }
    // Odebranie ostatnich danych
    for (i = 0; i < ilhost; i++)
    {
        bufid = pvm_recv(-1, 200);
        pvm_buinfo(bufid , &tmp, &tmp, &child_tid);
        // pobranie obliczonej silni
        pvm_upkint(&child_id_komorki , 1, 1);
        pvm_upkint(&child_silnia , 1, 1);
        // napisanie wartości w tablicy
        T[child_id_komorki] = child_silnia;
    }
    pvm_exit();
    return 0;
}

```


2.5 Treść

Program przeszukuje fragment obrazu (*wzorzec*) o rozmiarze $n \times n$ w obrazie *Obraz* o rozmiarze $m \times m$ ($m \gg n$). Funkcja *int find_pattern(x, y, n)*, gdzie *x, y* to współrzędne w obrazie, a *n* rozmiar wzorca szuka danego wzorca.

```
#include "pvm3.h"
extern int m;           // wymiar obraz
extern int n;           // wymiar wzorca

int main()
{
    int liczbaWystapien = 0;
    // tablica pikseli, zakładamy że tak reprezentowany jest obraz
    int obraz[m][m];
    int ilhost, ilarch;   //parametry z PVMA
    struct pvmhostinfo * info;
    pvm_config(&ilhost, &ilarch, &info);
    printf("Liczba hostow: %d\n", ilhost);

    int id_wiersza = 0;
    int tid;
    // Dla każdego hosta - inicjujemy go
    for ( i = 0; i < ilhost; i++ )
    {
        //powołanie potomka
        pvm_spawn( "/potomek", 0, PvmTaskHost, info[i].hi_name, 1, &tid);
        //jeśli nie udało się powołać potomka, zmniejszamy liczbę hostów i
        //kontynuujemy
        if ( tid < 0 )
        {
            ilhost--;
            continue;
        }
        pvm_initsend(PvmDataDefault);
        // wysyłamy:
        // elementy obrazu
        for (int i = 0; i < n; i++)
        {
            //pakowanie całego wiersza
            pvm_pkint(&obraz[id_wiersza + i][0], m, 1);
        }
        pvm_send(tid, 100);   //wysłanie wiersza
        id_wiersza++;
    }
}
```

```

{
    // ...
    // Wykonywanie programu aż do przedostatniej pętli
    int bufid, child_tid, child_liczba_wystapien, tmp;
    while ( id_wiersza < (m - n) )
    {
        // odebranie info o zakończeniu pracy
        bufid = pvm_recv(-1, 200);
        pvm_bufinfo(bufid, &tmp, &tmp, &child_tid);
        // pobranie liczby wystapien
        pvm_upkint(&child_liczba_wystapien, 1, 1);
        // zwiększenie wystapien wzorca
        liczbaWystapien += child_liczba_wystapien;
        // wysłanie nowych danych
        pvm_initSend(PvmDataDefault);
        // nowa czesc obrazu
        for (int i = 0; i < n; i++)
        {
            pvm_pkint(&obraz[id_wiersza + i][0], m, 1);
        }
        pvm_send(child_tid, 100);
        id_wiersza++;
    }
    // Odebranie ostatnich danych
    for (i = 0; i < id_wiersza - ilhost + 1; i++)
    {
        bufid = pvm_recv(-1, 200);
        pvm_bufinfo(bufid, &tmp, &tmp, &child_tid);
        // pobranie liczby wystapien
        pvm_upkint(&child_liczba_wystapien, 1, 1);
        // zwiększenie wystapien wzorca
        liczbaWystapien += child_liczba_wystapien;
    }
    printf("Liczba_wystapien_wzorca: %d", liczbaWystapien);
    pvm_exit();
    return 0;
};

```

3 Java Spaces

3.1 Laborki

3.1.1 Treść

Napisać program zawierający jednego Nadzorcę oraz wielu Pracowników. Nadzorca przekazuje do Java-Space 2 równe tablice zawierające obiekty typu Integer, a następnie otrzymuje wynikową tablicę zawierającą sumy odpowiadających sobie komórek. Operację dodawania mają realizować Pracownicy.

3.1.2 Rozwiązanie

Zadanie obliczania sumy tabel dzielimy na dwie części: *Task* oraz *Result*. *Taski* są generowane przez *Nadzorcę* i przekazywane *Pracownikom*, ci zaś wykonują zadanie i tworzą obiekty klasy *Result*, a następnie przekazują je *Nadzorcę*. *Nadzorca* je odbiera, kompletuje i ew. coś z nimi robi.

Nadzorca przydziela tyle zadań, ile potrzebuje, z kolei *Pracownicy* działają w nieskończoność. Aby zakończyć ich pracę, *Nadzorca* musi wysłać zadania z tzw. zatrutą pigułką (ang. *Poisoned Pill*), czyli obiekt zadania z nietypowym parametrem, który sygnalizuje zakończenie pracy. Może to być np. *Boolean* o wartości *false*, *Integer* o wartości -1, itp. Składowymi klas implementujących interfejs *Entry* nie mogą być typu prostego (*int*, *double* itp.), muszą być opakowane (*Integer*, *Double* itp.). Najbezpieczniej dawać je wszędzie.

```
/**
 * @author Son Mati
 * @waifu Itsuka Kotori
 */
public class Task implements Entry {
    public Integer cellID; // ID komórki tabeli
    public Integer valueA; // wartość z tabeli A
    public Integer valueB; // wartość z tabeli B
    public Boolean isPill; // czy zadanie jest zatrutą pigułką

    // Domyślny konstruktor, musi się znajdować
    public Task() {
        this.cellID = this.valueA = this.valueB = null;
        this.isPill = false;
    }

    public Task(Integer entryID, Integer valueA, Integer valueB, Boolean
        isPill) {
        this.cellID = entryID;
        this.valueA = valueA;
        this.valueB = valueB;
        this.isPill = isPill;
    }
}
```

```

/**
 * @author Son Mati
 * @waifu Itsuka Kotori
 */
public class Result implements Entry {
    public Integer cellID , value;
    public Result() {
        this.cellID = this.value = null;
    }
    public Result(final Integer EntryID, final Integer Value) {
        this.cellID = EntryID;
        this.value = Value;
    }
}

```

```

public class Client {
    protected Integer defaultLease = 100000;
    protected JavaSpace space;
    protected Lookup lookup;
    public Client() {
        lookup = new Lookup(JavaSpace.class);
    }
}

```

```

/**
 * @author Son Mati
 * @waifu Itsuka Kotori
 */
public class Worker extends Client {
    public Worker() {
    }
    public void startWorking() {
        while(true) {
            try {
                this.space = (JavaSpace)lookup.getService();
                Task task = new Task();
                task = (Task) space.take(task, null, defaultLease);
                if (task.isPill == true)
                {
                    space.write(task, null, defaultLease);
                    System.out.println("Koniec_pracy_workera.");
                    return;
                }
                Integer res = task.valueA + task.valueB;
                Result result = new Result(task.cellID, res);
                space.write(result, null, defaultLease);
            }
            catch (Exception ex) {}
        }
    }
    public static void main(String[] args) {
        Worker w = new Worker(); // utworzenie obiektu
        w.startWorking(); // realizacja zadania
    }
}

```

```

/**
 * @author Son Mati
 * @waifu Itsuka Kotori
 */
public class Supervisor extends Client {
    static final Integer INT_NUMBER = 125;
    public Integer [] TableA = new Integer [INT_NUMBER];
    public Integer [] TableB = new Integer [INT_NUMBER];
    public Integer [] TableC = new Integer [INT_NUMBER];
    // konstruktor
    public Supervisor () {
    }
    // wygenerowanie zawartości tablic
    public void generateData () {
        Random rand = new Random();
        for (int i = 0; i < INT_NUMBER; ++i) {
            TableA[i] = rand.nextInt(INT_NUMBER);
            TableB[i] = rand.nextInt(INT_NUMBER);
            TableC[i] = 0;
        }
    }
    // rozpoczęcie pracy
    public void startProducing () {
        try {
            this.space = (JavaSpace)lookup.getService();
            // utworzenie zadania
            for (Integer i = 0; i < INT_NUMBER; ++i) {
                Task task = new Task(i, this.TableA[i], this.TableB[i],
                    false);
                space.write(task, null, defaultLease);
            }
            // pobranie wyniku zadania
            System.out.println("Tablica C:");
            for (Integer i = 0 ; i < INT_NUMBER; ++i) {
                Result result = new Result();
                result = (Result) space.take(result, null, defaultLease);
                TableC[result.cellID] = result.value;
            }
            // utworzenie zatrutej pigulki na sam koniec
            Task poisonPill = new Task(null, null, null, true);
            space.write(poisonPill, null, defaultLease);
        }
        catch (Exception ex) {
        }
    }

    public static void main(String [] args) {
        // utworzenie obiektu
        Supervisor sv = new Supervisor();
        // utworzenie zadan
        sv.generateData();
        sv.startProducing();
    }
}

```

3.2 Treść

Program umieszczający w przestrzeni JavaSpace **200** obiektów zadań zawierających **dwa** pola typu całkowitego oraz **dwa** pola typu łańcuch znakowy (zawartość nieistotna, różna od NULL), podać deklarację klasy zadań. Następnie odebrać z przestrzeni kolejno **100** obiektów klasy *Odpowiedź* o atrybutach *id* typu *Integer* oraz *wynik* typu *Integer* posiadające w atrybucie *id* wartość **35**, a następnie wszystkie z atrybutem *id* = **10**. Przyjąć, że klasa *Odpowiedź* jest już zdefiniowana zgodnie z powyższym opisem.

3.2.1 Rozwiązanie

Klasa Zadanie

```
// deklaracja klasy, muszą być widoczne:
// implementacja interfejsu Entry
public class Zadanie implements Entry {
    // publiczne składowe, opakowujące typy zmiennych
    public Integer liczba;
    public String napis1;
    public String napis2;
    public Boolean poisonPill;
    // konstruktor domyślny, obowiązkowy
    public Zadanie() {
        Random rand = new Random();
        this.liczba = rand.nextInt();
        this.napis1 = Integer.toString(rand.nextInt());
        this.napis2 = Integer.toString(rand.nextInt());
        this.poisonPill = false;
    }
    public Zadanie(Integer liczba, String napis1, String napis2, Boolean
        poisonPill) {
        this.liczba = liczba;
        this.napis1 = napis1;
        this.napis2 = napis2;
        this.poisonPill = poisonPill;
    }
}
```

Nadrzędna klasa Klienta

```
/**
 * @author Son Mati & Doxus
 */
public class Klient {
    protected Integer defaultLease = 100000;
    protected JavaSpace space;
    protected Lookup lookup;
    public Klient() {
        lookup = new Lookup(JavaSpace.class);
    }
}
```

Klasa Nadzorcy

```
/**
 * @author Son Mati & Dexus
 */
public class Boss extends Client {
    // domyślne wartości dla zadania
    static final int DEFAULT_TASK_NUMBER = 200;
    static final int DEFAULT_MAX_MISSES = 100;

    Integer taskNumber;
    Integer maxMisses;

    public Integer getTaskNumber() {
        return taskNumber;
    }
    public Integer getMaxMisses() {
        return maxMisses;
    }
    // obowiązkowy domyślny konstruktor
    public Boss() {
        taskNumber = DEFAULT_TASK_NUMBER;
        maxMisses = DEFAULT_MAX_MISSES;
    }
    public Boss(Integer taskNumber, Integer maxMisses) {
        this.taskNumber = taskNumber;
        this.maxMisses = maxMisses;
    }
    /**
     * Wygenerowanie zadania z losowymi wartościami
     * @param id identyfikator
     * @param poisonPill pigułka, tak czy nie
     */
    public Zadanie generateTask(int id, boolean poisonPill) {
        Random rand = new Random();
        return new Zadanie(id, Integer.toString(rand.nextInt(1000)),
            Integer.toString(rand.nextInt(1000)), poisonPill);
    }
    /**
     * Utworzenie zadań
     * @param count ilość zadań
     */
    public void createTasksInJavaSpace(int count) {
        try {
            this.space = (JavaSpace)lookup.getService();
            for (int i = 0; i < count; ++i) {
                Zadanie zad = this.generateTask(i, false);
                space.write(zad, null, defaultLease);
                System.out.println("Wygenerowałem zad_ " + i + "_o_stringach_ " + zad.napis1 + "_i_" + zad.napis2);
            }
        }
        catch (RemoteException | TransactionException ex) {
            System.out.println("Dupa_XD");
        }
    }
}
```

```

public class Boss extends Client {
    /**
     * Uzyskanie odpowiedzi
     * @param id odpowiedzi, która nas interesuje
     * @param costam interesujący nas wynik
     * @param count liczba odpowiedzi do odbioru
     */
    public Integer receiveData(Integer id, Integer costam, Integer count) {
        Integer found = 0;
        try {
            this.space = (JavaSpace)lookup.getService();
            Odpowiedz wzor = new Odpowiedz(id, costam);
            for (int i = 0; i < count; i++) {
                // odczyt blokujący, zatrzymuje przepływ dopóki odp się nie pojawi
                Odpowiedz wynik = (Odpowiedz) space.takeIfExists(wzor, null, defaultLease);
                if (wynik != null) {
                    System.out.println("Odpowiedz: „id „ + wynik.getId() +
                        „ „wynik „ + wynik.getWynik());
                    found++;
                }
            }
        } catch (UnusableEntryException | TransactionException |
            InterruptedException | RemoteException ex) {
        }
        return found;
    }
    /**
     * ZATRUIJ DZIECIACZKI XD
     */
    public void poisonKids() {
        // utworzenie zatrutej pigulki
        Zadanie poisonPill = new Zadanie(null, null, null, true);
        try {
            space.write(poisonPill, null, defaultLease);
        } catch (TransactionException | RemoteException ex) {
        }
    }

    public static void main(String[] args) {
        Integer misses = 0;
        Boss boss = new Boss();
        boss.createTasksInJavaSpace(boss.getTaskNumber());
        boss.receiveData(35, null, 100);
        // boss odbiera pozostałe odpowiedzi, o id 10, dopóki nie trafi na
        // pewną liczbę chybień
        while(misses < boss.getMaxMisses()) {
            if (boss.receiveData(10, null, 1) == 0)
                misses++;
        }
        boss.poisonKids();
    }
}

```


Klasa Pracownika

```
/**
 * @author Son Mati & Dorus
 */
public class Sidekick extends Client {
    // obowiązkowy domyślny konstruktor
    public Sidekick() {
    }
    // praca
    public void zacznijMurzynic() {
        while(true) {
            try {
                Random rand = new Random();
                this.space = (JavaSpace)lookup.getService();
                Zadanie zad = new Zadanie(null, null, null, null);
                zad = (Zadanie) space.takeIfExists(zad, null, defaultLease)
                ;
                if (zad != null) {
                    if (zad.poisonPill == true) {
                        space.write(zad, null, defaultLease);
                        return;
                    }
                    System.out.println("Odebrałem zadanie o id " + zad.
                        liczba
                        + " i napisach " + zad.napis1 + " i " + zad.napis2);
                }
                Odpowiedz odp = new Odpowiedz(rand.nextInt(51), rand.
                    nextInt(1000));
                space.write(odp, null, defaultLease);
            } catch (TransactionException | RemoteException |
                UnusableEntryException | InterruptedException ex) {
                Logger.getLogger(Sidekick.class.getName()).log(Level.SEVERE
                    , null, ex);
            }
        }
    }
    // obowiązkowy Run
    public static void main(String[] args) {
        Sidekick murzyn = new Sidekick();
        murzyn.zacznijMurzynic();
    }
}
```

3.3 Treść

Kod programu głównego wykonawczego do przetwarzania z wykorzystaniem maszyny JavaSpace przetwarzającego obiekty zadań zawierające dwie wartości całkowite, oraz numer obiektu i flagę logiczną początkowo zawierającą wartość *FALSE*. W momencie pobrania obiektu zadania program wykonawczy ma podmienić w przestrzeni JavaSpace pobrany obiekt na ten sam, ale z flagą ustawioną na wartość *TRUE*. Przetwarzanie obiektu realizowane jest w funkcji *int check(int, int)* do której należy przekazać wartości z obiektu zadania. Po skończeniu przetwarzania zadania, przed zwróceniem wyniku, należy usunąć z przestrzeni JavaSpace obiekt przetwarzanego zadania. Wynik funkcji *check* należy umieścić w obiekcie wynikowym którego strukturę proszę zaproponować. Obsłużyć koniec działania programu przez skonsumowanie "zatrutej pigułki".

3.3.1 Rozwiązanie

Klasa Odpowiedzi (wynik Zadania):

```
/**
 * @author Son Mati
 * @waifu Itsuka Kotori
 */
public class Odpowiedz implements Entry {
    public Integer id;
    public Integer wynik;

    public Odpowiedz() {
    }

    public Odpowiedz(Integer id, Integer wynik) {
        this.id = id;
        this.wynik = wynik;
    }
}
```

Klasa Wykonawcy:

```
/**
 * @author Son Mati
 * @waifu Itsuka Kotori
 */
public class Sidekick {
    protected Integer defaultLease = 100000;
    protected JavaSpace space;
    protected Lookup lookup;
    // domyślny konstruktor obowiązkowy
    public Sidekick() {
        lookup = new Lookup(JavaSpace.class);
    }
    public void zacznijMurzynic() {
        while(true) {
            try {
                this.space = (JavaSpace)lookup.getService();
                Zadanie zad = new Zadanie(null, null, null, false, null);
                zad = (Zadanie) space.take(zad, null, defaultLease);
                if (zad.poisonPill == true) {
                    space.write(zad, null, defaultLease);
                    return;
                }
                zad.flag = true;
                space.write(zad, null, defaultLease);
                int result = this.check(zad.liczba1, zad.liczba2);
                Odpowiedz odp = new Odpowiedz(zad.id, result);
                space.take(zad, null, defaultLease); // pełen wzorzec
                space.write(odp, null, defaultLease);
            } catch (TransactionException | RemoteException |
                UnusableEntryException | InterruptedException ex) {
                Logger.getLogger(Sidekick.class.getName()).log(Level.SEVERE
                    , null, ex);
            }
        }
    }
    // dla przykładu
    public int check(int a, int b) {
        return a + b;
    }
    // obowiązkowy Run
    public static void main(String[] args) {
        Sidekick murzyn = new Sidekick();
        murzyn.zacznijMurzynic();
    }
}
```

3.4 Treść

Program umieszczający w przestrzeni JavaSpace **1000** obiektów zadań zawierających **dwa** pola typu całkowitego oraz **dwa** pola typu łańcuch znakowy (zawartość nieistotna, różna od NULL), podać deklarację klasy zadań. Następnie odebrać z przestrzeni **20** obiektów klasy *Odpowiedź* o atrybutach *id* typu *Integer* oraz *wynik* typu *Integer* posiadające w atrybucie *id* wartość **50** (przyjąć, że klasa *Odpowiedź* jest już zdefiniowana zgodnie z powyższym opisem).

3.4.1 Rozwiązanie

Działające i przetestowane w warunkach domowych na Jini.

Klasa Zadanie

```
/**
 * @author Son Mati & Doxus
 */
// deklaracja klasy, muszą być widoczne:
// implementacja interfejsu Entry
public class Zadanie implements Entry {
    // publiczne składowe, muszą być wielkich typów opakowujących
    public Integer liczba;
    public String napis1;
    public String napis2;
    public Boolean poisonPill;
    // konstruktor domyślny, wymagany
    public Zadanie() {
        Random rand = new Random();
        this.liczba = rand.nextInt();
        this.napis1 = Integer.toString(rand.nextInt());
        this.napis2 = Integer.toString(rand.nextInt());
        this.poisonPill = false;
    }
    // konstruktor z parametrami
    public Zadanie(Integer liczba, String napis1, String napis2, Boolean
        poisonPill) {
        this.liczba = liczba;
        this.napis1 = napis1;
        this.napis2 = napis2;
        this.poisonPill = poisonPill;
    }
}
```

Klasa nadzorcy

```
/**
 * @author Son Mati & Dorus
 */
public class Boss extends Client {
    // liczba zadań do wykonania
    static final int TASKNUMBER = 1000;
    // obowiązkowy domyślny konstruktor
    public Boss() {
    }
    /**
     * Wygenerowanie zadania z losowymi wartościami
     * @param count ilość zadań
     */
    public Zadanie generateTask(int id, boolean poisonPill) {
        Random rand = new Random();
        return new Zadanie(id, Integer.toString(rand.nextInt(1000)),
            Integer.toString(rand.nextInt(1000)), poisonPill);
    }
    /**
     * Utworzenie zadań
     * @param count ilość zadań
     */
    public void createTasksInJavaSpace(int count) {
        try {
            this.space = (JavaSpace)lookup.getService();
            for (int i = 0; i < count; ++i) {
                Zadanie zad = this.generateTask(i, false);
                space.write(zad, null, defaultLease);
                System.out.println("Wtgenerowałem zad_ " + i + "_o_stringach_ " + zad.napis1 + "_i_" + zad.napis2);
            }
        }
        catch (RemoteException | TransactionException ex) {
            System.out.println("Dupa XD");
        }
    }
}
```

```

public class Boss extends Client {
    /**
     * Uzyskanie odpowiedzi
     * @param id odpowiedzi, ktora nas interesuje
     * @param costam interesujacy nas wynik
     * @param count liczba odpowiedzi do odbioru
     */
    public void receiveData(Integer id, Integer costam, Integer count) {
        try {
            this.space = (JavaSpace)lookup.getService();
            Odpowiedz wzor = new Odpowiedz(id, costam);
            for (int i = 0; i < count; i++) {
                // odczyt blokujacy, zatrzymuje przeplyw dopóki odp się nie
                // pojawi
                Odpowiedz wynik = (Odpowiedz) space.take(wzor, null,
                    defaultLease);
                System.out.println("Odpowiedz: id=" + wynik.getId() + ",
                    wynik=" + wynik.getWynik());
            }
        } catch (UnusableEntryException | TransactionException |
            InterruptedException | RemoteException ex) {
            Logger.getLogger(Boss.class.getName()).log(Level.SEVERE, null,
                ex);
        }
    }
    /**
     * ZATRUIJ DZIECIACZKI XD
     */
    public void poisonKids() {
        // utworzenie zatrutej pigulki
        Zadanie poisonPill = new Zadanie(null, null, null, true);
        try {
            space.write(poisonPill, null, defaultLease);
        } catch (TransactionException | RemoteException ex) {
            Logger.getLogger(Boss.class.getName()).log(Level.SEVERE, null,
                ex);
        }
    }
    /**
     * Obowiazkowy Run dla nadzorcy
     */
    public static void main(String[] args) {
        Boss boss = new Boss();
        boss.createTasksInJavaSpace(TASK_NUMBER);
        boss.receiveData(50, null, 20);
        boss.poisonKids();
    }
}

```

Klasa Pracownika

```
/**
 * @author Son Mati & Dorus
 */
public class Sidekick extends Client {
    // obowiązkowy domyślny konstruktor
    public Sidekick() {
    }
    // rozpoczęcie pracy
    public void zacznijMurzynic() {
        while(true) {
            try {
                Random rand = new Random();
                this.space = (JavaSpace)lookup.getService();
                Zadanie zad = new Zadanie(null, null, null, null);
                zad = (Zadanie) space.takeIfExists(zad, null, defaultLease)
                    ;
                if (zad != null) {
                    if (zad.poisonPill == true) {
                        space.write(zad, null, defaultLease);
                        return;
                    }
                    System.out.println("Odebrałem_zadanie_o_id_" + zad.
                        liczba
                        + "_i_napisach_" + zad.napis1 + "_i_" + zad.napis2);
                }
                Odpowiedz odp = new Odpowiedz(rand.nextInt(51), rand.
                    nextInt(1000));
                space.write(odp, null, defaultLease);
            } catch (TransactionException | RemoteException |
                UnusableEntryException | InterruptedException ex) {
                Logger.getLogger(Sidekick.class.getName()).log(Level.SEVERE
                    , null, ex);
            }
        }
    }
    // obowiązkowy punkt wejścia
    public static void main(String[] args) {
        Sidekick murzyn = new Sidekick();
        murzyn.zacznijMurzynic();
    }
}
```

4 CUDA

4.1 Wyszukiwanie minimum i maximum w macierzach A i B wg wzorów:

- $\text{minMatrix}[i, j] = \min(A[i, j], B[i, j])$
- $\text{maxMatrix}[i, j] = \max(A[i, j], B[i, j])$

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define MATRIX_SIZE 10

__global__ void transposeKernel(int * matrixA, int * matrixB, int *
    matrixMin, int * matrixMax)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < MATRIX_SIZE && j < MATRIX_SIZE)
    {
        int index = i * MATRIX_SIZE + j;
        matrixMin[index] = matrixA[index] > matrixB[index] ? matrixB[index]
            : matrixA[index];
        matrixMax[index] = matrixA[index] < matrixB[index] ? matrixB[index]
            : matrixA[index];
    }
}

cudaError_t calculateMinMaxWithCuda(int ** matrixA, int ** matrixB, int **
    matrixMin, int ** matrixMax)
{
    // zdefiniowanie macierzy, która jest w pamięci karty
    int * dev_a;
    int * dev_b;
    int * dev_max, dev_min;
    size_t size = MATRIX_SIZE * MATRIX_SIZE;
    // cudne dynksy
    cudaError_t cudaStatus;
    cudaStatus = cudaSetDevice(0);
    // alokacja pamięci po 100 elementów
    cudaStatus = cudaMalloc((void**)&dev_a, size * sizeof(int));
    cudaStatus = cudaMalloc((void**)&dev_b, size * sizeof(int));
    cudaStatus = cudaMalloc((void**)&dev_min, size * sizeof(int));
    cudaStatus = cudaMalloc((void**)&dev_max, size * sizeof(int));
```



```

{
    // Spłaszczamy macierze do jednego wymiaru
    // Kopiujemy po wierszu, 10 x 10
    for (int i = 0; i < MATRIX_SIZE; i++)
    {
        cudaMemcpy((dev_a + i * MATRIX_SIZE), matrixA[i], MATRIX_SIZE *
            sizeof(int), cudaMemcpyHostToDevice);
        cudaMemcpy((dev_b + i * MATRIX_SIZE), matrixB[i], MATRIX_SIZE *
            sizeof(int), cudaMemcpyHostToDevice);
    }
    // blok z wątkami 3x3
    dim3 block_size(3, 3);
    // liczba potrzebnych bloków, tak żeby size = liczba wątków
    int block_count = MATRIX_SIZE / block_size.x + (MATRIX_SIZE %
        block_size.x == 0 ? 0 : 1);

    minMaxKernel << <grid_size, block_size >> >(dev_a, dev_b, dev_min,
        dev_max);
    // przywrócenie dwóch wymiarów i przepisanie wyników
    for (int i = 0; i < MATRIX_SIZE; i++)
    {
        cudaMemcpy(matrixMin[i], (dev_min + i * MATRIX_SIZE), MATRIX_SIZE *
            sizeof(int), cudaMemcpyDeviceToHost);
        cudaMemcpy(matrixMax[i], (dev_max + i * MATRIX_SIZE), MATRIX_SIZE *
            sizeof(int), cudaMemcpyDeviceToHost);
    }
    return 1;
}

int main()
{
    int ** matrixA, ** matrixB;
    int ** matrixMin, ** matrixMax;
    matrixA = (int**) malloc(sizeof(int*) * MATRIX_SIZE);
    matrixB = (int**) malloc(sizeof(int*) * MATRIX_SIZE);
    matrixMin = (int**) malloc(sizeof(int*) * MATRIX_SIZE);
    matrixMax = (int**) malloc(sizeof(int*) * MATRIX_SIZE);
    for (int i = 0; i < MATRIX_SIZE; i++)
    {
        matrixA[i] = (int*) malloc(sizeof(int) * MATRIX_SIZE);
        matrixB[i] = (int*) malloc(sizeof(int) * MATRIX_SIZE);
        matrixMin[i] = (int*) malloc(sizeof(int) * MATRIX_SIZE);
        matrixMax[i] = (int*) malloc(sizeof(int) * MATRIX_SIZE);
    }
    for (int i = 0; i < MATRIX_SIZE; i++)
    {
        for (int j = 0; j < MATRIX_SIZE; j++)
        {
            matrixA[i][j] = rand() % 20;
            matrixB[i][j] = rand() % 20;
            matrixMin[i] = matrixMax[i] = 0;
        }
    }
    cudaError_t cudaStatus = calculateMinMaxWithCuda(matrixA, matrixB,
        matrixMin, matrixMax);
    return 0;
}

```

4.2 Transpozycja macierzy kwadratowej

$$(a_{out}[i, j] = a_{in}[j, i])$$

```
#include <stdio.h>
#include <stdlib.h>
const unsigned int MATRIX_SIZE = 5;

cudaError_t transposeWithCuda(int ** matrix, int ** matrixOut)
{
    int * dev_a, dev_b;
    size_t size = MATRIX_SIZE * MATRIX_SIZE;
    // blocek z wgtkami 2x2, zgodnie z treścią zadania
    dim3 block_size(2, 2);
    // liczba potrzebnych bloków, tak żeby size = liczba wgtków
    int block_count;
    block_count = MATRIX_SIZE / block_size.x + (MATRIX_SIZE % block_size.x
        == 0 ? 0 : 1);
    // rozmiar gridu w blokach
    // blocek jest naszą podstawową jednostką (ma 4 wgtki)
    // grid ma mieć tyle bloków ile potrzebujemy
    dim3 grid_size(block_count, block_count);
    // jakieś dynksy do cudy
    cudaError_t cudaStatus;
    cudaStatus = cudaSetDevice(0);
    //alokacja tablic w pamięci karty graficznej
    cudaStatus = cudaMalloc((void**) &dev_a, size * sizeof(int));
    cudaStatus = cudaMalloc((void**) &dev_b, size * sizeof(int));
    // Spłaszczamy macierze do jednego wymiaru
    // przekopiowanie macierza do dev_a
    for(int i = 0; i < MATRIX_SIZE; i++)
    {
        cudaStatus = cudaMemcpy((dev_a + i * MATRIX_SIZE), matrix[i],
            MATRIX_SIZE * sizeof(int), cudaMemcpyHostToDevice);
    }
    transposeKernel<<<grid_size, block_size>>>(dev_a, dev_b);
    // Konwersja z wektora 1D na macierz 2D
    // przekopiowanie dev_b do matrixOut
    for(int i = 0; i < MATRIX_SIZE; i++) {
        cudaStatus = cudaMemcpy(matrixOut[i], (dev_b + i * MATRIX_SIZE),
            MATRIX_SIZE * sizeof(int), cudaMemcpyDeviceToHost);
    }
    return 1;
}
```

```

__global__ void transposeKernel(int * a, int * out)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < MATRIX_SIZE && j < MATRIX_SIZE)
    {
        out[i * MATRIX_SIZE + j] = a[j * MATRIX_SIZE + i];
    }
}

int main()
{
    int ** matrix;
    int ** matrixOut;
    matrix = (int**) malloc (sizeof(int*) * MATRIX_SIZE);
    matrixOut = (int**) malloc (sizeof(int*) * MATRIX_SIZE);

    for (int i = 0; i < MATRIX_SIZE; i++)
    {
        matrix[i] = (int*) malloc (sizeof(int) * MATRIX_SIZE);
        matrixOut[i] = (int*) malloc (sizeof(int) * MATRIX_SIZE);
    }
    for (int i = 0; i < MATRIX_SIZE; i++)
    {
        for (int j = 0; j < MATRIX_SIZE; j++)
        {
            matrix[i][j] = rand() % 20;
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
    // Add vectors in parallel.
    cudaError_t cudaStatus = transposeWithCuda(matrix, matrixOut);

    return 0;
}

```

5 MOSIX

5.1 Treść

Funkcja wykorzystuje n węzłów klastra, liczy wartość $F(x, y)$. Nagłówek funkcji:

```
double fun(double x, double y, int k)
```

gdzie wartością zwracaną jest obliczona wartość funkcji; k - liczba elementów iloczynu; x, y - argumenty funkcji. Założyć, że k jest podzielne przez n bez reszty.

5.1.1 Rozwiązanie

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

double calculateProduct(double x, double y, double a) {
    return a * sqrt(x * y) / (2 * pow(x, 3) + 5 * y);
}

int main(int argc, char * argv[]) {
    int n = atoi(argv[1]); // liczba węzłów klastra
    int k = atoi(argv[2]); // liczba elementów iloczynu
    double x = atof(argv[3]); // pierwszy argument
    double y = atof(argv[4]); // drugi argument
    int final_product = 1; // iloczyn wynikowy
    int process_number = k / n;
    int response_stream[2]; // strumień dla danych
    double part_result = 1; // jebnięcie potoków
    pipe(response_stream);
    // utworzenie procesów potomnych
    int i, j;
    for (i = 0; i < process_number; i++)
    {
        if (fork() == 0)
        {
            for (j = i * process_number; j < (i + 1) * process_number; j++)
            {
                part_result = part_result * calculateProduct(x, y, j);
                write(response_stream[1], &part_result, sizeof(double));
            }
            exit(0);
        }
    }
    // odczytanie danych częściowych
    for (i = 0; i < process_number; i++)
    {
        read(response_stream[0], &part_result, sizeof(double));
        final_product = final_product * part_result;
    }
    printf("Twój _szczęśliwy _iloczyn _to _%g", final_product);
    return 0;
}
```

5.2 2014, 1 termin, Daniel Kostrzewa

5.2.1 Treść

Mają do dyspozycji n węzłów klastra napisać funkcję, która obliczy całkę metodą trapezów z funkcji zapisanej poniżej. Nagłówek funkcji wygląda następująco:

```
double calka(double x1, double xp, double krok)
```

gdzie wartością zwracaną jest obliczona wartość całki; x1 - początek przedziału całkowania, xp - koniec przedziału całkowania, krok - wartość kroku całkowania.

$$y = 15x^3 - 7x^2 + \sin(2x^4) - 8$$

5.2.2 Rozwiązanie

```
#include <stdio.h>
#include <stdlib.h>
#define N 4

double func(double x)
{
    return 15 * pow(x, 3) - 7*x*x + sin(2 * pow(x, 4)) - 8;
}
// x1: wspolrzedna x poczatku przedzialu calkowania
// xk: wspolrzedna x konca przedzialu calkowania
// krok: wysokosc trapezu (x2 - x1)
double liczCalke(double x1, double xk, double krok)
{
    double x2 = x1 + krok;           // Wspolrzedna x drugiej
    // podstawy trapezu
    double a, b;                     // Wartosci funkcji w x1, x2
    double wynik = 0;

    while (x2 < xk)
    {
        a = func(x1);
        b = func(x2);
        wynik += (a + b) * krok / 2;

        x1 += krok;
        x2 += krok;
    }
    return wynik;
}
double calka(double x1, double xp, double krok)
{
    int i, forkResult;
    // Wartosc calki oznaczonej w calym przedziale / przedziale dziecka
    double mainResult = 0, childResult = 0;
    // Szerokosc przedzialu, w jakim dziecko liczy calke
    double delta = (xp - x1) / N;
    // Potok do zapisu i odczytu
    int potok[2];
    pipe(potok);
```

```

{
    // Kindermachen
    for (i = 0; i < N; i++)
    {
        forkResult = fork();
        if (forkResult == 0)
        {
            childResult = liczCalke(xl, xl + delta, krok);
            write(potok[1], &childResult, sizeof(childResult));
            exit(0);
        }
        else if (forkResult < 0)
        {
            printf("Wystapil_blad_podczas_tworzenia_dziecka");
            return -1;
        }
        xl += delta;
    }
    // Odczytanie i zsumowanie wyników przez rodzica
    for (i = 0; i < N; i++)
    {
        read(potok[0], &childResult, sizeof(childResult));
        mainResult += childResult;
    }
    printf("\nCalka_obliczona_metoda_trapezow_jest_rowna: %f\n", mainResult);
    return mainResult;
}

```

5.3 2015, 0 termin, Daniel Kostrzewa

5.3.1 Treść

Program tworzy n procesów potomnych. Proces zarządzający ma wysyłać zestaw liczb do procesów potomnych (założyć, że liczba przesyłanych danych wynosi k). Procesy potomne mają w pętli wykonywać następujące czynności: czekać na liczbę wysłaną przez proces zarządzający, na podstawie odebranej liczby obliczyć pole koła (odebrana liczba jest promieniem koła), odesłać wynik do procesu zarządzającego. Proces zarządzający po wysłaniu wszystkich liczb przechodzi w stan odbierania danych i sumuje pola kół. Końcowa wartość ma zostać wyświetlona na ekranie.

5.3.2 Rozwiązanie

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159

float calculate_circle_area(int radius)
{
    return PI * pow((float) radius, 2);
}

int main(int argc, char* argv[])
{
    int n = atoi(argv[1]);
    int k = atoi(argv[2]);
    //promień -1 oznacza że jest to trujaca pigulka zabijajaca proces
    potomny
    int poison = -1;
    //allokacja tablicy k liczb (promieni)
    int * tab = (int*) malloc(sizeof(int) * k);
    int i;
    for (i = 0; i < k; i++)
    {
        tab[i] = rand() % 20;
    }
    int process_number = n / k;
    // 2 strumienie, jeden do wysyłania danych, drugi do odbioru odpowiedzi
    int data_stream[2], response_stream[2];
    // wyniki
    int part;
    int sum = 0;
    // jebnięcie potoków
    // odpowiedzi z procesów potomnych
    pipe(response_stream);
    // in - promienie kół
    // out - otrzymane wyniki - pola kół
    pipe(data_stream);
```

```

// utworzenie procesów potomnych
for (i = 0; i < n; i++)
{
    if (fork() == 0)
    {
        int radius;
        // wykonywanie obliczeń dopóki nie zostanie OTRUTY(!)
        while(true)
        {
            if (read(data_stream[0], &radius, sizeof(int)) != sizeof(int))
                continue;
            if (radius == -1) // wyłącza się tylko jak otrzymamy pigułkę
                exit(0);
            float result = calculate_circle_area(radius);
            write(response_stream[1], &result, sizeof(float));
        }
    }
}
// wysłanie danych do procesów potomnych
for (i = 0; i < k; i++)
{
    write(data_stream[1], &tab[i], sizeof(int));
}
// Halo odbjooor danych
for (i = 0; i < n; i++)
{
    read(response_stream[0], &part, sizeof(int));
    sum = sum + part;
}
// ZABIJANIE DZIECI
for (i = 0; i < k; i++)
{
    write(data_stream[1], &poison, sizeof(int));
}
// wypisanie odpowiedzi
printf("Suma_pól_kół: %d", sum);
return 0;

```


5.4 Treść

Program liczy wartość wielomianu k -tego stopnia w punkcie x wyrażonego wzorem. Liczby k oraz x są podawane jako parametry wywołania programu. W Zadaniu należy stworzyć potok oraz dwa procesy potomne. Jeden z procesów potomnych wyznacza wartość wielomianu dla $k-1$ stopnia, natomiast drugi dla $k-2$ stopnia. Zadanie rozwiązać metodą rekurencyjną.

$$P_k(x) = 15P_{k-1}\left(\frac{3}{8}x\right) - P_{k-2}\left(\frac{x^2}{2}\right) \quad P_0(x) = 1 \quad P_1(x) = x$$

5.4.1 Rozwiązanie

```
#include <stdio.h>
#include <stdlib.h>
// Main
int main(int argc, char *argv[])
{
    double wynik = 0; // Ostateczny wynik programu
    double x;
    int k;
    if (argc == 3)
    {
        k = atoi(argv[1]);
        x = atof(argv[2]);
        wynik = liczWielomian(k, x);
        printf("Wielomian jest rowny: %f\n", wynik);
    }
    else
        printf("Podano nieprawidlowe parametry!\n");
    return 0;
}
// Metoda do liczenia wartosci wielomianu
double licz2(int k, double x)
{
    double result, Pk1, Pk2;
    if (k == 0)
        return 1;
    if (k == 1)
        return x;
    Pk1 = licz2(k - 1, 0.375 * x);
    Pk2 = licz2(k - 2, x*x / 2.0);
    result = 15 * Pk1 - Pk2;
    return result;
}
```

```

// Obliczenie całości wielomianu
double liczWielomian(int k, double x)
{
    double mainResult = 0, childResult1, childResult2;
    int potok[2];
    if (k == 0)
        return 1;
    if (k == 1)
        return x;
    pipe(potok);
    // Pierwsze dziecko - wielomian k-1 stopnia
    if ( fork() == 0 )
    {
        childResult1 = 15 * licz2(k - 1, 0.375 * x);
        write( potok[1], &childResult1, sizeof(childResult1) );
        exit(0);
    }
    // Drugie dziecko - wielomian k-2 stopnia
    if ( fork() == 0 )
    {
        childResult2 = licz2(k - 2, x*x / 2.0);
        write( potok[1], &childResult2, sizeof(childResult2) );
        exit(0);
    }
    read( potok[0], &childResult1, sizeof(childResult1) );
    read( potok[0], &childResult2, sizeof(childResult2) );
    mainResult = childResult1 - childResult2;
    return mainResult;
}

```