

Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Реализация вычислительно сложного алгоритма с применением
программного пакета PVM

Отчет по лабораторной работе №5 дисциплины
«Параллельное программирование»

Выполнил студент группы ИВТ-31 _____/Крючков И. С/
Проверил _____/Долженкова М. Л./

Киров 2023

1. Цель лабораторной работы

Знакомство с программным пакетом PVM, получение навыков реализации параллельных приложений с его использованием.

2. Задание

- 1) Изучить основные принципы работы с пакетом PVM
- 2) Выделить в полученной в ходе первой лабораторной работы реализации алгоритма фрагменты кода, выполнение которых может быть разнесено на несколько процессоров.
- 3) Реализовать параллельную версию алгоритма с помощью языка C++, используя при этом предлагаемые PVM механизмы
- 4) Показать корректность полученной реализации путём осуществления тестирования на построенном в ходе первой лабораторной работы наборе тестов
- 5) Провести доказательную оценку эффективности PVM-реализации алгоритма.

3. Области распараллеливания алгоритма

Для вычисления результата Штрассен предложил алгоритм с семью умножениями:

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_2 = (A_{21} + A_{22})B_{11}$$

$$P_3 = A_{11}(B_{12} - B_{22})$$

$$P_4 = A_{22}(B_{21} - B_{11})$$

$$P_5 = (A_{11} + A_{12})B_{22}$$

$$P_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

Вычисление значения каждого P_i выполняется независимо, поэтому их вычисление можно ускорить за счет выполнения в несколько потоков.

Получение матрицы результата:

$$C_{11} = P_1 + P_4 - P_5 + P_7$$

$$C_{12} = P_3 + P_5$$

$$C_{21} = P_2 + P_4$$

$$C_{22} = P_1 + P_3 - P_2 + P_6$$

Вычисления каждой подматрицы результата можно разбить на независимые части и выполнять в отдельных потоках:

$$Q_1 = P_1 + P_4$$

$$Q_2 = P_2 + P_4$$

$$Q_3 = P_3 + P_6$$

$$Q_4 = P_7 - P_5$$

$$Q_5 = P_3 + P_5$$

$$Q_6 = P_1 - P_2$$

Данные преобразования сводятся к каскадной схеме вычислений, представленной на рисунке 1.

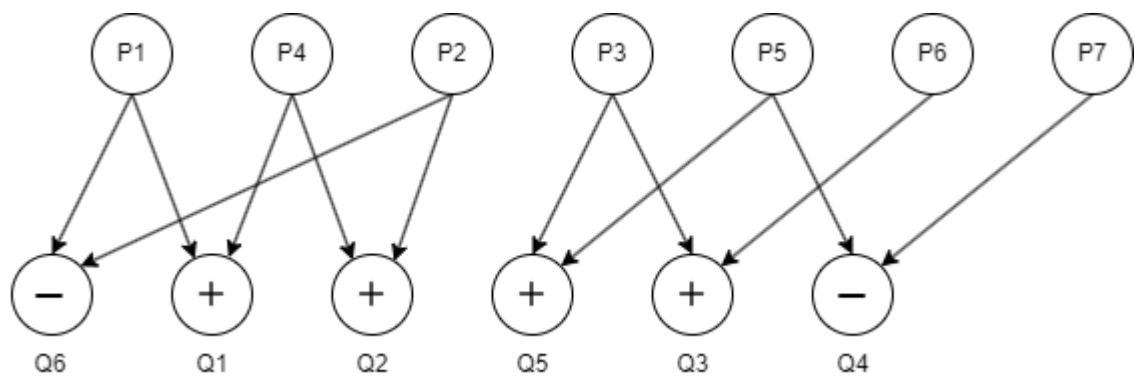


Рисунок 1 – Каскадная схема

4. Схема взаимодействия процессов

Графическая схема изображения на рисунке 2.

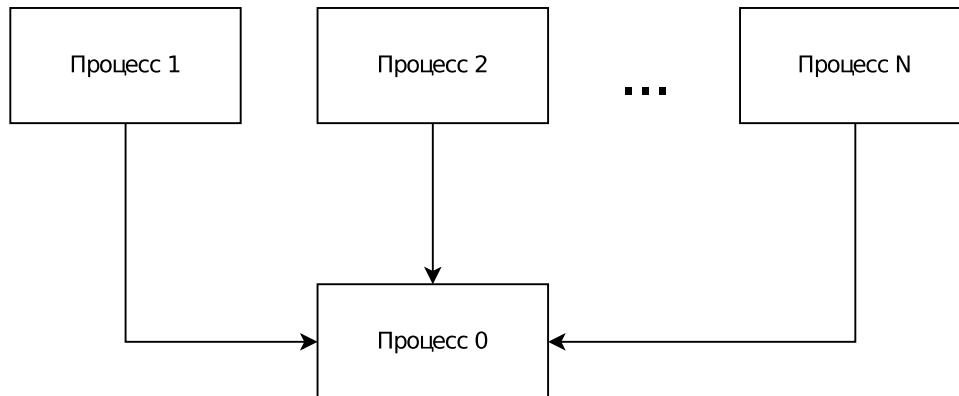


Рисунок 2 – Графическая схема

5. Программная реализация

Листинг программной реализации приведен в приложении А.

6. Тестирование

При тестировании выполнялось умножение квадратных матриц, сгенерированных случайным образом.

Тестирование выполнялось на 2х компьютерах, соединенных в локальную сеть.

Характеристики компьютеров приведены в таблице 1.

Таблица 1 – Характеристики компьютеров

	1 компьютер	2 компьютер
ОС	Linux Mint	Linux Mint
Процессор	Intel Core i5-8250U 1.6 GHz, 4 физических, 8 логических ядер	Intel Core i5-7200U 2.50GHz 2 физических, 4 логических ядра
ОЗУ	6 Гб	6 Гб

Результаты тестирования приведены в таблице 2.

Таблица 2 – Результаты тестирования.

№	Размер матриц	Линейный алгоритм, с	Параллельный алгоритм, с	OpenMP, с	MPI, с	PVM, с
1	2048	3.562	0.998	1.01	1.1	12.8
2	4096	24.81	6.745	6.7	7.4	85.0
3	8192	176.352	46.397	46.0	51.4	510.0

7. Вывод

В ходе выполнения лабораторной работы был изучен программный пакет PVM. Получены навыки построения локальных сетей, принципы создания многопроцессорных приложений, запускаемых в вычислительных сетях.

Был разработан параллельный алгоритм умножения матриц методом Штрассена с использованием PVM.

Реализованный с помощью PVM параллельный алгоритм оказался примерно в 10 раз медленнее всех параллельных алгоритмов, реализованных в предыдущих лабораторных работах и в 4 раза медленнее линейного алгоритма. Это связано с тем, что передача данных между процессами выполняется по локальной сети.

Приложение А.

Листинг программной реализации

master.cpp

```
#include "pvm3.h"
#include <iostream>
#include <fstream>
#include <cstring>
#include <chrono>

#define NUM_PROCS 7

int** newMatrix(int64_t n) {
    int* data = new int [n*n];
    int** arr = new int* [n];

    for (int64_t i = 0; i < n; ++i) {
        arr[i] = &(data[n*i]);
    }

    return arr;
}

void deleteMatrix(int** m) {
    delete[] m[0];
    delete[] m;
}

void read_matrix(std::ifstream &in, int** m, int64_t n, int64_t real_n) {
    for (int64_t i = 0; i < real_n; ++i) {
        memset(m[i], 0, n * sizeof *m[i]);
        for (int64_t j = 0; j < real_n; ++j) {
            in >> m[i][j];
        }
    }
}

int64_t new_size(int64_t n) {
    int64_t r = 1;
    while((n >>= 1) != 0) {
        r++;
    }
    return 1 << r;
}

bool isPowerOfTwo(int64_t v) {
    return v && !(v & (v - 1));
}

int main(int argc, char** argv) {

    if (argc < 3) {
        std::cout << "Run program with args: input.txt output.txt" << std::endl;
        return 1;
    }

    int n = 0;
    int real_n = 0;

    std::ifstream in;
    in.open(argv[1]);

    if (!in.is_open()) {
        std::cout << "matrix.txt open error";
        return 1;
    }
}
```

```

}

in >> real_n;
n = real_n;

if (!isPowerOfTwo(real_n) || real_n == 1) {
    n = new_size(real_n);
}

int** a = newMatrix(n);
int** b = newMatrix(n);

read_matrix(in, a, n, real_n);
read_matrix(in, b, n, real_n);

in.close();

int tIds[NUM_PROCS];

pvm_spawn("slave", (char**)0, PvmTaskDefault, "", NUM_PROCS, tIds);

// #0
for (int i = 0; i < NUM_PROCS; ++i) {
    pvm_initSend(PvmDataDefault);
    pvm_pkint(&i, 1, 1);
    pvm_send(tIds[i], 0);
}

std::chrono::steady_clock::time_point begin;
std::chrono::steady_clock::time_point end;
int64_t elapsed_ms;

int** result = newMatrix(n);
begin = std::chrono::steady_clock::now();

// #1 send n
pvm_initSend(PvmDataDefault);
pvm_pkint(&n, 1, 1);
pvm_mcast(tIds, NUM_PROCS, 1);

// #2 send a
pvm_initSend(PvmDataDefault);
pvm_pkint(&a[0][0], n*n, 1);
pvm_mcast(tIds, NUM_PROCS, 2);

// #3 send b
pvm_initSend(PvmDataDefault);
pvm_pkint(&b[0][0], n*n, 1);
pvm_mcast(tIds, NUM_PROCS, 3);

// #18 send tId
pvm_initSend(PvmDataDefault);
pvm_pkint(&tIds[0], NUM_PROCS, 1);
pvm_mcast(tIds, NUM_PROCS, 18);

pvm_recv(tIds[0], 17);
pvm_upkint(result[0], n*n, 1);

end = std::chrono::steady_clock::now();
elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count();

std::ofstream out(argv[2]);
if (!out.is_open()) {
    std::cout << "Result file open error";
    return 1;
}

```

```

    for (int64_t i = 0; i < real_n; ++i) {
        for (int64_t j = 0; j < real_n; ++j) {
            out << result[i][j] << " ";
        }
        out << std::endl;
    }

    deleteMatrix(a);
    deleteMatrix(b);
    deleteMatrix(result);

    out.close();

    std::cout << "Ok " << std::endl;
    std::cout << "Time (s): " << (double) elapsed_ms/1000 << std::endl;

    pvm_exit();

    return 0;
}

```

slave.cpp

```

#include "pvm3.h"
#include <iostream>
#include <cstring>
#include <string>
#include <fstream>

#define NUM_PROCS 7

int** newMatrix(int64_t n) {
    int* data = new int [n*n];
    int** arr = new int* [n];

    for (int64_t i = 0; i < n; ++i) {
        arr[i] = &(data[n*i]);
    }

    return arr;
}

void deleteMatrix(int** m) {
    delete[] m[0];
    delete[] m;
}

int** matrix_multiply(int** a, int** b, int n) {
    int** result = newMatrix(n);

    int i, j, k;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            result[i][j] = 0;
            for (k = 0; k < n; k++)
                result[i][j] += a[i][k] * b[k][j];
        }
    }

    return result;
}

int** addMatrix(int** a, int** b, int64_t n) {
    int** result = newMatrix(n);
    for (int64_t i = 0; i < n; ++i) {
        for (int64_t j = 0; j < n; ++j) {
            result[i][j] = a[i][j] + b[i][j];
        }
    }
}

```



```

    }

    return result;
}

int** subMatrix(int** a, int** b, int64_t n) {
    int** result = newMatrix(n);
    for (int64_t i = 0; i < n; ++i) {
        for (int64_t j = 0; j < n; ++j) {
            result[i][j] = a[i][j] - b[i][j];
        }
    }

    return result;
}

int** getSlice(int** m, int oi, int oj, int64_t n) {
    int** matrix = newMatrix(n);

    for (int64_t i = 0; i < n; ++i) {
        for (int64_t j = 0; j < n; ++j) {
            matrix[i][j] = m[i+oi][j+oj];
        }
    }

    return matrix;
}

int** combMatrix(int** c11, int** c12, int** c21, int** c22, int64_t n) {
    int64_t m = n*2;

    int** result = newMatrix(m);

    for (int64_t i = 0; i < m; ++i) {
        for (int64_t j = 0; j < m; ++j) {
            if (i < n && j < n) {
                result[i][j] = c11[i][j];
            } else if (i < n) {
                result[i][j] = c12[i][j-n];
            } else if (j < n) {
                result[i][j] = c21[i-n][j];
            } else {
                result[i][j] = c22[i-n][j-n];
            }
        }
    }

    return result;
}

int** strassen(int**a, int**b, int64_t n) {
    if (n <= 64) {
        return matrix_multiply(a, b, n);
    } else {

        n = n >> 1;

        int** a11 = getSlice(a, 0, 0, n);
        int** a12 = getSlice(a, 0, n, n);
        int** a21 = getSlice(a, n, 0, n);
        int** a22 = getSlice(a, n, n, n);
        int** b11 = getSlice(b, 0, 0, n);
        int** b12 = getSlice(b, 0, n, n);
        int** b21 = getSlice(b, n, 0, n);
        int** b22 = getSlice(b, n, n, n);

        int** t1;
        int** t2;
    }
}

```

```

// A11 + A22
t1 = addMatrix(a11, a22, n);
// B11 + B22
t2 = addMatrix(b11, b22, n);
// P1 = t1 * t2
int** p1 = strassen(t1, t2, n);
deleteMatrix(t1);
deleteMatrix(t2);

// A21 + A22
t1 = addMatrix(a21, a22, n);
// P2 = t1 * B11
int** p2 = strassen(t1, b11, n);
deleteMatrix(t1);

// B12 - B22
t1 = subMatrix(b12, b22, n);
// P3 = A11 * t1
int** p3 = strassen(a11, t1, n);
deleteMatrix(t1);

// B21 - B11
t1 = subMatrix(b21, b11, n);
// P4 = A22 * t1
int** p4 = strassen(a22, t1, n);
deleteMatrix(t1);

// A11 + A12
t1 = addMatrix(a11, a12, n);
// P5 = t1 * B22
int** p5 = strassen(t1, b22, n);
deleteMatrix(t1);

// A21 - A11
t1 = subMatrix(a21, a11, n);
deleteMatrix(a11);
deleteMatrix(a21);
// B11 + B12
t2 = addMatrix(b11, b12, n);
deleteMatrix(b11);
deleteMatrix(b12);
// P6 = t1 * t2
int** p6 = strassen(t1, t2, n);
deleteMatrix(t1);
deleteMatrix(t2);

// A12 - A22
t1 = subMatrix(a12, a22, n);
deleteMatrix(a12);
deleteMatrix(a22);
// B21 + B22
t2 = addMatrix(b21, b22, n);
deleteMatrix(b21);
deleteMatrix(b22);
// P7 = t1 * t2
int** p7 = strassen(t1, t2, n);
deleteMatrix(t1);
deleteMatrix(t2);

t1 = addMatrix(p1, p4, n);
t2 = subMatrix(p7, p5, n);
deleteMatrix(p7);

int** c11 = addMatrix(t1, t2, n);

```

```

deleteMatrix(t1);
deleteMatrix(t2);

int** c12 = addMatrix(p3, p5, n);
deleteMatrix(p5);
int** c21 = addMatrix(p2, p4, n);
deleteMatrix(p4);

t1 = addMatrix(p1, p3, n);
deleteMatrix(p1);
deleteMatrix(p3);

t2 = subMatrix(p6, p2, n);
deleteMatrix(p2);
deleteMatrix(p6);

int** c22 = addMatrix(t1, t2, n);
deleteMatrix(t1);
deleteMatrix(t2);

int** res = combMatrix(c11, c12, c21, c22, n);
deleteMatrix(c11);
deleteMatrix(c12);
deleteMatrix(c21);
deleteMatrix(c22);

return res;
}
}

void strassen_pvm(int** a, int** b, int64_t n, int**& result, int rank, int* tIds) {

    n = n >> 1;

    int** a11 = nullptr; // r0 r2 r4 r5
    int** a12 = nullptr; // r4 r6
    int** a21 = nullptr; // r1 r5
    int** a22 = nullptr; // r0 r1 r3 6
    int** b11 = nullptr; // r0 r1 r3 r5
    int** b12 = nullptr; // r2 r5
    int** b21 = nullptr; // r3 r6
    int** b22 = nullptr; // r0 r2 r4 r6

    int** p1 = nullptr; // r0, r5
    int** p2 = nullptr; // r1, r5
    int** p3 = nullptr; // r2, r4
    int** p4 = nullptr; // r0, r1
    int** p5 = nullptr; // r3, r4
    int** p6 = nullptr; // r2
    int** p7 = nullptr; // r3

    if (rank == 0) {
        a11 = getSlice(a, 0, 0, n);
        a22 = getSlice(a, n, n, n);
        b11 = getSlice(b, 0, 0, n);
        b22 = getSlice(b, n, n, n);

        p1 = newMatrix(n);
        p4 = newMatrix(n);

        // A11 + A22
        int** t1 = addMatrix(a11, a22, n);
        // B11 + B22
        int** t2 = addMatrix(b11, b22, n);
        // P1 = t1 * t2
        p1 = strassen(t1, t2, n);
    }
}

```

```

deleteMatrix(t1);
deleteMatrix(t2);
deleteMatrix(a11);
deleteMatrix(a22);
deleteMatrix(b11);
deleteMatrix(b22);

// + send p1, task 5, tag 4
pvm_initsend(PvmDataDefault);
pvm_pkint(&(p1[0][0]), n*n, 1);
pvm_send(tIds[5], 4);

// + recv p4, task 3, tag 5
pvm_recv(tIds[3], 5);
pvm_upkint(p4[0], n*n, 1);
}

if (rank == 1) {
    a21 = getSlice(a, n, 0, n);
    a22 = getSlice(a, n, n, n);
    b11 = getSlice(b, 0, 0, n);

    p2 = newMatrix(n);
    p4 = newMatrix(n);

    // A21 + A22
    int** t1 = addMatrix(a21, a22, n);
    // P2 = t1 * B11
    p2 = strassen(t1, b11, n);

    deleteMatrix(t1);
    deleteMatrix(a21);
    deleteMatrix(a22);
    deleteMatrix(b11);

    // + send p2, task 5, tag 6
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&(p2[0][0]), n*n, 1);
    pvm_send(tIds[5], 6);

    // + recv p4, task 3, tag 7
    pvm_recv(tIds[3], 7);
    pvm_upkint(p4[0], n*n, 1);
}

if (rank == 2) {
    a11 = getSlice(a, 0, 0, n);
    b12 = getSlice(b, 0, n, n);
    b22 = getSlice(b, n, n, n);

    p3 = newMatrix(n);
    p6 = newMatrix(n);

    // B12 - B22
    int** t1 = subMatrix(b12, b22, n);
    // P3 = A11 * t1
    p3 = strassen(a11, t1, n);

    deleteMatrix(t1);
    deleteMatrix(a11);
    deleteMatrix(b12);
    deleteMatrix(b22);

    // + send p3, task 4, tag 8
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&(p3[0][0]), n*n, 1);

```

```

    pvm_send(tIds[4], 8);

    // + recv p6, task 5, tag 9
    pvm_recv(tIds[5], 9);
    pvm_upkint(p6[0], n*n, 1);
}

if (rank == 3) {
    a22 = getSlice(a, n, n, n);
    b11 = getSlice(b, 0, 0, n);
    b21 = getSlice(b, n, 0, n);

    p5 = newMatrix(n);
    p7 = newMatrix(n);

    // B21 - B11
    int** t1 = subMatrix(b21, b11, n);
    // P4 = A22 * t1
    p4 = strassen(a22, t1, n);

    deleteMatrix(t1);
    deleteMatrix(a22);
    deleteMatrix(b11);
    deleteMatrix(b21);

    // + send p4, task 0, tag 5
    pvm_itsend(PvmDataDefault);
    pvm_pkint(&(p4[0][0]), n*n, 1);
    pvm_send(tIds[0], 5);

    // + recv p5, task 4, tag 10
    pvm_recv(tIds[4], 10);
    pvm_upkint(p5[0], n*n, 1);

    // + send p4, task 1, tag 7
    pvm_itsend(PvmDataDefault);
    pvm_pkint(&(p4[0][0]), n*n, 1);
    pvm_send(tIds[1], 7);

    // + recv p7, task 6, tag 11
    pvm_recv(tIds[6], 11);
    pvm_upkint(p7[0], n*n, 1);
}

if (rank == 4) {
    a12 = getSlice(a, 0, n, n);
    a11 = getSlice(a, 0, 0, n);
    b22 = getSlice(b, n, n, n);

    p3 = newMatrix(n);
    p5 = newMatrix(n);

    // A11 + A12
    int** t1 = addMatrix(a11, a12, n);
    // P5 = t1 * B22
    p5 = strassen(t1, b22, n);

    deleteMatrix(t1);
    deleteMatrix(a11);
    deleteMatrix(a12);
    deleteMatrix(b22);

    // + send p5, task 3, tag 10
    pvm_itsend(PvmDataDefault);
    pvm_pkint(&(p5[0][0]), n*n, 1);
    pvm_send(tIds[3], 10);

    // + recv p3, task 2, tag 8
    pvm_recv(tIds[2], 8);

```

```

    pvm_upkint(p3[0], n*n, 1);
}

if (rank == 5) {
    a11 = getSlice(a, 0, 0, n);
    a21 = getSlice(a, n, 0, n);
    b11 = getSlice(b, 0, 0, n);
    b12 = getSlice(b, 0, n, n);

    p1 = newMatrix(n);
    p2 = newMatrix(n);

    // A21 - A11
    int** t1 = subMatrix(a21, a11, n);
    // B11 + B12
    int** t2 = addMatrix(b11, b12, n);
    // P6 = t1 * t2
    p6 = strassen(t1, t2, n);

    deleteMatrix(t1);
    deleteMatrix(t2);
    deleteMatrix(a11);
    deleteMatrix(a21);
    deleteMatrix(b11);
    deleteMatrix(b12);

    // + send p6, task 2, tag 9
    pvm_itsend(PvmDataDefault);
    pvm_pkint(&(p6[0][0]), n*n, 1);
    pvm_send(tIds[2], 9);

    // + recv p1, task 0, tag 4
    pvm_recv(tIds[0], 4);
    pvm_upkint(p1[0], n*n, 1);

    // + recv p2, task 1, tag 6
    pvm_recv(tIds[1], 6);
    pvm_upkint(p2[0], n*n, 1);
}

if (rank == 6) {
    a12 = getSlice(a, 0, n, n);
    a22 = getSlice(a, n, n, n);
    b21 = getSlice(b, n, 0, n);
    b22 = getSlice(b, n, n, n);

    // A12 - A22
    int** t1 = subMatrix(a12, a22, n);
    // B21 + B22
    int** t2 = addMatrix(b21, b22, n);
    // P7 = t1 * t2
    p7 = strassen(t1, t2, n);

    deleteMatrix(t1);
    deleteMatrix(t2);
    deleteMatrix(a12);
    deleteMatrix(a22);
    deleteMatrix(b21);
    deleteMatrix(b22);

    // + send p7, task 3, tag 11
    pvm_itsend(PvmDataDefault);
    pvm_pkint(&(p7[0][0]), n*n, 1);
    pvm_send(tIds[3], 11);
}

pvm_barrier("strassen", NUM_PROCS);

```

```

int** q1 = nullptr;
int** q2 = nullptr;
int** q3 = nullptr;
int** q4 = nullptr;
int** q5 = nullptr;
int** q6 = nullptr;

if (rank == 0) {
    q1 = newMatrix(n);
    q2 = newMatrix(n);
    q3 = newMatrix(n);
    q4 = newMatrix(n);
    q5 = newMatrix(n);
    q6 = newMatrix(n);

    q1 = addMatrix(p1, p4, n);

    deleteMatrix(p1);
    deleteMatrix(p4);

    // recv q2, task 1, tag 12
    pvm_recv(tIds[1], 12);
    pvm_upkint(q2[0], n*n, 1);

    // recv q3, task 2, tag 13
    pvm_recv(tIds[2], 13);
    pvm_upkint(q3[0], n*n, 1);

    // recv q4, task 3, tag 14
    pvm_recv(tIds[3], 14);
    pvm_upkint(q4[0], n*n, 1);

    // recv q5, task 4, tag 15
    pvm_recv(tIds[4], 15);
    pvm_upkint(q5[0], n*n, 1);

    // recv q6, task 5, tag 16
    pvm_recv(tIds[5], 16);
    pvm_upkint(q6[0], n*n, 1);
}

if (rank == 1) {
    q2 = addMatrix(p2, p4, n);

    deleteMatrix(p2);
    deleteMatrix(p4);

    // send q2, task 0, tag 12
    pvm_initSend(PvmDataDefault);
    pvm_pkint(&(q2[0][0]), n*n, 1);
    pvm_send(tIds[0], 12);
}

if (rank == 2) {
    q3 = addMatrix(p3, p6, n);

    deleteMatrix(p3);
    deleteMatrix(p6);

    // send q3, task 0, tag 13
    pvm_initSend(PvmDataDefault);
    pvm_pkint(&(q3[0][0]), n*n, 1);
    pvm_send(tIds[0], 13);
}

if (rank == 3) {
    q4 = subMatrix(p7, p5, n);

```

```

        deleteMatrix(p5);
        deleteMatrix(p7);

        // send q4, task 0, tag 14
        pvm_initsend(PvmDataDefault);
        pvm_pkint(&(q4[0][0]), n*n, 1);
        pvm_send(tIds[0], 14);
    }

    if (rank == 4) {
        q5 = addMatrix(p3, p5, n);

        deleteMatrix(p3);
        deleteMatrix(p5);

        // send q5, task 0, tag 15
        pvm_initsend(PvmDataDefault);
        pvm_pkint(&(q5[0][0]), n*n, 1);
        pvm_send(tIds[0], 15);
    }

    if (rank == 5) {
        q6 = subMatrix(p1, p2, n);

        deleteMatrix(p1);
        deleteMatrix(p2);

        // send q6, task 0, tag 16
        pvm_initsend(PvmDataDefault);
        pvm_pkint(&(q6[0][0]), n*n, 1);
        pvm_send(tIds[0], 16);
    }

    pvm_barrier("strassen", NUM_PROCS);

    if (rank == 0) {
        int** c11 = addMatrix(q1, q4, n);
        int** c22 = addMatrix(q6, q3, n);

        result = combMatrix(c11, q5, q2, c22, n);

        deleteMatrix(c11);
        deleteMatrix(c22);

        deleteMatrix(q1);
        deleteMatrix(q2);
        deleteMatrix(q3);
        deleteMatrix(q4);
        deleteMatrix(q5);
        deleteMatrix(q6);
    }
}

int main(int argc, char** argv) {

    int parent = pvm_parent();

    // tag #0: get rank
    int rank;
    pvm_rcv(parent, 0);
    pvm_upkint(&rank, 1, 1);

    pvm_joygroup("strassen");

    int n;
    pvm_rcv(parent, 1);
    pvm_upkint(&n, 1, 1);

```



```

int** a = newMatrix(n);
int** b = newMatrix(n);

// tag #2: get a
pvm_recv(parent, 2);
pvm_upkint(a[0], n*n, 1);

// tag #3: get b
pvm_recv(parent, 3);
pvm_upkint(b[0], n*n, 1);

int* tIds = new int [NUM_PROCS];
// tag #18: get tIds
pvm_recv(parent, 18);
pvm_upkint(tIds, NUM_PROCS, 1);

int** result = nullptr;
strassen_pvm(a, b, n, result, rank, tIds);

pvm_barrier("strassen", NUM_PROCS);

if (rank == 0){
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&(result[0][0]), n*n, 1);
    pvm_send(parent, 17);
}
pvm_exit();
}

```