

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА. ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНЫХ СИСТЕМ

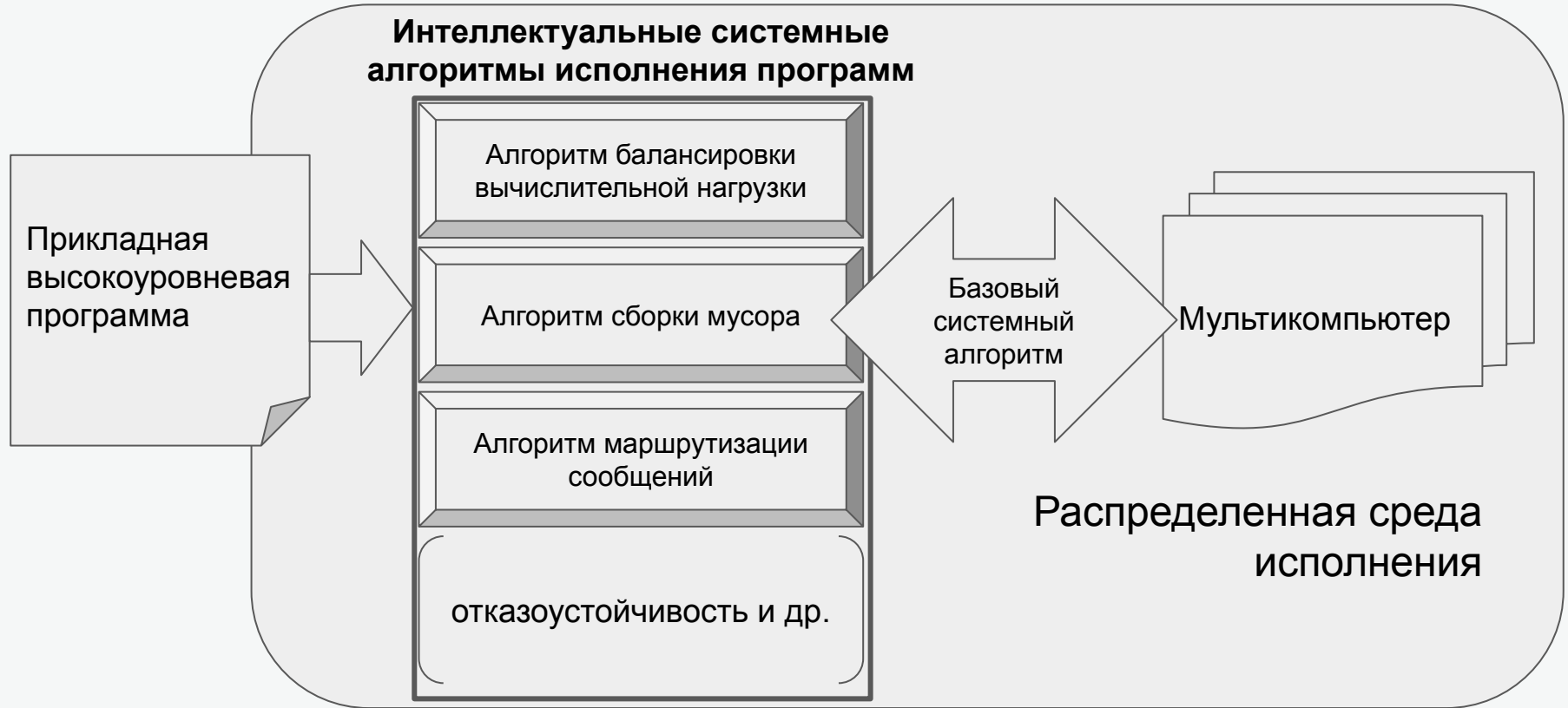
Разработка и реализация переносимых алгоритмов распределенного исполнения фрагментированных программ

Выполнил: **Ажбаков Артем Альбертович**

Научный руководитель: **Малышкин Виктор Эммануилович**
д.т.н., зав. кафедрой, кафедра ПВ ФИТ

Рецензент: **Загорулько Юрий Алексеевич**
к.т.н., зав. лабораторией ИСИ СО РАН

Системы автоматизации параллельного программирования упрощают разработку прикладных программ



Разработка системных алгоритмов исполнения параллельных программ является трудоемкой

- Вычислительные кластера пополняются разнородным оборудованием
- Реализуются перспективы объединения вычислительных сетей

Тенденция роста неоднородности вычислительных ресурсов **увеличивает роль испытаний** системных алгоритмов автоматизации параллельного исполнения **на различных конфигурациях неоднородного оборудования.**

Имеется **потребность в программной платформе**, облегчающей проведение экспериментальных исследований системных алгоритмов распределенного исполнения параллельных программ в неоднородной среде.

От программной платформы требуется:

- возможность исполнения прикладных программ на мультикомпьютерах с высокой степенью неоднородности
- возможность замены используемых системных алгоритмов на исследуемые (с целью проведения экспериментов)

Обзор существующих систем параллельного программирования

Для использования в качестве платформы для экспериментальных исследований системных алгоритмов исполнения параллельных программ важны **характеристики**:

1. Модель вычислений
2. Переносимость
3. Модульность программной архитектуры

Общие недостатки:
закрытость, недостаточная
практическая переносимость

Charm++	+ фрагментирование, + множество системных алгоритмов - закрытая программная архитектура, - фиксация управления вычислительными ресурсами в прикладной программе
PaRSEC	+ гибридное управление вычислениями, - неуниверсальный язык
Legion	+ независимость прикладной задачи и управления, + высокая параметризуемость + многоуровневая программная архитектура
LuNA	+ независимость прикладной задачи и управления, + единственное присваивание

Цель работы

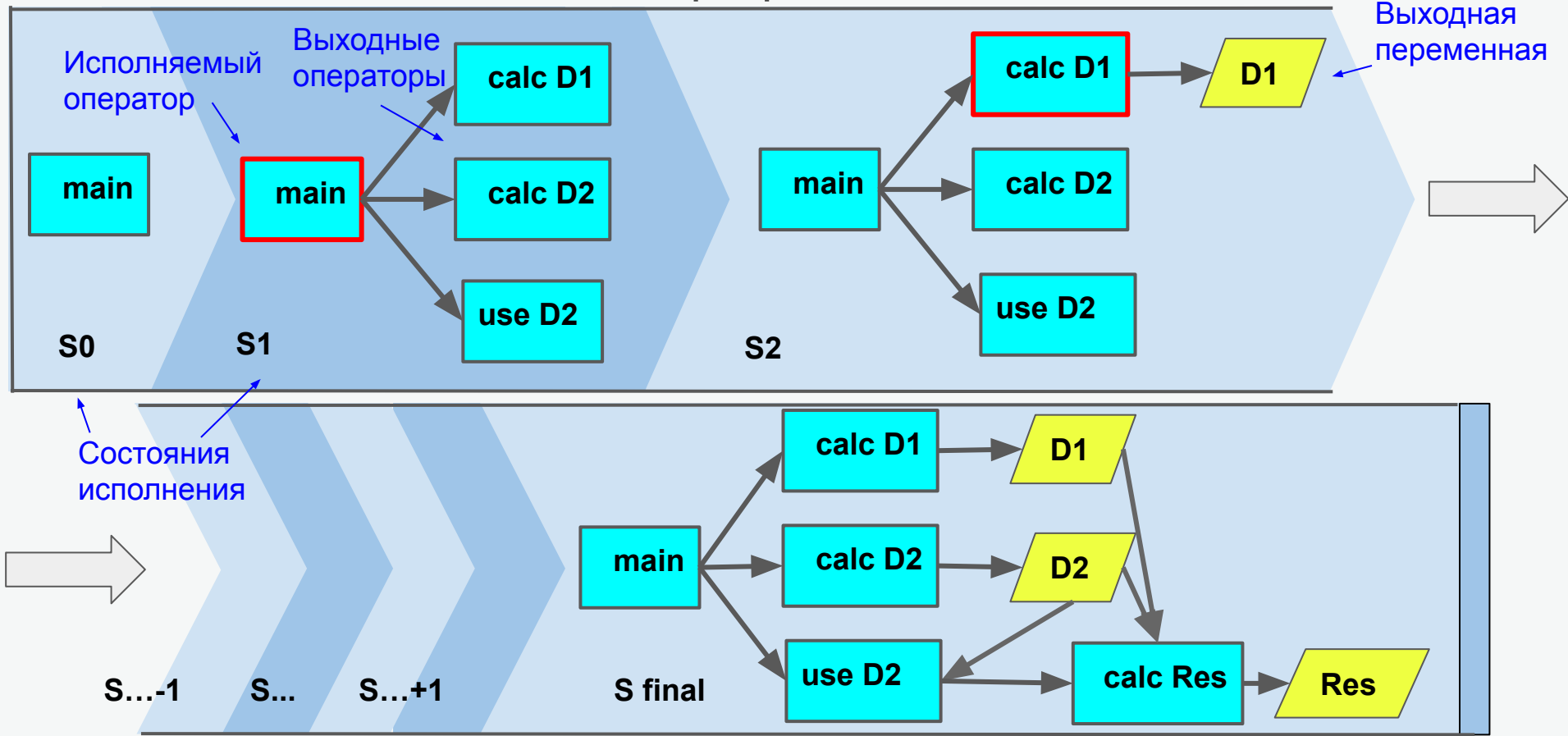
- *разработка переносимой распределенной среды исполнения для системы фрагментированного программирования задач численного моделирования LuNA.*

Требования: переносимость, модульность, масштабируемость.

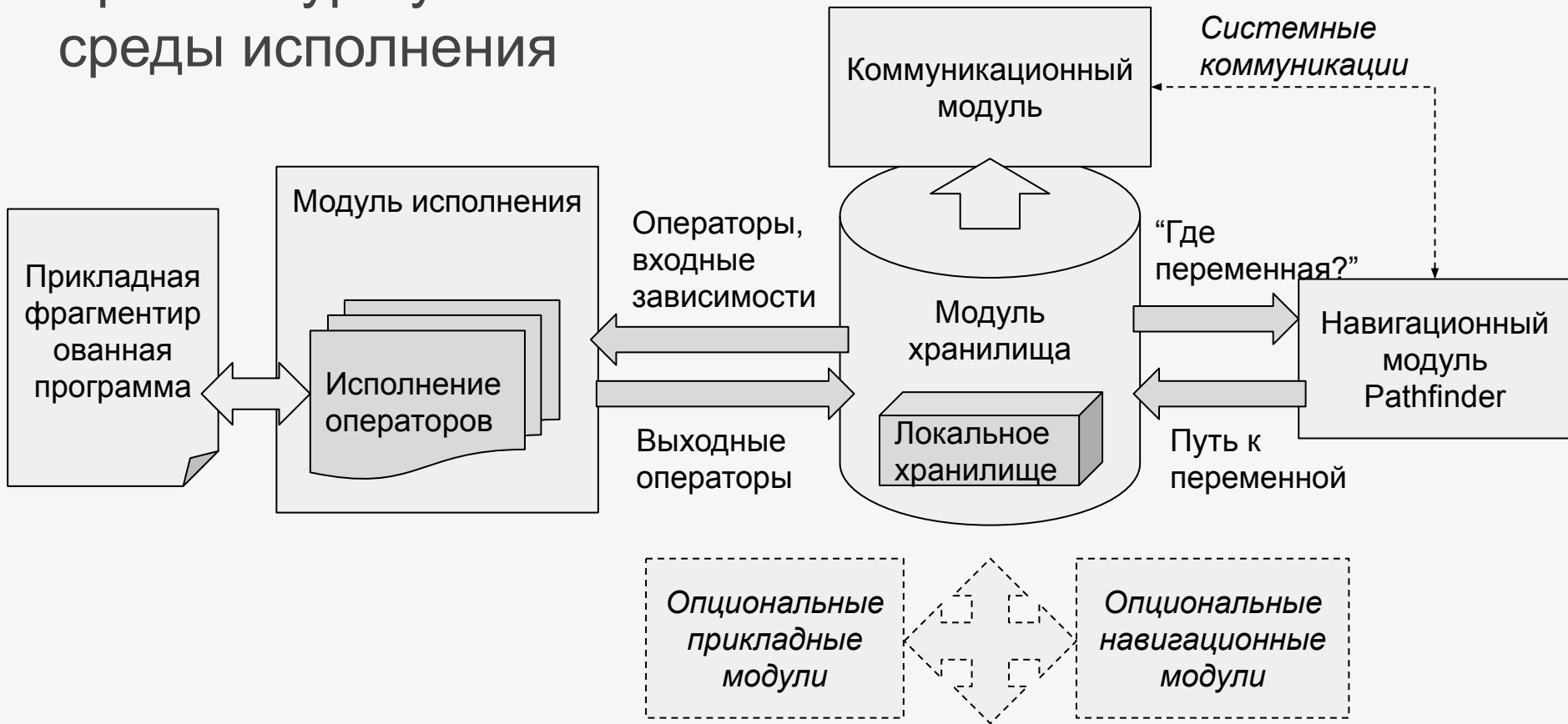
Задачи:

- Сформулировать модель исполнения LuNA-программы;
- Проанализировать модель исполнения с целью выявления функций среды исполнения LuNA и их особенностей;
- Разработать модульную архитектуру среды исполнения и алгоритм ее работы;
- Реализовать программно в соответствии с требованиями.

Модель исполнения LuNA-программы



Архитектура узла среды исполнения



Программная реализация

JS

Сервер

- Выдача страниц
- Создание/настройка вычислительных задач
- Проектирование представления для загружаемых топологий
- Связывание узлов (signaling)
- Загрузка и хранение пользовательских ресурсов, доступ к ним

Клиентское приложение

- Загрузка файлов на сервер
- Создание, конфигурирование, подключение к вычислительной задаче
- Связывание узлов (установление p2p-связи)
- Исполнение LuNA-программ
- Пользовательский интерфейс (разметка LuNA-программы, просмотр LuNA-операторов, состояние соединения с соседями, управление исполнением)

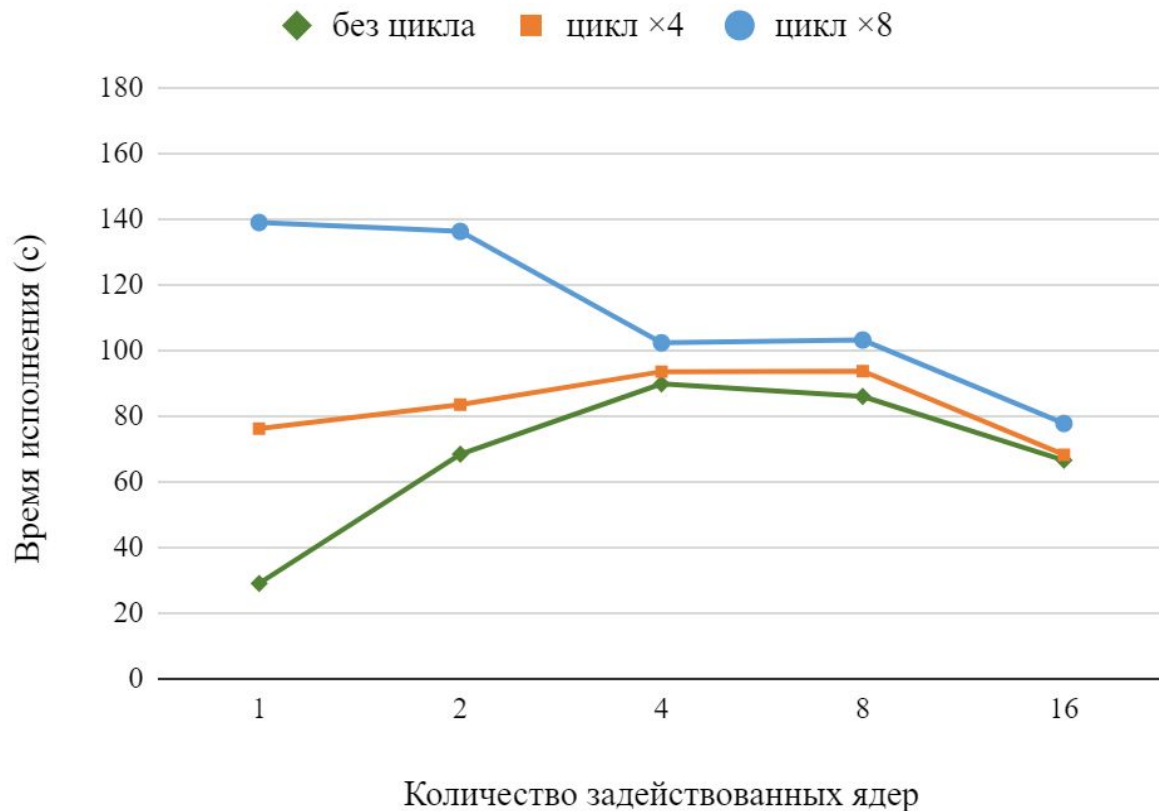
Тестирование

Тест 1: решение уравнения Пуассона методом Якоби в трехмерной сетке размером 200x200x200, 85 итераций

Топология сети: одномерная решетка

Intel Core i5-7600K (4 core, 3.8 Ghz), 8Gb RAM

Тест 2: тестирование переносимости (Windows ПК, ноутбук с Ubuntu, 3 смартфона Android)



Заключение, защищаемые положения:

Создана программная платформа для исследования системных алгоритмов распределенного исполнения параллельных программ.

- Сформулирована модель исполнения фрагментированной программы;
- Определены функции среды исполнения;
- Разработана архитектура среды исполнения и переносимый алгоритм ее работы;
- Реализована переносимая среда исполнения;
- Выполнено тестирование, разработанная среда исполнения способна распределенно исполнять LuNA-программы, замена модулей позволяет управлять исполнением параллельной программы

Планы:

- Проводить исследования системных алгоритмов исполнения параллельных программ с использованием системы;
- Улучшение программной платформы: отладка и профилирование, повышение удобства использования

Спасибо за внимание

Ажбаков Артем Альбертович

Node.js vs Java

<https://benchmarkgame-team.pages.debian.net/benchmarkgame/faster/javascript.html>

regex-redux

source	secs	mem	gz	cpu	cpu load			
<u>Node js</u>	12.08	850,588	408	12.78	7%	24%	33%	44%
<u>Java</u>	10.34	645,952	740	30.60	70%	77%	78%	73%

n-body

source	secs	mem	gz	cpu	cpu load			
<u>Node js</u>	26.30	33,604	1297	26.30	0%	100%	3%	0%
<u>Java</u>	21.94	35,852	1429	21.99	0%	100%	2%	1%

mandelbrot

source	secs	mem	gz	cpu	cpu load			
<u>Node js</u>	18.11	621,468	748	65.41	95%	87%	97%	83%
<u>Java</u>	5.96	79,316	796	23.34	97%	98%	98%	99%

reverse-complement

source	secs	mem	gz	cpu	cpu load			
<u>Node js</u>	17.02	1,136,952	1103	18.52	14%	52%	35%	8%
<u>Java</u>	3.25	701,084	2183	7.42	51%	78%	46%	56%

fannkuch-redux

source	secs	mem	gz	cpu	cpu load			
<u>Node js</u>	81.12	31,040	473	81.10	0%	1%	100%	0%
<u>Java</u>	14.30	34,412	1282	56.26	99%	98%	99%	97%

binary-trees

source	secs	mem	gz	cpu	cpu load			
<u>Node js</u>	48.04	597,244	434	91.26	49%	44%	55%	43%
<u>Java</u>	8.33	985,604	835	27.40	89%	77%	81%	87%

k-nucleotide

source	secs	mem	gz	cpu	cpu load			
<u>Node js</u>	64.39	1,865,484	935	135.40	68%	66%	77%	96%
<u>Java</u>	8.77	465,584	1812	27.40	73%	72%	70%	98%

Функции среды исполнения

1. Выбор топологии вычислительной сети;
2. Выбор коммуникационных протоколов и технологий передачи данных между узлами вычислительной сети;
3. Начальное расположение оператора вызова подпрограммы main;
4. Выбор узлов мультимпьютера для новых операторов и фрагментов данных:
 - 4.1. Использовать ли системные коммуникации для получения информации о состоянии окружающих узлов при выборе узла-получателя;
 - 4.2. Выбор узлов, с которыми осуществляются системные коммуникации;
 - 4.3. Организация кэширования фрагментов данных на нескольких узлах мультимпьютера;
5. Выбор алгоритма сборки мусора (освобождение памяти использованных фрагментов данных):
 - 5.1. Алгоритм распределенной сборки мусора при использовании кэширования фрагментов данных, протокол сопутствующих системных коммуникаций;
6. При обеспечении отказоустойчивости (логическое замещение вычислительного узла при аварийном отключении):
 - 6.1. Дублирование фрагментов данных и операторов, избыточное вычисление, алгоритм голосования при определении результата избыточного исполнения оператора;
7. При выполнении операторов:
 - 7.1. Способы организации параллельного исполнения операторов в пределах одного узла мультимпьютера;
 - 7.2. Стратегии раскрутки циклов;

React App localhost:5000/lobby

CleanTry1

- Home
- Counter
- Fragment Viewer
- Execution**
- Lobbies
- Tasks
- View Debug
- Exec Debug

Worker

Execute CF

CF storage

```

print_border ( int, int, int, value )

print_info ( 10 : int, 10 : int, 10 : int, 10 : int, 1000 : int ) ⇒ 0.0.main._l64

init_dist ( 10 : int, 10 : int, DF : name, DF : name, DF : name ) ⇒ 0.0.main._l66

for i in ( 0 ... 9 ) ⇒ 0.0.main.for.0

set_real ( DF : name, 1 : real ) ⇒ 0.0.main._l71

while &&
print_int ( DF : int ) ⇒ 0.0.main._l79

find_maxdiff ( DF : it: int, DF : dim: name, DF : sft: name, DF : FF: name, DF : mx)

```

Expression
Action: int
Value: *not ready*

Type Cast
Cast: >
Value: *not ready*

Expression
Action: DF
Value: *not ready*

Data Fragment
ID: 0.0.main.maxdiff[0]
Value: *not ready*

React App localhost:5000/lobby

CleanTry1

```
print_border(int, int, int, value)
```

Worker

Execute CF

CF storage

Empty

DF storage

0.0 main dt: [Obj]	0.0 main st: [Obj]	0.0 main dm: [Obj]
0.0 main F [0][0]: [Obj]	0.0 main F [0][4]: [Obj]	
0.0 main F [0][7]: [Obj]	0.0 main F [0][0]: [Obj]	
0.0 main F [0][4]: [Obj]	0.0 main F [0][7]: [Obj]	
0.0 main F [0][0]: [Obj]		

1 2 3 ... 197 198 199

Source

```
change_border(int, value, value, value, value, name, int)
ps(int, value, value, value, name, name, name, name, int)
main()
{
  dt dt st dm F mx iter maxdf
  print_info(10, 10, 10, 10, 1000) => _i64
  init_dist(10, 10, dt, st, dm) => _i66
  for i in (0..9)
  {
    init_F(i, 10, 10, st, dm, F[0][0]) => init_f(i)
```

React App localhost:5000/lobby

CleanTry1

Worker

Execute CF

CF storage

Empty

DF storage

0.0 main maxdf [0] 1	0.0 main dt: [Obj]	0.0 main st: [Obj]
0.0 main F [0][4]: [Obj]	0.0 main F [0][7]: [Obj]	
0.0 main F [0][4]: [Obj]	0.0 main F [0][7]: [Obj]	
0.0 main F [0][4]: [Obj]	0.0 main F [0][7]: [Obj]	
0.0 main F [0][4]: [Obj]		

1 2 3 ... 192 193 194

Source

```
change_border(int, value, value, value, value, name, int)
ps(int, value, value, value, name, name, name, name, int)
main()
{
  dt dt st dm F mx iter maxdf
  print_info(10, 10, 10, 10, 1000) => _i64
  init_dist(10, 10, dt, st, dm) => _i66
  for i in (0..9)
  {
    init_F(i, 10, 10, st, dm, F[0][0]) => init_f(i)
```

React App localhost:5000/lobby

CleanTry1

Worker

Execute CF

CF storage

Empty

DF storage

0.0 main F [0][5]: [Obj]	0.0 main F [0][8]: [Obj]
0.0 main F [0][2]: [Obj]	0.0 main F [0][5]: [Obj]
0.0 main F [0][8]: [Obj]	0.0 main F [0][2]: [Obj]
0.0 main F [0][5]: [Obj]	0.0 main F [0][8]: [Obj]
0.0 main maxdf [0] 1	0.0 main _i75 FF [0][2]: [Obj]

1 2 3 4 ... 177 178 179

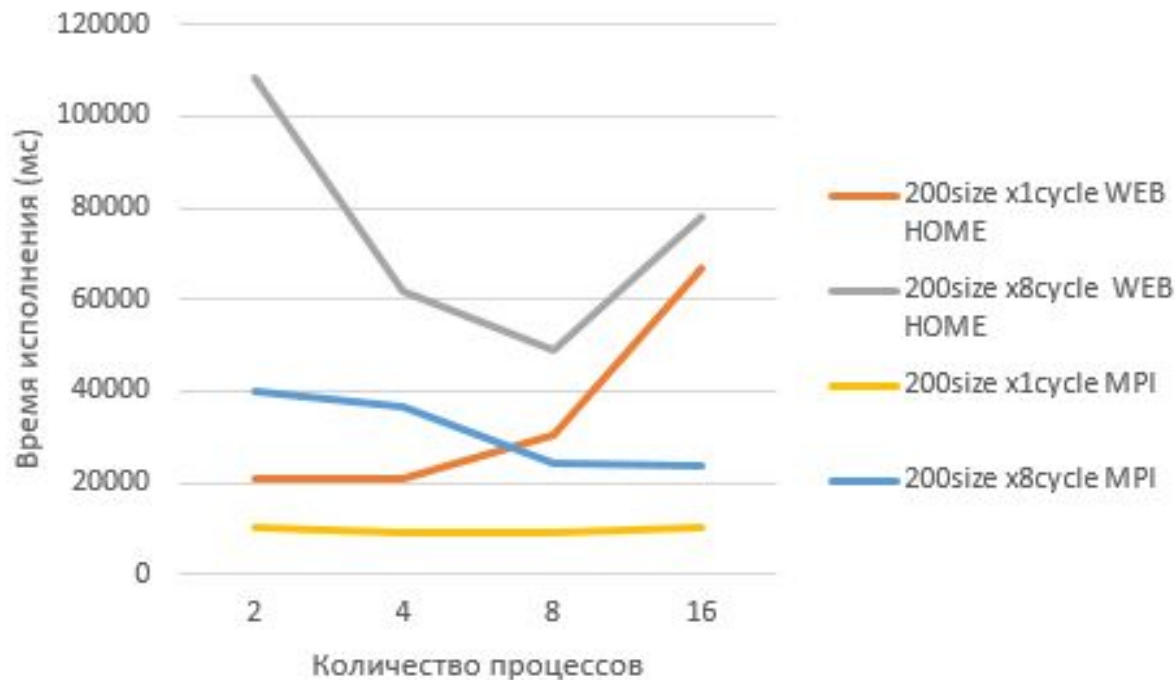
Source

```
change_border(int, value, value, value, value, name, int)
ps(int, value, value, value, name, name, name, name, int)
main()
{
  dt dt st dm F mx iter maxdf
  print_info(10, 10, 10, 10, 1000) => _i64
  init_dist(10, 10, dt, st, dm) => _i66
  for i in (0..9)
  {
    init_F(i, 10, 10, st, dm, F[0][0]) => init_f(i)
```

Тестирование

Тест: решение уравнения Пуассона методом Якоби в трехмерной сетке размером 200x200x200, 85 итераций

Топология сети: одномерная решетка



Web:
Intel Core i5-7600K
(4 core, 3.8 Ghz),
8Gb RAM

MPI:
Intel Core i7-4790K
(4 core, 4.0 Ghz),
32Gb RAM