МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждениевысшего образования

«Самарский национальный исследовательский университетимени академика С.П. Королёва» (Самарский университет)

Институт информатики и кибернетики

Кафедра информационных систем и технологий

**ОТЧЕТ**

по лабораторной работе по дисциплине

«Облачные технологии»

Выполнили:

Алкеев М.Г., Гижевская В.Д., Грушенков М.А., Кремнев Д.А., гр. 6131-020402D

Проверил:

доцент каф. ИСТ, д.т.н. Востокин С.В.

Самара 2023

# ЗАДАНИЕ

Сформулируйте задачу, опишите метод решения с иллюстрацией взаимодействия акторов. Опишите процесс в виде алгоритма обработки сообщений в акторах (в Templet SDK). Убедитесь в его корректности. Выделите в исходном алгоритме "задачи". Переработайте алгоритм, убедитесь в корректности нового алгоритма. С использованием системы имитационного моделирования исследуйте потенциальное ускорение алгоритма. Реализуйте задачи в виде консольных приложений. Организуйте их распределенное исполнение в системе Everest. Этапы выполнения задания оформите в письменном отчете.

**СОДЕРЖАНИЕ**

[Задание 2](#_Toc136473578)

[Введение 4](#_Toc136473579)

[1 Формулировка задачи, словесное описание метода решения с иллюстрацией взаимодействия акторов 5](#_Toc136473580)

[2 Описание процесса в виде алгоритма обработки сообщений в акторах (в Templet SDK) и проверка его корректности 6](#_Toc136473581)

[3 Выделение в исходном алгоритме "задачи", переработка алгоритма, проверка корректности нового алгоритма 10](#_Toc136473582)

[4 Исследование потенциального ускорения алгоритма с использованием системы имитационного моделирования 15](#_Toc136473583)

[4 Реализация задачи в виде консольных приложений и организация их распределенного исполнения в системе Everest 20](#_Toc136473584)

[Заключение 26](#_Toc136473585)

[Список использованных источников 27](#_Toc136473586)

**ВВЕДЕНИЕ**

Умножение матриц на кольце процессов - это параллельный алгоритм умножения матриц, который использует несколько процессов, организованных в кольцо. Кольцо состоит из последовательности процессов, связанных между собой таким образом, что каждый процесс передает данные следующему процессу в кольце до тех пор, пока не будет завершено умножение матриц.

Умножение матриц на кольце процессов используется для распараллеливания алгоритма умножения матриц и ускорения вычислений. Оно может быть полезным в следующих случаях:

* умножение матриц – вычислительно интенсивная операция, особенно для больших матриц, поэтому использование кольца процессов позволяет распределить вычислительную нагрузку между несколькими процессами и выполнять умножение матриц параллельно, что приводит к ускорению вычислений и сокращению времени выполнения;
* умножение матриц на кольце процессов обладает хорошей масштабируемостью, поэтому при увеличении количества процессов можно добиться еще большего ускорения вычислений;
* при использовании кольца процессов каждый процесс работает с определенным блоком данных, что позволяет эффективно распределить данные между процессами, и это особенно полезно при работе с большими матрицами, когда данные не могут быть хранены в оперативной памяти одного процесса;

В целом, умножение матриц на кольце процессов позволяет улучшить производительность и эффективность вычислений при работе с матрицами больших размеров и при использовании параллельных или распределенных систем

# 1 Формулировка задачи, словесное описание метода решения с иллюстрацией взаимодействия акторов

Разрабатываемая система должна проводить умножение матриц на кольце, используя модель акторов для распределенных вычислений.

В системе есть 2 вида акторов: «Master» и «Slave».

Актор «Master» является стартовым, распределяющим начальные индексы строк и столбцов матриц A и B для исполняющих акторов «Slave».

Акторы «Slave» хранят в себе ссылку на следующего актора «Slave» для передачи сообщения, в котором содержится индекс столбца. При получении сообщения актор выполняет вычисление и сохраняет результат в матрицу C.

Структура взаимодействия акторов на примере матриц размерностью 3 на 3 изображена на рисунке 1.

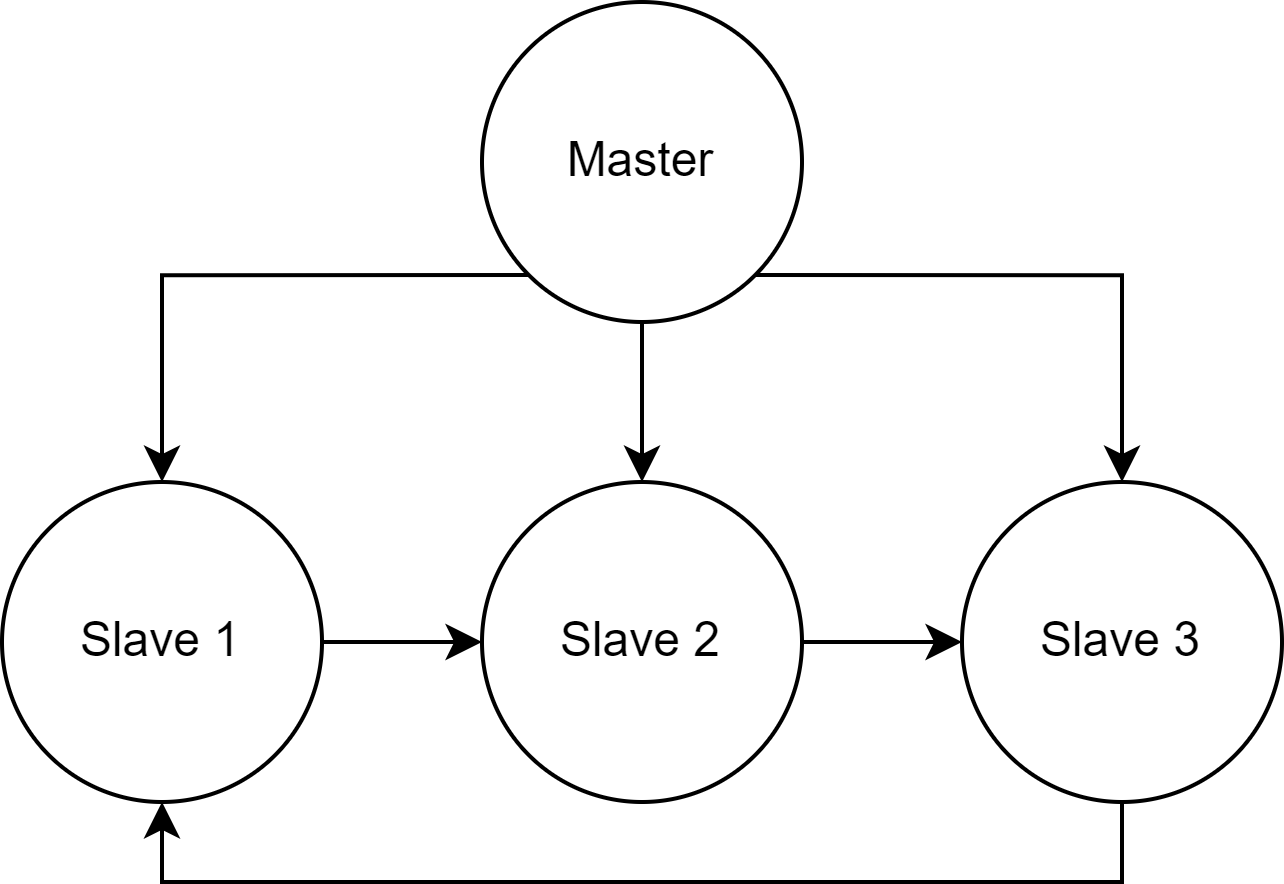


Рисунок 1 – Структура взаимодействия акторов

# 2 ****Описание процесса в виде алгоритма обработки сообщений в акторах (в Templet SDK) и проверка его корректности****

Листинг разработанного приложения, представляющего вышеописанный алгоритм взаимодействия акторов, приведен ниже:

#include "..\\..\\konst\Downloads\templet.hpp" #include <iostream>

using namespace templet; using namespace std;

const int N = 3;

double A[N][N] = { {1.0,2.0,3.0},

{1.0,2.0,3.0},

{1.0,2.0,3.0} };

double B[N][N] = { {1.0,1.0,1.0},

{2.0,2.0,2.0},

{3.0,3.0,3.0} };

double C[N][N] = { 0.0 };

class column : public templet::message { public:

column(templet::actor\* a = 0, templet::message\_adaptor ma = 0)

:templet::message(a, ma) {} int j; // column index

};

/\*$TET$\*/

#pragma templet !Link(in?column,out!column,stop!message) struct Link :public templet::actor {

static void on\_in\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_in(\*(column\*)m);

}

static void on\_out\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_out(\*(column\*)m);

}

static void on\_stop\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_stop(\*(message\*)m);

}

Link(templet::engine& e) :Link() { Link::engines(e);

}

Link() :templet::actor(true), out(this, &on\_out\_adapter), stop(this, &on\_stop\_adapter)

{

/\*$TET$Link$Link\*/ num\_of\_ready\_columns = 0;

/\*$TET$\*/

}

void engines(templet::engine& e) { templet::actor::engine(e);

/\*$TET$Link$engines\*//\*$TET$\*/

}

void start() {

/\*$TET$Link$start\*/ on\_in(out);

/\*$TET$\*/

}

inline void on\_in(column& m) {

/\*$TET$Link$in\*/

if (num\_of\_ready\_columns != 0 && &m == &out) return;

int j = m.j;

for (int k = 0; k < N; k++) C[i][j] += A[i][k] \* B[k][j];

next->in(m); m.send();

if (++num\_of\_ready\_columns == N) stop.send();

/\*$TET$\*/

}

inline void on\_out(column& m) {

/\*$TET$Link$out\*/

/\*$TET$\*/

}

inline void on\_stop(message& m) {

/\*$TET$Link$stop\*/

/\*$TET$\*/

}

void in(column& m) { m.bind(this, &on\_in\_adapter); } column out;

message stop;

/\*$TET$Link$$footer\*/ int i; // row index

int num\_of\_ready\_columns; Link\* next;

/\*$TET$\*/

};

#pragma templet !Stopper(in?message)

struct Stopper :public templet::actor {

static void on\_in\_adapter(templet::actor\* a, templet::message\* m) { ((Stopper\*)a)->on\_in(\*(message\*)m);

}

Stopper(templet::engine& e) :Stopper() { Stopper::engines(e);

}

Stopper() :templet::actor(true)

{

/\*$TET$Stopper$Stopper\*/ num\_of\_ready\_rows = 0;

/\*$TET$\*/

}

void engines(templet::engine& e) { templet::actor::engine(e);

/\*$TET$Stopper$engines\*/

/\*$TET$\*/

}void start() {

/\*$TET$Stopper$start\*/

/\*$TET$\*/

}

inline void on\_in(message& m) {

/\*$TET$Stopper$in\*/

if (++num\_of\_ready\_rows == N) stop();

/\*$TET$\*/

}

void in(message& m) { m.bind(this, &on\_in\_adapter); }

/\*$TET$Stopper$$footer\*/ int num\_of\_ready\_rows;

/\*$TET$\*/

};

/\*$TET$$footer\*/

/\*$TET$\*/

int main()

{

engine eng;

Stopper stopper(eng); Link l[N];

for (int i = 0; i < N; i++) { l[i].engines(eng); stopper.in(l[i].stop); l[i].next = &l[(i + 1) % N];

l[i].i = i; l[i].out.j = i;

}

eng.start();

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

std::cout << " C[" << i << "," << j << "] = " << C[i][j];

}

std::cout << endl;

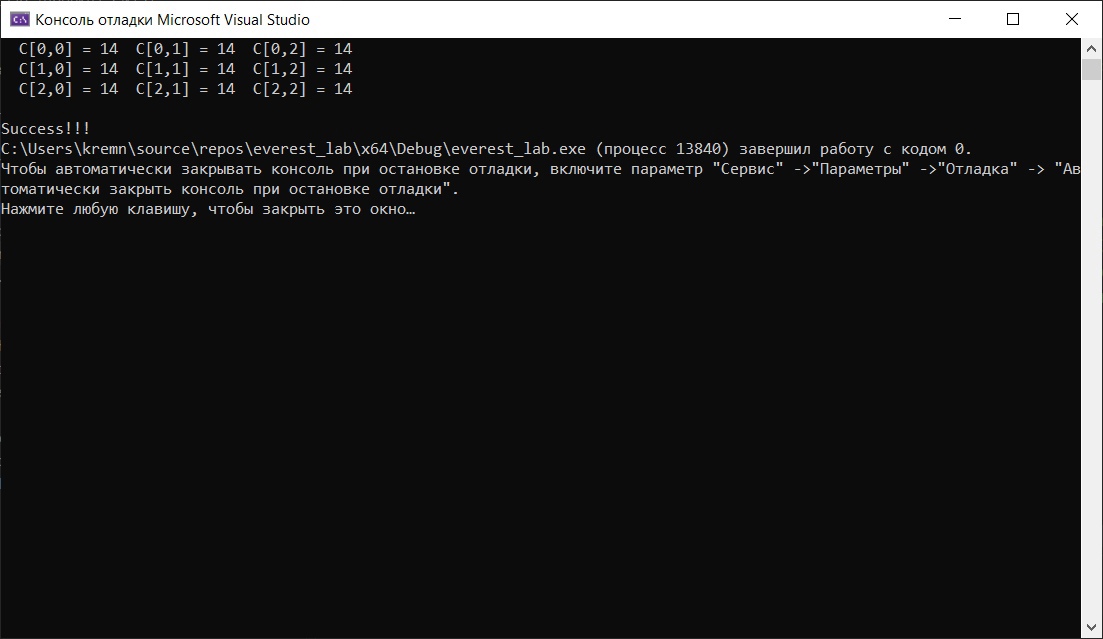
}

std::cout << endl;

std::cout << (eng.stopped() ? "Success!!!" : "Failure...");

}

Результат работы на исходных матрицах представлен на рисунке 2.

  
Рисунок 2 – Результат работы программы

# 3 Выделение в исходном алгоритме "задачи", переработка алгоритма, проверка корректности нового алгоритма

В качестве задачи, в исходном алгоритме было выделено скалярное произведение строки на столбец. Сравнение было выведено в отдельный класс задачи Multiplytask. Задача запускается в акторах «Slave» при получении сообщения от соседнего актора.

Ниже описан код с выделением задачи для исходного алгоритма:

#include "..\\..\\konst\Downloads\templet.hpp" #include <iostream>

using namespace templet; using namespace std;

const int N = 3;

double A[N][N] = { {1.0,2.0,3.0},

{1.0,2.0,3.0},

{1.0,2.0,3.0} };

double B[N][N] = { {1.0,1.0,1.0},

{2.0,2.0,2.0},

{3.0,3.0,3.0} };

double C[N][N] = { 0.0 };

class column : public templet::message { public:

column(templet::actor\* a = 0, templet::message\_adaptor ma = 0)

:templet::message(a, ma) {} int j; // column index

};

class Multiplytask : protected templet::base\_task { public:

Multiplytask(templet::actor\* a, templet::task\_adaptor ta) :base\_task(a, ta) {} void engine(templet::base\_engine& te) { templet::base\_task::engine(te); }

void submit\_k(int r, int c) { i = r; j = c; submit(); } private:

void run() override { for (int k = 0; k < N; k++) C[i][j] += A[i][k] \* B[k][j]; } int i;

int j;

};

/\*$TET$\*/

#pragma templet !Link(in?column,out!column,stop!message) struct Link :public templet::actor {

static void on\_in\_adapter(templet::actor\* a, templet::message\* m) {

((Link\*)a)->on\_in(\*(column\*)m);

}

static void on\_out\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_out(\*(column\*)m);

}

static void on\_stop\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_stop(\*(message\*)m);

}

static void on\_t\_adapter(templet::actor\* a, templet::task\* t) { ((Link\*)a)->on\_t(\*(Multiplytask\*)t);

}

Link(templet::engine& e, templet::base\_engine& te\_base) :Link() { Link::engines(e,te\_base);

}

Link() :templet::actor(true), out(this, &on\_out\_adapter), stop(this, &on\_stop\_adapter), t(this, &on\_t\_adapter)

{

/\*$TET$Link$Link\*/ num\_of\_ready\_columns = 0;

/\*$TET$\*/

}

void engines(templet::engine& e, templet::base\_engine& te\_base) { templet::actor::engine(e);

t.engine(te\_base);

/\*$TET$Link$engines\*/

/\*$TET$\*/

}

void start() {

/\*$TET$Link$start\*/ on\_in(out);

/\*$TET$\*/

}

inline void on\_in(column& m) {

/\*$TET$Link$in\*/

if (num\_of\_ready\_columns != 0 && &m == &out) return;

std::cout << i << "actor receive" << m.j << std::endl; int j = m.j;

t.submit\_k(i, j);

std::cout << i << "actor send" << m.j << std::endl; next->in(m); m.send();

if (++num\_of\_ready\_columns == N) stop.send();

/\*$TET$\*/

}

inline void on\_out(column& m) {

/\*$TET$Link$out\*/

/\*$TET$\*/

}

inline void on\_t(Multiplytask& t) {

/\*$TET$mediator$t\*/

//t.delay(DELAY);

/\*$TET$\*/

}

inline void on\_stop(message& m) {

/\*$TET$Link$stop\*/

/\*$TET$\*/

}

void in(column& m) { m.bind(this, &on\_in\_adapter); } column out;

column inner; message stop; Multiplytask t;

/\*$TET$Link$$footer\*/ int i; // row index

int num\_of\_ready\_columns; Link\* next;

/\*$TET$\*/

};

#pragma templet !Stopper(in?message)

struct Stopper :public templet::actor {

static void on\_in\_adapter(templet::actor\* a, templet::message\* m) { ((Stopper\*)a)->on\_in(\*(message\*)m);

}

Stopper(templet::engine& e) :Stopper() { Stopper::engines(e);

}

Stopper() :templet::actor(true)

{

/\*$TET$Stopper$Stopper\*/ num\_of\_ready\_rows = 0;

/\*$TET$\*/

}

void engines(templet::engine& e) { templet::actor::engine(e);

/\*$TET$Stopper$engines\*/

/\*$TET$\*/

}

void start() {

/\*$TET$Stopper$start\*/

/\*$TET$\*/

}

inline void on\_in(message& m) {

/\*$TET$Stopper$in\*/

if (++num\_of\_ready\_rows == N) stop();

/\*$TET$\*/

}

void in(message& m) { m.bind(this, &on\_in\_adapter); }

/\*$TET$Stopper$$footer\*/ int num\_of\_ready\_rows;

/\*$TET$\*/

};

/\*$TET$$footer\*/

/\*$TET$\*/

int main()

{

engine eng; templet::base\_engine teng;

Stopper stopper(eng); Link l[N];

for (int i = 0; i < N; i++) { l[i].engines(eng,teng); stopper.in(l[i].stop); l[i].next = &l[(i + 1) % N];

l[i].i = i; l[i].out.j = i;

}

eng.start();

teng.run();

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

std::cout << " C[" << i << "," << j << "] = " << C[i][j];

}

std::cout << endl;

}

std::cout << endl;

std::cout << (eng.stopped() ? "Success!!!" : "Failure...");

}

Результат работы на исходных матрицах представлен на рисунке 3.

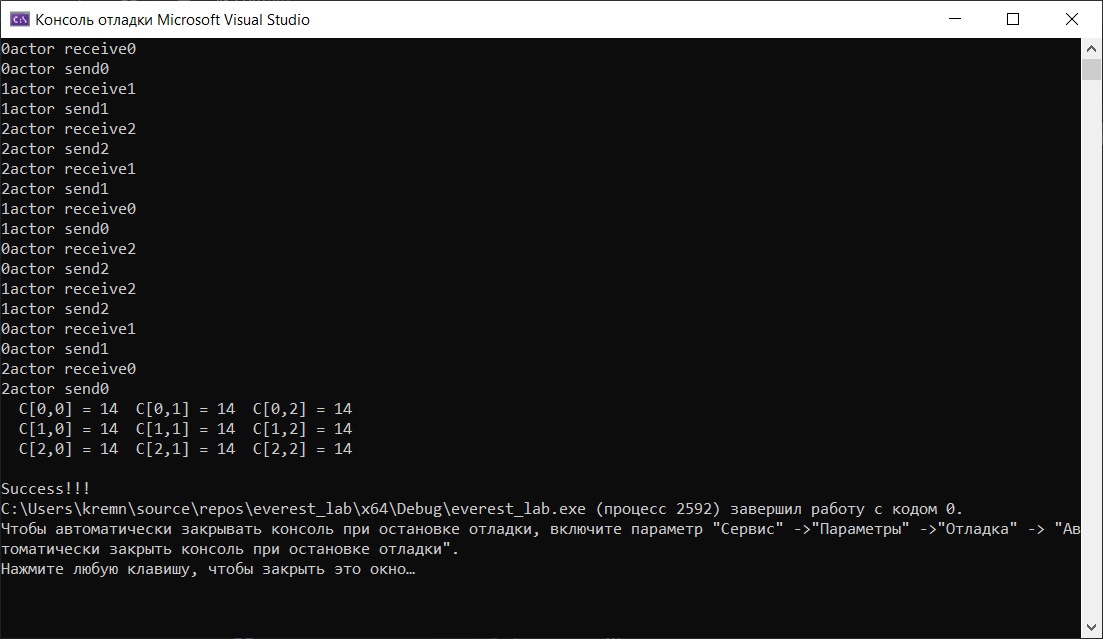


Рисунок 3 – Результат работы программы

# 4 Исследование потенциального ускорения алгоритма с использованием системы имитационного моделирования

В качестве задачи, в исходном алгоритме было выделено скалярное произведение строки на столбец. Сравнение было выведено в отдельный класс задачи Multiplytask. Задача запускается в акторах «Slave» при получении сообщения от соседнего актора.

Для исследования потенциального ускорения алгоритма переработаем алгоритм с использованием имитационного движка Basesim\_engine. Листинг переработанного алгоритма представлен ниже.

#include "..\\..\\konst\Downloads\templet.hpp" #include "..\\..\\konst\Downloads\basesim.hpp" #include <iostream>

using namespace templet; using namespace std;

const int N = 3;

double A[N][N] = { {1.0,2.0,3.0},

{1.0,2.0,3.0},

{1.0,2.0,3.0} };

double B[N][N] = { {1.0,1.0,1.0},

{2.0,2.0,2.0},

{3.0,3.0,3.0} };

double C[N][N] = { 0.0 };

class column : public templet::message { public:

column(templet::actor\* a = 0, templet::message\_adaptor ma = 0)

:templet::message(a, ma) {} int j; // column index

};

class Multiplytask : protected templet::base\_task { public:

Multiplytask(templet::actor\* a, templet::task\_adaptor ta) :base\_task(a, ta) {} void engine(templet::base\_engine& te) { templet::base\_task::engine(te); }

void submit\_k(int r, int c) { i = r; j = c; submit(); } private:

void run() override { for (int k = 0; k < N; k++) C[i][j] += A[i][k] \* B[k][j]; } int i;

int j;

};

/\*$TET$\*/

#pragma templet !Link(in?column,out!column,stop!message) struct Link :public templet::actor {

static void on\_in\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_in(\*(column\*)m);

}

static void on\_out\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_out(\*(column\*)m);

}

static void on\_stop\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_stop(\*(message\*)m);

}

static void on\_t\_adapter(templet::actor\* a, templet::task\* t) { ((Link\*)a)->on\_t(\*(basesim\_task\*)t);

}Link(templet::engine& e, templet::basesim\_engine& te\_base) :Link() { Link::engines(e,te\_base);

}

Link() :templet::actor(true), out(this, &on\_out\_adapter), stop(this, &on\_stop\_adapter), t(this, &on\_t\_adapter)

{

/\*$TET$Link$Link\*/ num\_of\_ready\_columns = 0;

/\*$TET$\*/

}

void engines(templet::engine& e, templet::basesim\_engine& te\_base) { templet::actor::engine(e);

t.engine(te\_base);

/\*$TET$Link$engines\*/

/\*$TET$\*/

}

void start() {

/\*$TET$Link$start\*/ on\_in(out);

/\*$TET$\*/

}

inline void on\_in(column& m) {

/\*$TET$Link$in\*/

if (num\_of\_ready\_columns != 0 && &m == &out) return;

std::cout << i << "actor receive" << m.j << std::endl; int j = m.j;

inner = m; t.submit();

std::cout << i << "actor send" << m.j << std::endl; next->in(m); m.send();

if (++num\_of\_ready\_columns == N) stop.send();

/\*$TET$\*/

}

inline void on\_out(column& m) {

/\*$TET$Link$out\*/

/\*$TET$\*/

}

inline void on\_t(basesim\_task& t) {

/\*$TET$mediator$t\*/ t.delay(1);

for (int k = 0; k < N; k++) C[i][inner.j] += A[i][k] \* B[k][inner.j];

/\*$TET$\*/

}

inline void on\_stop(message& m) {

/\*$TET$Link$stop\*/

/\*$TET$\*/

}

void in(column& m) { m.bind(this, &on\_in\_adapter); } column out;

column inner; message stop; basesim\_task t;/\*$TET$Link$$footer\*/ int i; // row index

int num\_of\_ready\_columns; Link\* next;

/\*$TET$\*/

};

#pragma templet !Stopper(in?message)

struct Stopper :public templet::actor {

static void on\_in\_adapter(templet::actor\* a, templet::message\* m) { ((Stopper\*)a)->on\_in(\*(message\*)m);

}

Stopper(templet::engine& e) :Stopper() { Stopper::engines(e);

}

Stopper() :templet::actor(true)

{

/\*$TET$Stopper$Stopper\*/ num\_of\_ready\_rows = 0;

/\*$TET$\*/

}

void engines(templet::engine& e) { templet::actor::engine(e);

/\*$TET$Stopper$engines\*/

/\*$TET$\*/

}

void start() {

/\*$TET$Stopper$start\*/

/\*$TET$\*/

}

inline void on\_in(message& m) {

/\*$TET$Stopper$in\*/

if (++num\_of\_ready\_rows == N) stop();

/\*$TET$\*/

}

void in(message& m) { m.bind(this, &on\_in\_adapter); }

/\*$TET$Stopper$$footer\*/ int num\_of\_ready\_rows;

/\*$TET$\*/

};

/\*$TET$$footer\*/

/\*$TET$\*/

int main()

{

engine eng; templet::basesim\_engine teng;

Stopper stopper(eng); Link l[N];

for (int i = 0; i < N; i++) { l[i].engines(eng,teng); stopper.in(l[i].stop); l[i].next = &l[(i + 1) % N];l[i].i = i; l[i].out.j = i;

}

eng.start();

teng.run();

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

std::cout << " C[" << i << "," << j << "] = " << C[i][j];

}

std::cout << endl;

}

std::cout << endl;

std::cout << (eng.stopped() ? "Success!!!" : "Failure...");

std::cout << "Maximum number of tasks executed in parallel : " << teng.Pmax() << std::endl;

std::cout << "Time of sequential execution of all tasks : " << teng.T1() << std::endl;

std::cout << "Time of parallel execution of all tasks : " << teng.Tp() << std::endl;

}

Результат работы на исходных матрицах представлен на рисунке 4.

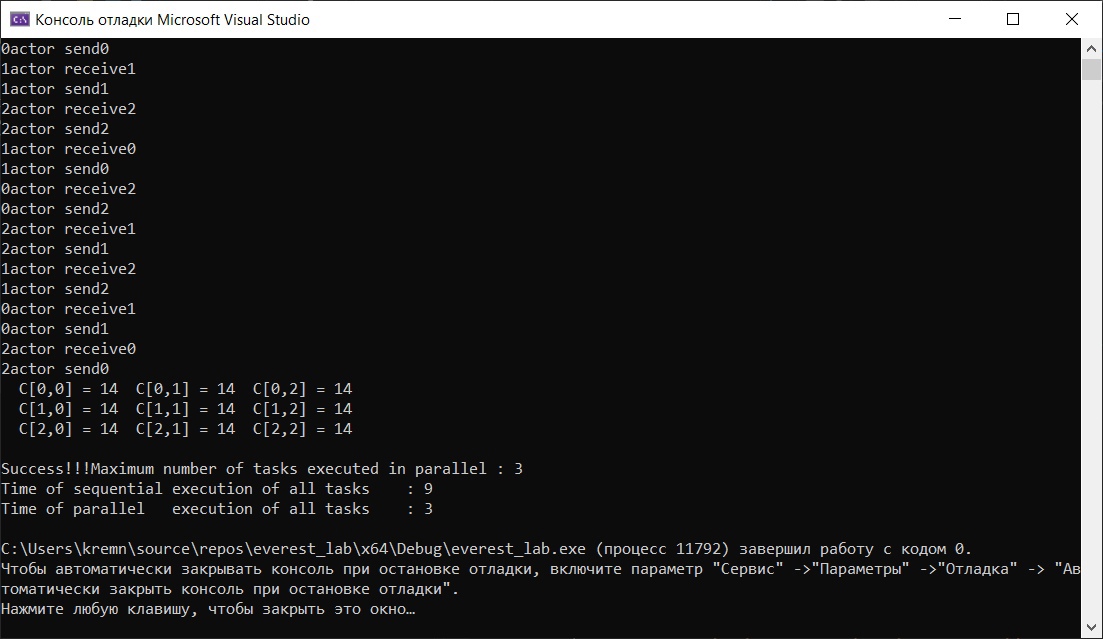


Рисунок 4 – Результаты имитационного моделирования

Как видно из результатов моделирования, потенциальное ускорение алгоритма равно 9/3 = 3.

# 4 Реализация задачи в виде консольных приложений и организация их распределенного исполнения в системе Everest

Для исполнения задач в системе Everest был переработан алгоритм. При получении сообщения, актор «Slave» отправляет в созданное приложение в системе Everest 6 чисел (первые 3 – строка, вторые – столбец). В приложении вычисляется скалярное произведение и полученный результат возвращается в исходную программу.

Скриншоты созданного приложения в системе Everest представлены на рисунках 5,6.

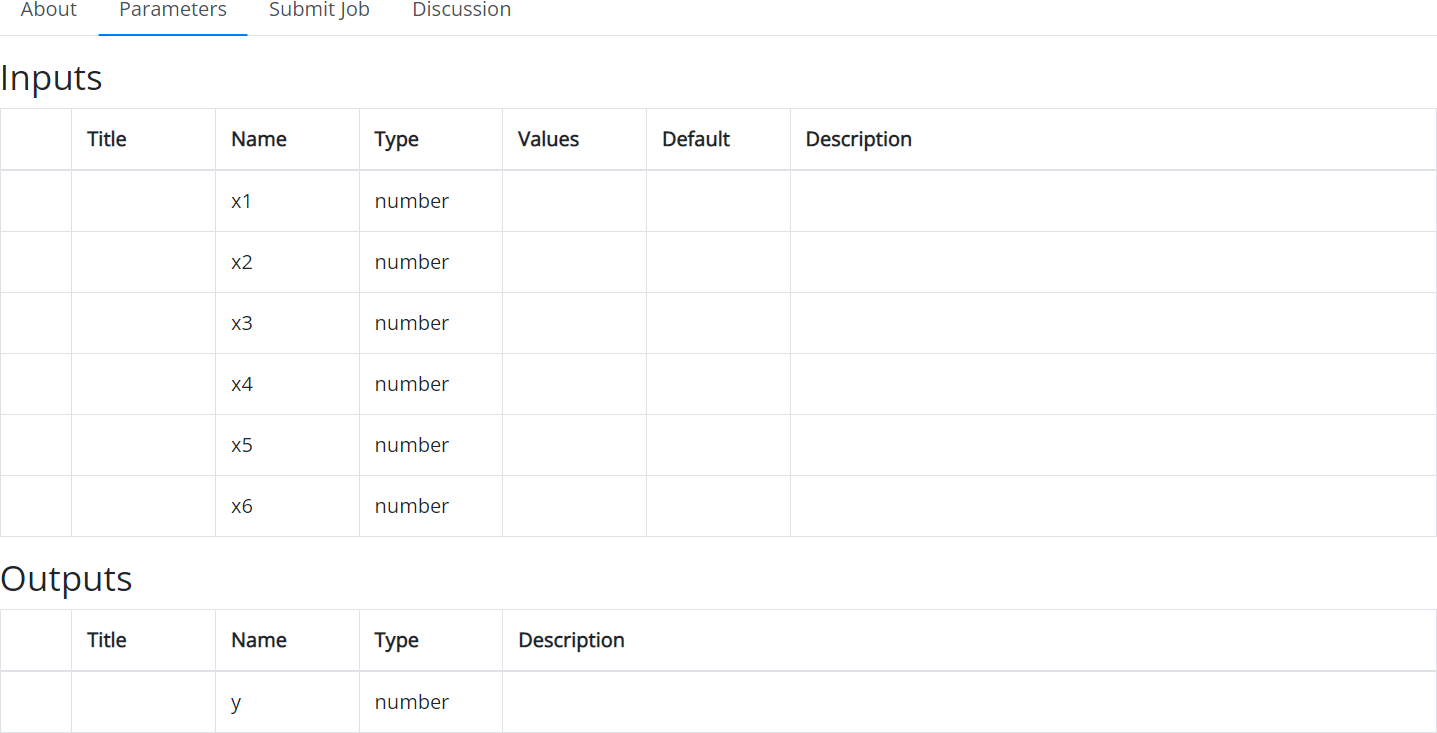
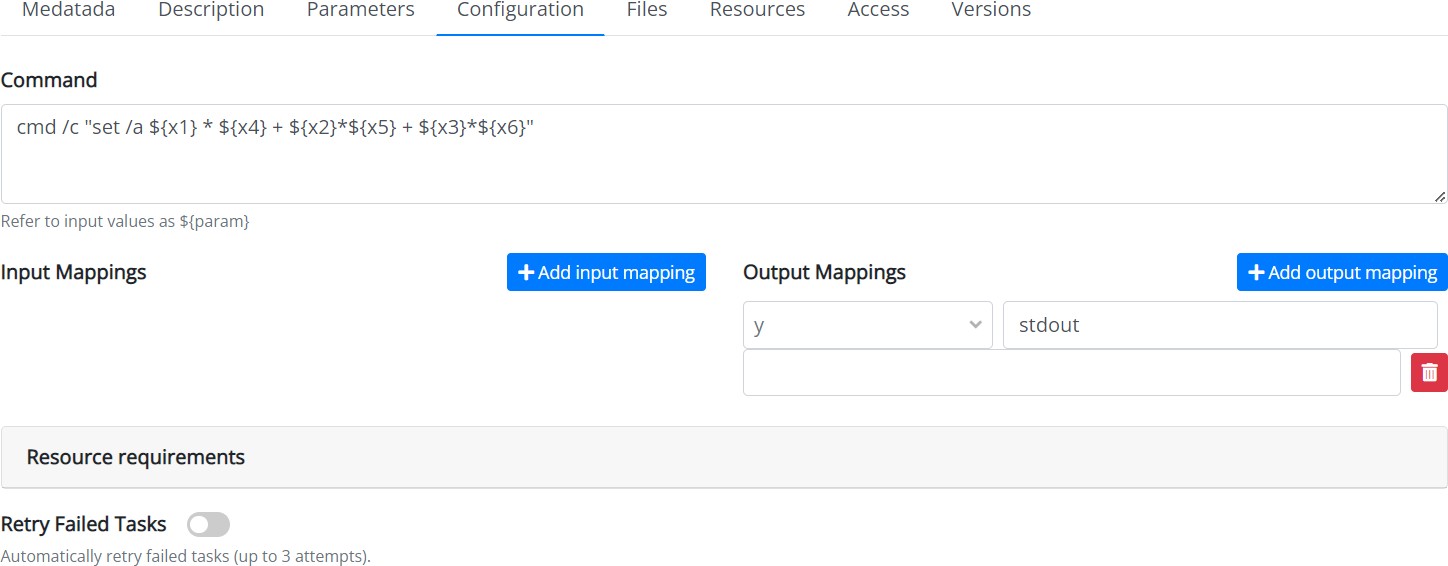


Рисунок 5 – Входные и выходные параметры

Рисунок 6 – Конфигурация приложения

Листинг переработанного алгоритма представлен ниже.

#include "..\\..\\konst\Downloads\templet.hpp" #include "..\\..\\konst\Downloads\everest.hpp" #include <iostream>

using namespace templet; using namespace std;

const int N = 3;

double A[N][N] = { {1.0,2.0,3.0},

{1.0,2.0,3.0},

{1.0,2.0,3.0} };

double B[N][N] = { {1.0,1.0,1.0},

{2.0,2.0,2.0},

{3.0,3.0,3.0} };

double C[N][N] = { 0.0 };

class column : public templet::message { public:

column(templet::actor\* a = 0, templet::message\_adaptor ma = 0)

:templet::message(a, ma) {} int j; // column index

};

class Multiplytask : protected templet::base\_task { public:

Multiplytask(templet::actor\* a, templet::task\_adaptor ta) :base\_task(a, ta) {} void engine(templet::base\_engine& te) { templet::base\_task::engine(te); }

void submit\_k(int r, int c) { i = r; j = c; submit(); } private:

void run() override { for (int k = 0; k < N; k++) C[i][j] += A[i][k] \* B[k][j]; } int i;

int j;

};

/\*$TET$\*/

#pragma templet !Link(in?column,out!column,stop!message) struct Link :public templet::actor {

static void on\_in\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_in(\*(column\*)m);

}

static void on\_out\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_out(\*(column\*)m);

}

static void on\_stop\_adapter(templet::actor\* a, templet::message\* m) { ((Link\*)a)->on\_stop(\*(message\*)m);

}

static void on\_t\_adapter(templet::actor\* a, templet::task\* t) { ((Link\*)a)->on\_t(\*(everest\_task\*)t);

}

Link(templet::engine& e, templet::everest\_engine& te\_base) :Link() { Link::engines(e, te\_base);

}

Link() :templet::actor(true)out(this, &on\_out\_adapter), stop(this, &on\_stop\_adapter), t(this, &on\_t\_adapter)

{

/\*$TET$Link$Link\*/ t.app\_id("646e1c50100000160083a3e8");

num\_of\_ready\_columns = 0;

/\*$TET$\*/

}

void engines(templet::engine& e, templet::everest\_engine& te\_base) { templet::actor::engine(e);

t.engine(te\_base);

/\*$TET$Link$engines\*/

/\*$TET$\*/

}

void start() {

/\*$TET$Link$start\*/ on\_in(out);

/\*$TET$\*/

}

inline void on\_in(column& m) {

/\*$TET$Link$in\*/

if (num\_of\_ready\_columns != 0 && &m == &out) return; inner = m;

std::cout << i << "actor receive" << m.j << std::endl; int j = m.j;

json in;

in["name"] = "lab\_vostokin"; in["inputs"]["x1"] = A[i][0];

in["inputs"]["x2"] = A[i][1];

in["inputs"]["x3"] = A[i][2];

in["inputs"]["x4"] = B[0][j];

in["inputs"]["x5"] = B[1][j];

in["inputs"]["x6"] = B[2][j];

if (t.submit(in)) std::cout << "task submit succeeded" << std::endl; else std::cout << "task submit failed" << std::endl;

std::cout << i << "actor send" << m.j << std::endl; next->in(m); m.send();

if (++num\_of\_ready\_columns == N) stop.send();

/\*$TET$\*/

}

inline void on\_out(column& m) {

/\*$TET$Link$out\*/

/\*$TET$\*/

}

inline void on\_t(everest\_task& t) {

/\*$TET$mediator$t\*/ json out = t.result();

C[i][inner.j] = out["y"];

/\*$TET$\*/

}

inline void on\_stop(message& m) {

/\*$TET$Link$stop\*//\*$TET$\*/

}

void in(column& m) { m.bind(this, &on\_in\_adapter); } column out;

column inner; message stop; everest\_task t;

/\*$TET$Link$$footer\*/ int i; // row index

int num\_of\_ready\_columns; Link\* next;

/\*$TET$\*/

};

#pragma templet !Stopper(in?message)

struct Stopper :public templet::actor {

static void on\_in\_adapter(templet::actor\* a, templet::message\* m) { ((Stopper\*)a)->on\_in(\*(message\*)m);

}

Stopper(templet::engine& e) :Stopper() { Stopper::engines(e);

}

Stopper() :templet::actor(true)

{

/\*$TET$Stopper$Stopper\*/ num\_of\_ready\_rows = 0;

/\*$TET$\*/

}

void engines(templet::engine& e) { templet::actor::engine(e);

/\*$TET$Stopper$engines\*/

/\*$TET$\*/

}

void start() {

/\*$TET$Stopper$start\*/

/\*$TET$\*/

}

inline void on\_in(message& m) {

/\*$TET$Stopper$in\*/

if (++num\_of\_ready\_rows == N) stop();

/\*$TET$\*/

}

void in(message& m) { m.bind(this, &on\_in\_adapter); }

/\*$TET$Stopper$$footer\*/ int num\_of\_ready\_rows;

/\*$TET$\*/

};

/\*$TET$$footer\*/

/\*$TET$\*/

int main()

{

// 646e1c50100000160083a3e8engine eng;

templet::everest\_engine teng("rtyswe", "\*\*\*\*\*\*\*\*\*\*\*\*");

Stopper stopper(eng); Link l[N];

for (int i = 0; i < N; i++) {

l[i].engines(eng, teng); stopper.in(l[i].stop); l[i].next = &l[(i + 1) % N];

l[i].i = i; l[i].out.j = i;

}

eng.start();

teng.run();

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

std::cout << " C[" << i << "," << j << "] = " << C[i][j];

}

std::cout << endl;

}

std::cout << endl;

std::cout << (eng.stopped() ? "Success!!!" : "Failure...");

}

# ЗАКЛЮЧЕНИЕ

В результате выполнения работы мы сформулировали задачу, описали метод решения с иллюстрацией взаимодействия акторов; описали процесс в виде алгоритма обработки сообщений в акторах (в Templet SDK) и убедились в его корректности; выделили в исходном алгоритме "задачи"; переработали алгоритм и убедились в корректности нового алгоритма; с использованием системы имитационного моделирования исследовали потенциальное ускорение алгоритма; реализовали задачи в виде консольных приложений и организовали их распределенное исполнение в системе Everest.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Karniadakis G.E. Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and their Implementation [Текст] / George Em Karniadakis, Robert M. Kirby II и Pekka Neittaanmäki − Самара, 2003. – 628 с.

2 Xiaoye S. Li. Communication-Avoiding Parallel Matrix Multiplication on Process Rings / Xiaoye S. Li, Si Zhang, Lexing Ying и Richard Vuduc – Berkeley, 2016. – 72с.

3 Волков, С. Ю. Реализация запуска многовариантных расчетов на платформе Everest / С. Ю. Волков, О. В. Сухорослов // Программные системы: теория и приложения. — 2014. — Т. 5(4). — С. 183-194.