

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»

Институт Информатики и кибернетики   
Кафедра Программных систем

**ОТЧЕТ**

к лабораторной работе по дисциплине

«Облачные технологии»

Выполнили:

Дьяконов А.В., Лазарев М.Ю.

гр. 6132-020402D

Проверил:

Заведующий каф. ПС, д.т.н. Востокин С.В.

Самара 2025

СОДЕРЖАНИЕ

[Введение 3](#_Toc198757045)

[1 Словесное описание метода решения с иллюстрацией взаимодействия акторов 4](#_Toc198757046)

[2 Первый этап - поместить код в отдельный каталог, провести логическую отладку, собрать код компилятором GCC 5](#_Toc198757047)

[3 Второй этап – выделить в коде задачи, задать длительность их исполнения в единицах модельного времени, исследовать параллелизм 8](#_Toc198757048)

[4 Третий этап – реализовать распределенные вычисления задач в облаке с использованием платформы Everest 12](#_Toc198757049)

[Заключение 20](#_Toc198757050)

[Список использованных источников 21](#_Toc198757051)

**Введение**

Технологии параллельного программирования численных методов решения уравнений в частных производных имеют большое прикладное значение. При помощи уравнений в частных производных описываются разнообразные физические процессы в непрерывных средах.

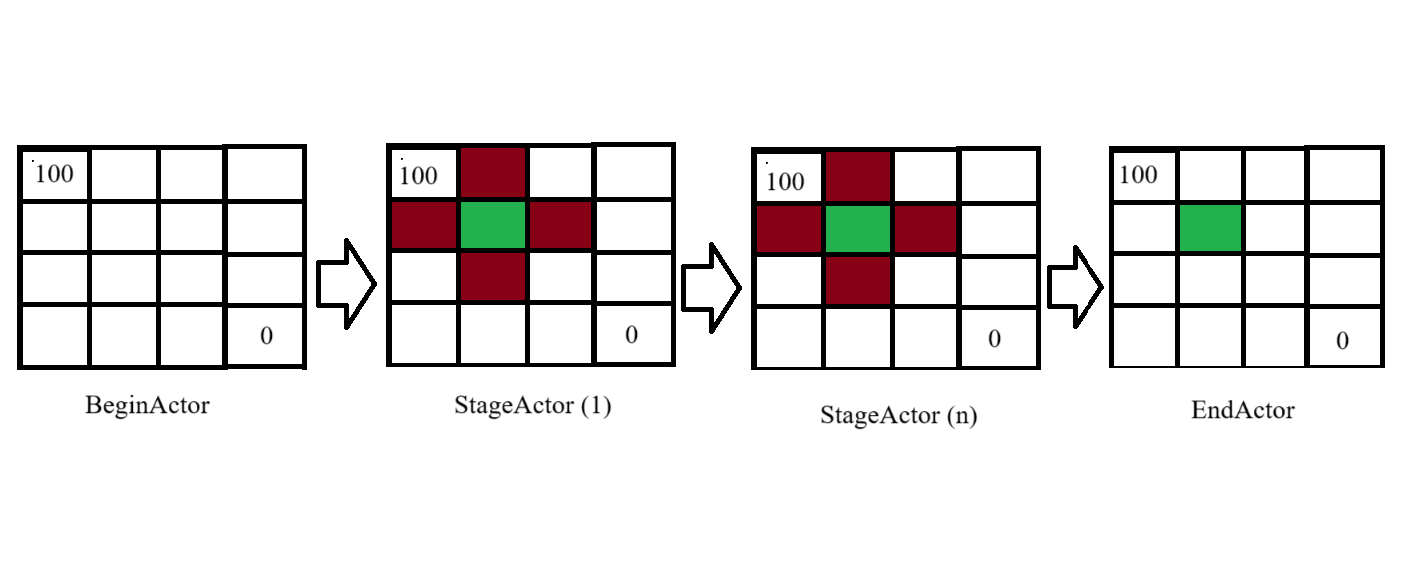
Алгоритмическим содержанием решения уравнений в частных производных являются итерационные методы решения систем линейных уравнений большой размерности с матрицами специального вида. Компоненты вектора неизвестных из этих уравнений отображаются на точки области пространства (сеточная область) и содержат значения некоторых физических параметров в этих точках. В данной работе мы рассматриваем процесс теплопроводности - волновой процесс по алгоритму Гаусса – Зейделя на двумерной области с расчетом поля температуры.

1. Словесное описание метода решения с иллюстрацией взаимодействия акторов

Система состоит из трех акторов. Первый актор BeginActor начинает волновой процесс и передаёт задачу посредством сообщения BorderMessage во второй актор StageActor. В StageActor происходит процесс вычисления, при этом количество акторов на данном этапе зависит от количества указанных итераций («волн»), по окончанию каждой волны последовательно результат передается следующему StageActor. Последний StageActor по окончании всех итераций передает рассчитанные значения в последний третий актор EndActor, который в свою фиксирует состояние завершения вычислений и останавливает систему. В итоге мы получаем рассчитанное поле температуры на двумерной сеточной области.

Вычисление в StageActor происходит посредством пятиточечного принципа, где на результат влияют ближайшие соседи (левый, правый, верхний и нижний соответственно). Для вычислений мы создаем сетку 10 на 10, расширяем её до 12 на 12, чтобы не выходить за пределы границ и получать более точные вычисления. Определяем граничные значения сверху и слева как 100 градусов, а справа и снизу как 0. Получаем сетку, где правый нижний угол «холодный», а верхний левый – «горячий».

Структура расчета и взаимодействия акторов схематически показаны на рисунке 1.

  
Рисунок 1 – Структура взаимодействия акторов и расчетов

1. **Первый этап – поместить код в отдельный каталог, провести логическую отладку, собрать код компилятором GCC**

#pragma once

#include <templet.hpp>

#include <iostream>

using namespace templet;

using namespace std;

inline constexpr int NX = 10;

inline constexpr int NY = 10;

inline constexpr int T = 10;

inline double area[NX+2][NY+2] = {};

class border : public templet::message {

public:

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

templet::message(a, ma) {}};

#pragma templet !Begin(out!border)

struct Begin :public templet::actor {

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

((Begin\*)a)->on\_out(\*(border\*)m);}

Begin(templet::engine&e) :Begin() {

Begin::engines(e);}

Begin() :templet::actor(true),

out(this, &on\_out\_adapter)

{

wave\_num = 0;}

void engines(templet::engine&e) {

templet::actor::engine(e);}

void start() {

on\_out(out);}

inline void on\_out(border&m) {

if(++wave\_num <= T) out.send();}

border out;

int wave\_num;};

#pragma templet Stage(in?border,out!border)

struct Stage :public templet::actor {

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

((Stage\*)a)->on\_in(\*(border\*)m);}

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

((Stage\*)a)->on\_out(\*(border\*)m);}

Stage(templet::engine&e) :Stage() {

Stage::engines(e);}

Stage() :templet::actor(false),

out(this, &on\_out\_adapter)

{

\_in = 0;}

void engines(templet::engine&e) {

templet::actor::engine(e);}

inline void on\_in(border&m) {

\_in = &m;

calculate();}

inline void on\_out(border&m) {

calculate();}

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

border out;

void calculate(){

if(access(\_in) && access(out)){

for(int j=1; j<NY+1; j++) {

double left = (offset-1 == 0) ? 100.0 : area[offset-1][j];

double right = (offset+1 == NX+1) ? 0.0 : area[offset+1][j];

double bottom = (j == 1) ? 100.0 : area[offset][j-1];

double top = (j == NY) ? 0.0 : area[offset][j+1];

area[offset][j] = 0.25\*(left + right + bottom + top);}

\_in->send(); out.send();}

}

border\* \_in;

int offset;};

#pragma templet End(in?border)

struct End :public templet::actor {

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

((End\*)a)->on\_in(\*(border\*)m);}

End(templet::engine&e) :End() {

End::engines(e);}

End() :templet::actor(false),

t(this, &on\_t\_adapter){

wave\_num = 0;}

void engines(templet::engine&e) {

templet::actor::engine(e);}

inline void on\_in(border&m) {

if(++wave\_num == T) stop();

else m.send();}

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

int wave\_num;};

int main (){

for(int j=0; j<NY+2; j++) {

area[0][j] = 100.0;

area[NX+1][j] = 0.0;

if(j==0 || j==NY+1) {

for(int i=0; i<NX+2; i++) {

area[i][j] = (j==0) ? 100.0 : 0.0;}

}

}

engine eng;

Begin b(eng);

Stage s[NX];

End e(eng);

border\* last\_out = &b.out;

e.in(s[NX-1].out);

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

s[i].engines(eng);

s[i].offset = i+1;

s[i].in(\*last\_out);

last\_out = &s[i].out;}

eng.start();

for(int j = 0; j < NY+2; j++) {

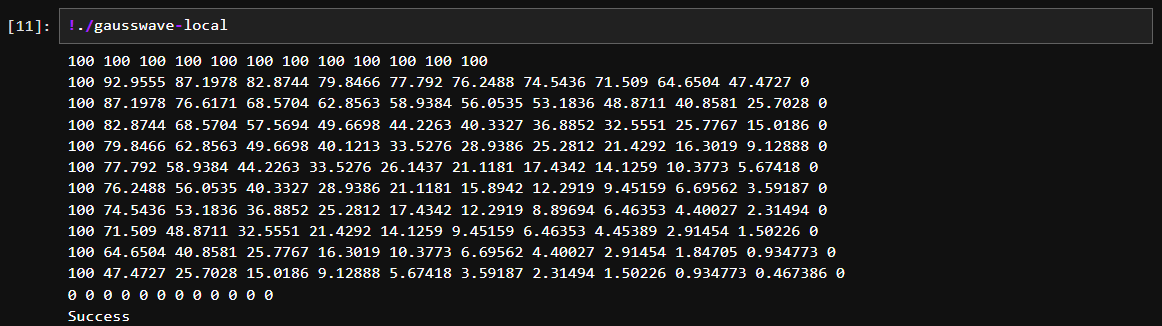
for(int i = 0; i < NX+2; i++) {

cout << area[i][j] << " ";}

cout << endl;}

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

Результат работы представлен на рисунке 2.

  
Рисунок 2 – Результат работы программы

1. Второй этап – выделить в коде задачи, задать длительность их исполнения в единицах модельного времени, исследовать параллелизм

В программе выполняются три основные задачи – инициализация и передача данных двумерного массива, непосредственно волновой расчет и фиксация итогового состояния распределения по двумерной сетке. Так как все задачи являются последовательными наблюдать каких-либо признаков параллелизма мы не можем. Убедиться в этом можно при помощи имитационного движка Basesim\_engine.

Ниже описан код с выделением задачи для исходного алгоритма:

#define \_CRT\_SECURE\_NO\_DEPRECATE

#include <templet.hpp>

#include <basesim.hpp>

#include <iostream>

using namespace templet;

using namespace std;

inline constexpr int NX = 10;

inline constexpr int NY = 10;

inline constexpr int T = 10;

inline double area[NX+2][NY+2] = {};

class border : public templet::message {

public:

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

templet::message(a, ma) {}

};

/\*$TET$\*/

#pragma templet !Begin(out!border)

struct Begin :public templet::actor {

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

((Begin\*)a)->on\_out(\*(border\*)m);}

Begin(templet::engine&e) :Begin() {

Begin::engines(e);

}

Begin() :templet::actor(true),

out(this, &on\_out\_adapter)

{

/\*$TET$Begin$Begin\*/

wave\_num = 0;

/\*$TET$\*/

}

void engines(templet::engine&e) {

templet::actor::engine(e);

/\*$TET$Begin$engines\*/

/\*$TET$\*/

}

void start() {

/\*$TET$Begin$start\*/

on\_out(out);

/\*$TET$\*/

}

inline void on\_out(border&m) {

/\*$TET$Begin$out\*/

if(++wave\_num <= T) out.send();

/\*$TET$\*/

}

border out;

/\*$TET$Begin$$footer\*/

int wave\_num;

/\*$TET$\*/

};

#pragma templet Stage(in?border,out!border)

struct Stage :public templet::actor {

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

((Stage\*)a)->on\_in(\*(border\*)m);}

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

((Stage\*)a)->on\_out(\*(border\*)m);}

Stage(templet::engine&e) :Stage() {

Stage::engines(e);

}

Stage() :templet::actor(false),

out(this, &on\_out\_adapter)

{

/\*$TET$Stage$Stage\*/

\_in = 0;

/\*$TET$\*/

}

void engines(templet::engine&e) {

templet::actor::engine(e);

/\*$TET$Stage$engines\*/

/\*$TET$\*/

}

inline void on\_in(border&m) {

/\*$TET$Stage$in\*/

\_in = &m;

calculate();

/\*$TET$\*/

}

inline void on\_out(border&m) {

/\*$TET$Stage$out\*/

calculate();

/\*$TET$\*/

}

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

border out;

/\*$TET$Stage$$footer\*/

void calculate(){

if(access(\_in) && access(out)){

for(int j=1; j<NY+1; j++) {

double left = (offset-1 == 0) ? 100.0 : area[offset-1][j];

double right = (offset+1 == NX+1) ? 0.0 : area[offset+1][j];

double bottom = (j == 1) ? 100.0 : area[offset][j-1];

double top = (j == NY) ? 0.0 : area[offset][j+1];

area[offset][j] = 0.25\*(left + right + bottom + top);

}

\_in->send(); out.send();

}

}

border\* \_in;

int offset;

/\*$TET$\*/

};

#pragma templet End(in?border,t:basesim)

struct End :public templet::actor {

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

((End\*)a)->on\_in(\*(border\*)m);}

static void on\_t\_adapter(templet::actor\*a, templet::task\*t) {

((End\*)a)->on\_t(\*(templet::basesim\_task\*)t);}

End(templet::engine&e,templet::basesim\_engine&te\_basesim) :End() {

End::engines(e,te\_basesim);

}

End() :templet::actor(false),

t(this, &on\_t\_adapter)

{

/\*$TET$End$End\*/

wave\_num = 0;

/\*$TET$\*/

}

void engines(templet::engine&e,templet::basesim\_engine&te\_basesim) {

templet::actor::engine(e);

t.engine(te\_basesim);

/\*$TET$End$engines\*/

/\*$TET$\*/

}

inline void on\_in(border&m) {

/\*$TET$End$in\*/

\_m = &m;

if(++wave\_num == T) stop();

t.submit();

/\*$TET$\*/

}

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

/\*$TET$End$t\*/

t.delay(10);

\_m->send();

/\*$TET$\*/

}

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

templet::basesim\_task t;

/\*$TET$End$$footer\*/

int wave\_num;

message\* \_m;

/\*$TET$\*/

};

/\*$TET$$footer\*/

int main (){

for(int j=0; j<NY+2; j++) {

area[0][j] = 100.0;

area[NX+1][j] = 0.0;

if(j==0 || j==NY+1) {

for(int i=0; i<NX+2; i++) {

area[i][j] = (j==0) ? 100.0 : 0.0;

}

}

}

engine eng;

basesim\_engine teng;

Begin b(eng);

Stage s[NX];

End e(eng, teng);

border\* last\_out = &b.out;

e.in(s[NX-1].out);

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

s[i].engines(eng);

s[i].offset = i+1;

s[i].in(\*last\_out);

last\_out = &s[i].out;

}

eng.start();

teng.run();

for(int j = 0; j < NY+2; j++) {

for(int i = 0; i < NX+2; i++) {

cout << area[i][j] << " ";

}

cout << endl;

}

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

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

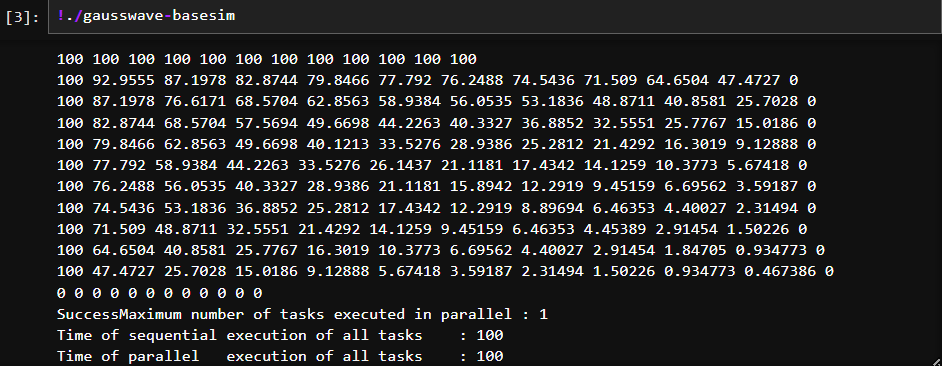
std::cout << "Time of sequential execution of all tasks : " << teng.T1() << std::endl;

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

}

/\*$TET$\*/

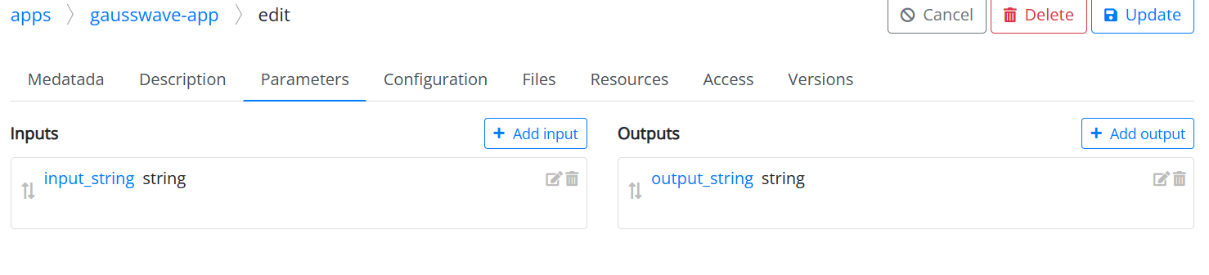
Результат работы представлен на рисунке 3.

  
Рисунок 3 – Результат работы программы

1. Третий этап – реализовать распределенные вычисления задач в облаке с использованием платформы Everest

Для распределения задач в системе Everest был добавлен промежуточный актор RemoteActor, заменяющий первый StageActor. При получении сообщения, актор BeginActor отправляет в RemoteActor начало волны, где вычисления первой итерации проводятся в созданном приложении в системе Everest. В приложении происходят вычисления первой волны и отправка результатов обратно. Далее результаты отправляются из RemoteActor в StageActor, где вычисляются оставшиеся итерации уже в основной системе. Для Everest создано и загружено отдельное приложение для вычисления первой волны.

Скриншоты созданного в системе Everest приложения представлены на рисунках 4-7.

  
Рисунок 4 – Входные и выходные параметры

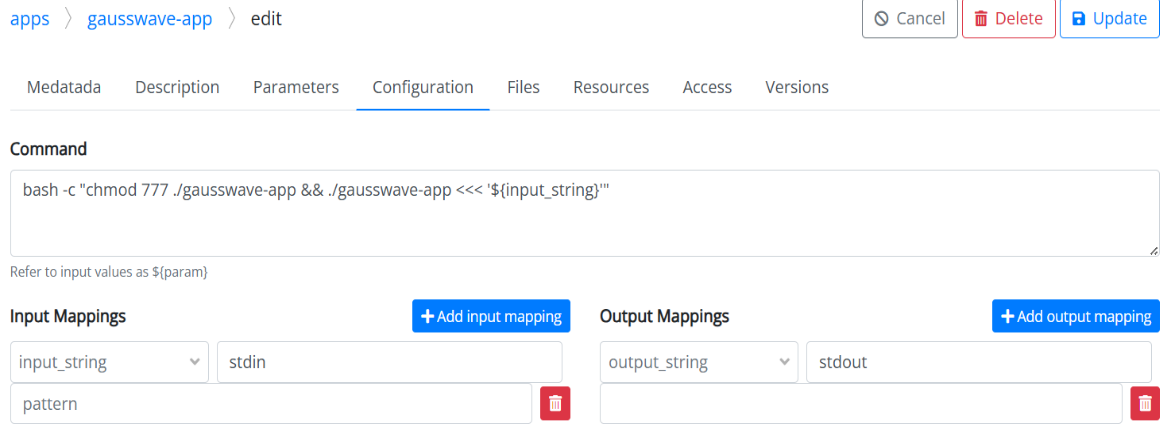
  
Рисунок 5 – Конфигурация приложения

  
Рисунок 6 – Входные данные для вычисления в Job

  
Рисунок 7 – Выходные данные после вычисления в Job

Код приложения для вычисления на Everest представлен ниже.

**#include <iostream>**

**#include <sstream>**

**#include <string>**

**#include <vector>**

**using namespace std;**

**int main() {**

**string data;**

**getline(cin, data);**

**istringstream ss(data);**

**int NX, NY;**

**ss >> NX >> NY;**

**vector<vector<double>> area(NX + 2, vector<double>(NY + 2));**

**for (int i = 0; i < NX + 2; ++i)**

**for (int j = 0; j < NY + 2; ++j)**

**ss >> area[i][j];**

**vector<vector<double>> new\_area = area;**

**for (int i = 1; i <= NX; ++i) {**

**for (int j = 1; j <= NY; ++j) {**

**double L = area[i][j - 1];**

**double R = area[i][j + 1];**

**double U = area[i - 1][j];**

**double D = area[i + 1][j];**

**new\_area[i][j] = 0.25 \* (L + R + U + D);**

**}**

**}**

**ostringstream out\_ss;**

**out\_ss << NX << " " << NY << " ";**

**for (int i = 0; i < NX + 2; ++i)**

**for (int j = 0; j < NY + 2; ++j)**

**out\_ss << new\_area[i][j] << " ";**

**string output\_string = out\_ss.str();**

**cout << output\_string << endl;**

**}**

Код основного алгоритма представлен ниже.

**#define \_CRT\_SECURE\_NO\_DEPRECATE**

**#include <templet.hpp>**

**#include <everest.hpp>**

**#include <sstream>**

**#include <iostream>**

**#include <chrono>**

**using namespace templet;**

**using namespace std;**

**inline constexpr int NX = 10;**

**inline constexpr int NY = 10;**

**inline constexpr int T = 10;**

**inline double area[NX+2][NY+2] = {};**

**class border : public templet::message {**

**public:**

**border(actor\* a=nullptr, message\_adaptor ma=0)**

**: templet::message(a,ma) {}**

**string payload;**

**};**

**/\*$TET$\*/**

**#pragma templet !Begin(out!border)**

**struct Begin : public templet::actor {**

**int wave\_num;**

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

**((Begin\*)a)->on\_out(\*(border\*)m);**

**}**

**Begin(templet::engine& e) : Begin() { Begin::engines(e); }**

**Begin() : templet::actor(true), out(this,&on\_out\_adapter) {**

**wave\_num = 0;**

**}**

**void engines(templet::engine& e) {**

**templet::actor::engine(e);**

**}**

**void start() {**

**on\_out(out);**

**}**

**inline void on\_out(border& m) {**

**if (++wave\_num <= 1) {**

**ostringstream ss;**

**ss << NX << " " << NY << " ";**

**for (int i = 0; i < NX+2; ++i)**

**for (int j = 0; j < NY+2; ++j)**

**ss << area[i][j] << " ";**

**m.payload = ss.str();**

**out.send();**

**}**

**}**

**border out;**

**};**

**/\*$TET$\*/**

**#pragma templet Remote(in?border,out!border,t:everest)**

**struct Remote : public templet::actor {**

**border\* \_in;**

**std::chrono::high\_resolution\_clock::time\_point start\_time;**

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

**((Remote\*)a)->on\_in(\*(border\*)m);**

**}**

**static void on\_t\_adapter(templet::actor\* a, templet::task\* t) {**

**((Remote\*)a)->on\_t(\*(everest\_task\*)t);**

**}**

**Remote(templet::engine& e, templet::everest\_engine& ev) : Remote() {**

**Remote::engines(e,ev);**

**}**

**Remote() : templet::actor(false),**

**out(this,&on\_out\_adapter),**

**t(this,&on\_t\_adapter) {**

**t.app\_id("682e3b721200001100ee4b84");**

**}**

**void engines(templet::engine& e, templet::everest\_engine& ev) {**

**templet::actor::engine(e);**

**t.engine(ev);**

**}**

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

**inline void on\_in(border& m) {**

**\_in = &m;**

**json in;**

**in["name"] = "gausswave-app";**

**in["inputs"]["input\_string"] = m.payload;**

**start\_time = std::chrono::high\_resolution\_clock::now();**

**if (t.submit(in)) cout<<"task submit succeeded\n";**

**else cout<<"task submit failed\n";**

**}**

**inline void on\_t(everest\_task& task) {**

**auto end\_time = std::chrono::high\_resolution\_clock::now();**

**auto duration = std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time);**

**std::cout << "Remote processing time: " << duration.count() << " ms\n";**

**json out\_j = task.result();**

**string data = out\_j["output\_string"].get<string>();**

**\_in->payload = data;**

**istringstream ss(data);**

**int nx\_tmp, ny\_tmp;**

**ss >> nx\_tmp >> ny\_tmp;**

**for (int i = 0; i < NX+2; ++i)**

**for (int j = 0; j < NY+2; ++j)**

**ss >> area[i][j];**

**cout<<"Wave 1 received from Everest\n";**

**out.payload = data;**

**out.send();**

**}**

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

**((Remote\*)a)->on\_out(\*(border\*)m);**

**}**

**void on\_out(border&) {}**

**templet::everest\_task t;**

**border out;**

**};**

**/\*$TET$\*/**

**#pragma templet Stage(in?border,out!border)**

**struct Stage : public templet::actor {**

**std::chrono::high\_resolution\_clock::time\_point start\_time;**

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

**((Stage\*)a)->on\_in(\*(border\*)m);**

**}**

**Stage(templet::engine& e) : Stage() { Stage::engines(e); }**

**Stage() : templet::actor(false), out(this,&on\_out\_adapter) {}**

**void engines(templet::engine& e) {**

**templet::actor::engine(e);**

**}**

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

**inline void on\_in(border&) {**

**start\_time = std::chrono::high\_resolution\_clock::now();**

**for (int step = 2; step <= T; ++step) {**

**double new\_area[NX+2][NY+2] = {};**

**for (int i = 1; i <= NX; ++i)**

**for (int j = 1; j <= NY; ++j)**

**new\_area[i][j] = 0.25 \* (**

**area[i+1][j] +**

**area[i-1][j] +**

**area[i][j+1] +**

**area[i][j-1]**

**);**

**for (int i = 1; i <= NX; ++i)**

**for (int j = 1; j <= NY; ++j)**

**area[i][j] = new\_area[i][j];**

**cout<<"Wave "<<step<<" processed by Stage\n";**

**}**

**ostringstream ss;**

**ss<<NX<<" "<<NY<<" ";**

**for (int i = 0; i < NX+2; ++i)**

**for (int j = 0; j < NY+2; ++j)**

**ss<<area[i][j]<<" ";**

**out.payload = ss.str();**

**auto end\_time = std::chrono::high\_resolution\_clock::now();**

**auto duration = std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time);**

**std::cout << "Stage processing time: " << duration.count() << " ms\n";**

**out.send();**

**}**

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

**((Stage\*)a)->on\_out(\*(border\*)m);**

**}**

**void on\_out(border&) {}**

**border out;**

**};**

**/\*$TET$\*/**

**#pragma templet End(in?border)**

**struct End : public templet::actor {**

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

**((End\*)a)->on\_in(\*(border\*)m);**

**}**

**End(templet::engine& e) : End() { End::engines(e); }**

**End() : templet::actor(false) {}**

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

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

**inline void on\_in(border&) {**

**cout<<"End received message, stopping engine\n";**

**stop();**

**}**

**};**

**/\*$TET$$footer\*/**

**int main(){**

**for(int j=0;j<NY+2;++j){**

**area[0][j] = 100;**

**area[NX+1][j] = 0;**

**if(j==0||j==NY+1)**

**for(int i=0;i<NX+2;++i)**

**area[i][j] = (j==0?100:0);**

**}**

**templet::engine eng;**

**templet::everest\_engine teng("g64cvbr3hzpfli1de47hl9fzr003yj2y4ldbsy6xemfvwcukk1peub94wpjkdzmb");**

**if(!teng){**

**cout<<"Everest engine not connected\n";**

**return EXIT\_FAILURE;**

**}**

**Begin b(eng);**

**Remote r(eng,teng);**

**Stage s(eng);**

**End e(eng);**

**b.out.bind(&r,&Remote::on\_in\_adapter);**

**r.out.bind(&s,&Stage::on\_in\_adapter);**

**s.out.bind(&e,&End::on\_in\_adapter);**

**b.start();**

**eng.start();**

**try\_continue:**

**teng.run();**

**if (eng.stopped()) {**

**cout << "\nFinal area:\n";**

**for (int j = 0; j < NY + 2; ++j) {**

**for (int i = 0; i < NX + 2; ++i)**

**cout << area[i][j] << " ";**

**cout << "\n";**

**}**

**cout << "Success" << endl;**

**return EXIT\_SUCCESS;**

**}**

**static int recovery\_tries = 3;**

**templet::everest\_error cntxt;**

**if (teng.error(&cntxt)) {**

**std::cout << "...task engine error..." << std::endl;**

**if (recovery\_tries--) {**

**std::cout << "error information:" << std::endl;**

**std::cout << "type ID : " << \*cntxt.\_type << std::endl;**

**std::cout << "HTML response code : " << cntxt.\_code << std::endl;**

**std::cout << "HTML response : " << cntxt.\_response << std::endl;**

**std::cout << "task input : " << cntxt.\_task\_input << std::endl;**

**switch (\*cntxt.\_type) {**

**case templet::everest\_error::NOT\_CONNECTED:**

**{**

**std::cout << "error string : NOT\_CONNECTED" << std::endl;**

**std::cout << "the task engine is not initialized properly -- fatal error, exiting" << std::endl;**

**return EXIT\_FAILURE;**

**}**

**case templet::everest\_error::SUBMIT\_FAILED:**

**{**

**std::cout << "error string : SUBMIT\_FAILED" << std::endl;**

**std::cout << "resubmitting the task" << std::endl;**

**json input = json::parse(cntxt.\_task\_input);**

**cntxt.\_task->resubmit(input);**

**break;**

**}**

**case templet::everest\_error::TASK\_CHECK\_FAILED:**

**{**

**std::cout << "error string : TASK\_CHECK\_FAILED" << std::endl;**

**std::cout << "trying to check the task status again" << std::endl;**

**\*cntxt.\_type = templet::everest\_error::NOT\_ERROR;**

**break;**

**}**

**case templet::everest\_error::TASK\_FAILED\_OR\_CANCELLED:**

**{**

**std::cout << "error string : TASK\_FAILED\_OR\_CANCELLED" << std::endl;**

**std::cout << "resubmitting the task" << std::endl;**

**json input = json::parse(cntxt.\_task\_input);**

**cntxt.\_task->resubmit(input);**

**}**

**}**

**goto try\_continue;**

**}**

**}**

**else**

**std::cout << "logical error" << std::endl;**

**return EXIT\_FAILURE;**

**}**

**/\*$TET$\*/**

**Заключение**

В результате выполнения работы мы переработали изначальный алгоритм для одномерного массива в алгоритм для двумерного; провели структуризацию и вычисления с помощью акторного подхода на Templet SDK; исследовали подход с параллельными и распределёнными вычислениями при помощи имитационного моделирования; реализовали приложение и провели распределённые вычисления в облачной системе Everest.

**Список использованных источников**

1 С.В. Востокин «Метод создания параллельных и распределенных программ в парадигме акторов» 2023. – С. 53-61.

2 Документация по С++ [Электронный ресурс]. URL: [https://learn.microsoft.com/ru-ru/cpp/cpp/](https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170) (дата обращения: 15.04.2025).

3 Документация Everest [Электронный ресурс]. URL: http://everest.distcomp.org/docs/tutorial/ (дата обращения: 10.05.2025).