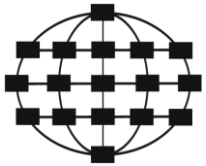


Institute of Computational Mathematics and Mathematical Geophysics SB RAS
Supercomputing Software Department

Language and System LuNA for Automatic Construction of Numerical Distributed Programs

Vladislav Perepelkin

ru-STEP: Russian Seminar on Software Engineering, Theory and Experimental Programming. Novosibirsk & Innopolis — 2021

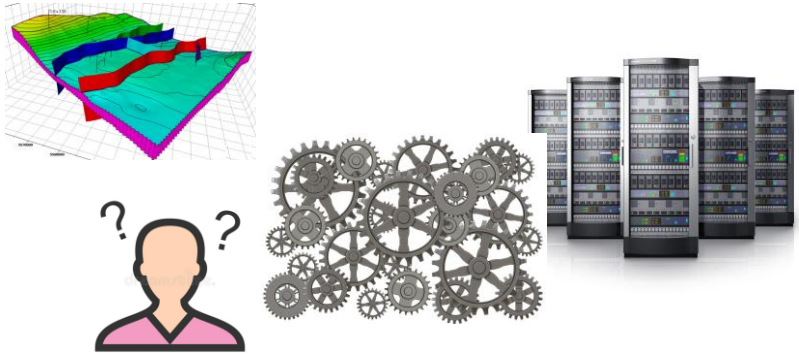
Motivation (1/2)

Development of efficient numerical parallel programs for supercomputers is complex, laborious and requires specialized skills and knowledge.

Parallel programming problems:

- Data and computations decomposition
- Distribution and dynamic redistribution of data and computations to computing nodes
- Management parallel processing of distributed data
- Achieving high performance and scalability

In particular cases dynamic load balancing, fault tolerance, checkpointing, etc. are necessary.



While solving the problems one should consider peculiarities of:

- applied algorithm
- computer
- input data

Factors, which complicate parallel programming further:

- Different configuration of computing nodes
- Usage of GPUs and FPGAs
- High quantity of computing nodes

Motivation (2/2)

Complexities in defelopment of *efficient* parallel programs impede numerical modeling on supercomputers, thus it is important to develop and improve means of automatic construction of parallel programs, based on high level description of the algorithm.

Advantages:

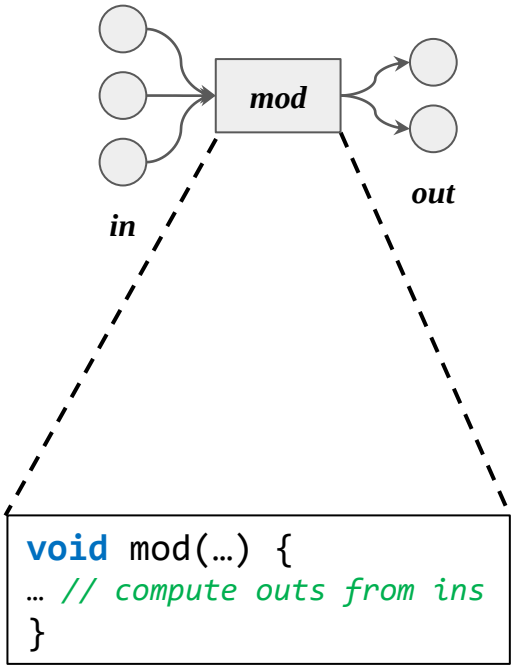
- Reduction of complexity and labor intensity of development, debugging and modification of parallel programs
- Automatic application of accumulated knowledge in the field of methods of parallel implementation of algorithms by including the knowledge into automation means
- Increase of programs portability and life time of software
- Ability to construct programs with different non-functional properties
- In future – improvement of programs' quality as compared to manual programming

Related Works

- Ю.И. Янов, В.Е. Котов, В.А. Вальковский, В.Э. Малышкин
- Язык Утопист (Э.Х. Тыугу, А.Л. Фуксман)
- Система Норма (И.Б. Задыхайло, А.Н. Андрианов)
- DVM-система (В.А. Крюков)
- T-Система (С.М. Абрамов)
- Charm++ (L. Kale)
- PaRSEC и DPLASMA (J. Dongarra)
- Legion и Regent (A. Aiken)

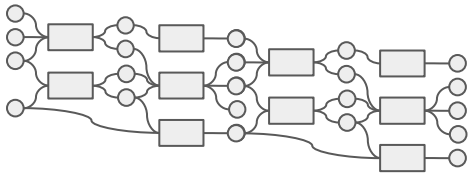
Parallel programming automation is being actively researched for decades in Russia and the world. Many original programming systems have been created and used, but the problem is still open.

The Idea of Fragmented Programming (by Victor Malyshkin)



Idea: to explicitly concern a parallel program as a set of triplets of form ‹in, mod, out›, where in & out are immutable arguments, and mod is a side-effect-free subroutine

Advantage: the ability to automatically construct parallel programs with various non-functional properties



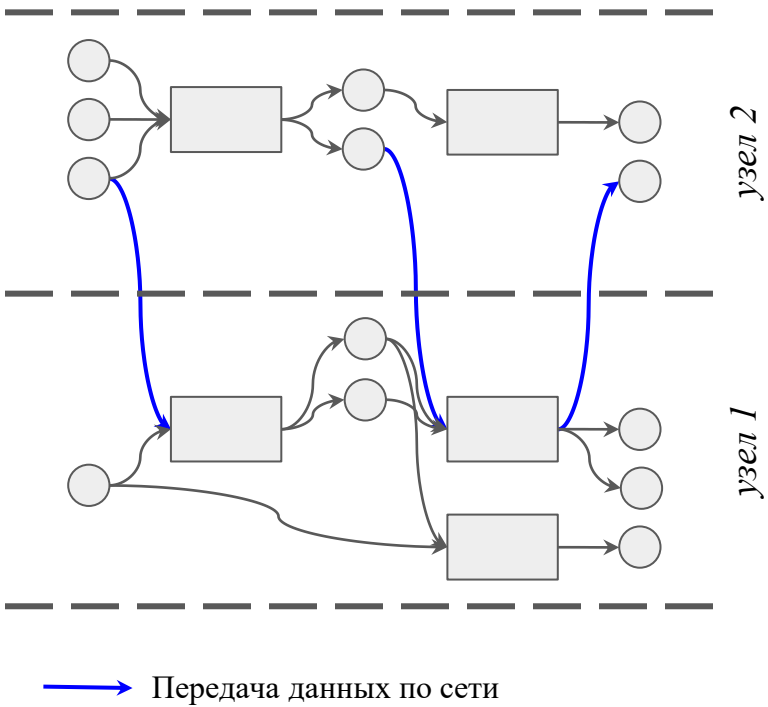
Distributed Execution of a Fragmented Program and Automatic Provision of its Non-Functional Properties

The system automatically distributes triplets to computing nodes and executes them once inputs are computed and available

Scheduling, message passing and garbage collection are performed automatically.

Dynamic load balancing is performed through triplets redistribution.

Checkpointing or fault tolerance automatic support is possible by saving intermediate data.



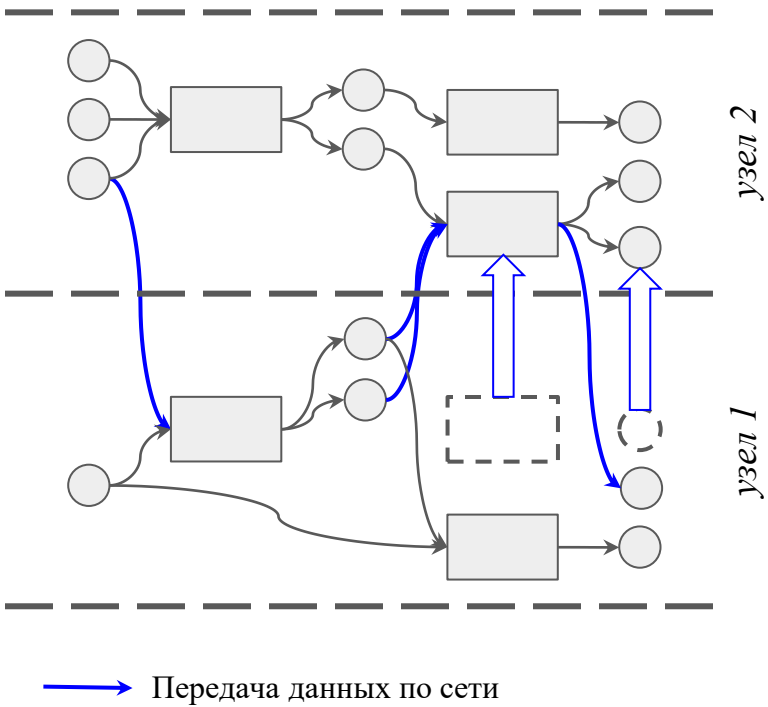
Distributed Execution of a Fragmented Program and Automatic Provision of its Non-Functional Properties

The system automatically distributes triplets to computing nodes and executes them once inputs are computed and available

Scheduling, message passing and garbage collection are performed automatically.

Dynamic load balancing is performed through triplets redistribution.

Checkpointing or fault tolerance automatic support is possible by saving intermediate data.



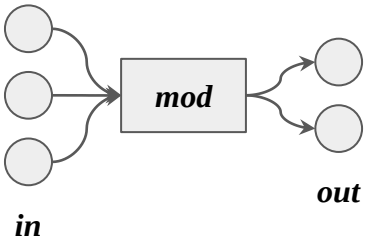
The **Goal** is to develop an experimental system for automatic construction of numerical parallel programs for multicomputers based on the approach

Objectives:

- Development of an input language LuNA
- Development of translation and execution algorithms for LuNA
- Implementation of the developed algorithms as an experimental programming system LuNA
- Experimental investigation of non-functional properties of LuNA and programs it constructs

Theoretical Study

Fragmented Algorithm (FA): an Input Representation of an Algorithm



- Reference expressions (e.g.: x[i][y[j]][z])
- Operator types:
  - Exec (a single triplet specification)
  - Arithmetic loop (a FOR loop analogue)
  - Loop with a precondition (a WHILE loop analogue)

Fragmented algorithm (FA) is a finite set of operators

Computational Fragment (CF) – a computational process, defined by an operator

Data Fragment (DF) – a custom value, addressable by reference expressions

CF takes DFs as input and produces CFs and/or DFs as the result

Execution of FA is the execution of all CFs according to data-flow model

FA is Turing-complete

Main Challenges of Efficient FA Execution

- Choice of CF and DF distribution to computing nodes (static and dynamic)
  - Balancing computing nodes load
  - Reducing network load by taking data locality into account
  - Increasing possible problem size by efficient use of distributed memory
- Choice of CFs execution order (without violating informational dependencies)
  - Reducing amount of intermediate data allocated at the moment
  - Increase of available parallelism at the moment
- Garbage collection (identification and disposal of DFs, which are no longer needed)
  - Reducing memory consumption
- Replication of DFs and their timely delivery to CFs
  - Reducing execution time by increasing data availability on computing nodes

In general the problems are algorithmicaly hard, thus particular solutions and heuristics are demanded.

Recommendations: a Way to Improve Performance Through Direct Execution Control

To improve quality of constructed programs *recommendations* are introduced.

Recommendations are high-level hints to the system, which affect the construction and execuition process.

Recommendations allow user to give a clue to the system how he sees efficient FA execution. Recommendations are given in a high-level DSL without the need to program the desired behavior in a low-level language.

Peculiarities of recommendations use:

- Impossibility of bringing an error into programs, only non-functional properties are affected (execution time, memory consumption, etc.)
- Ability for a separate specialist to tune FA execution performance.
- Recommendations can be safely discarded or replaced, in particular, generated by the system

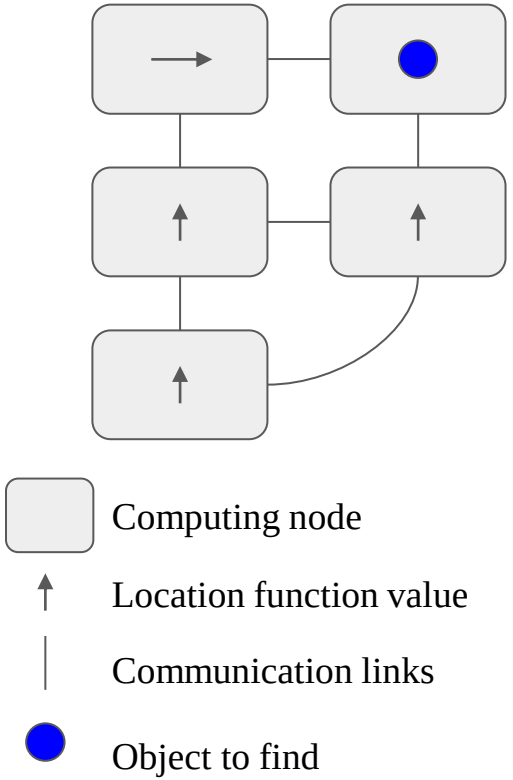
The Concept of Distributed Dynamic Objects Lookup

Goal: to provide dynamic decentralized lookup of a computing node, which contains an object given object's identifier.

Location function L: id × N × T → N

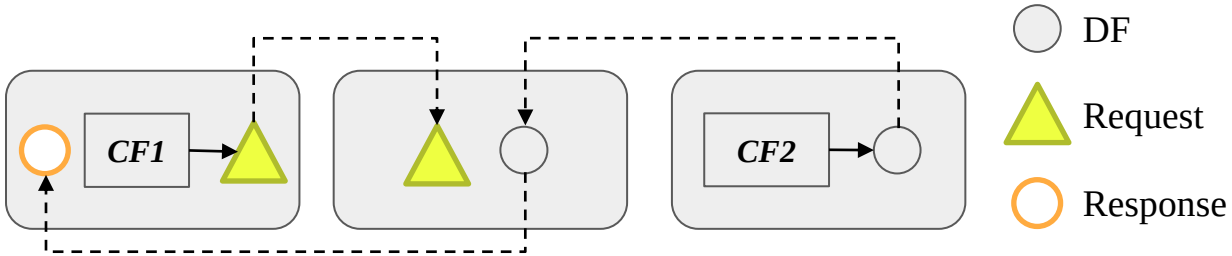
For given identifier on given computing node at given time moment the function defines the next computing node on the path to the object.

The path must lead to the object, sooner or later



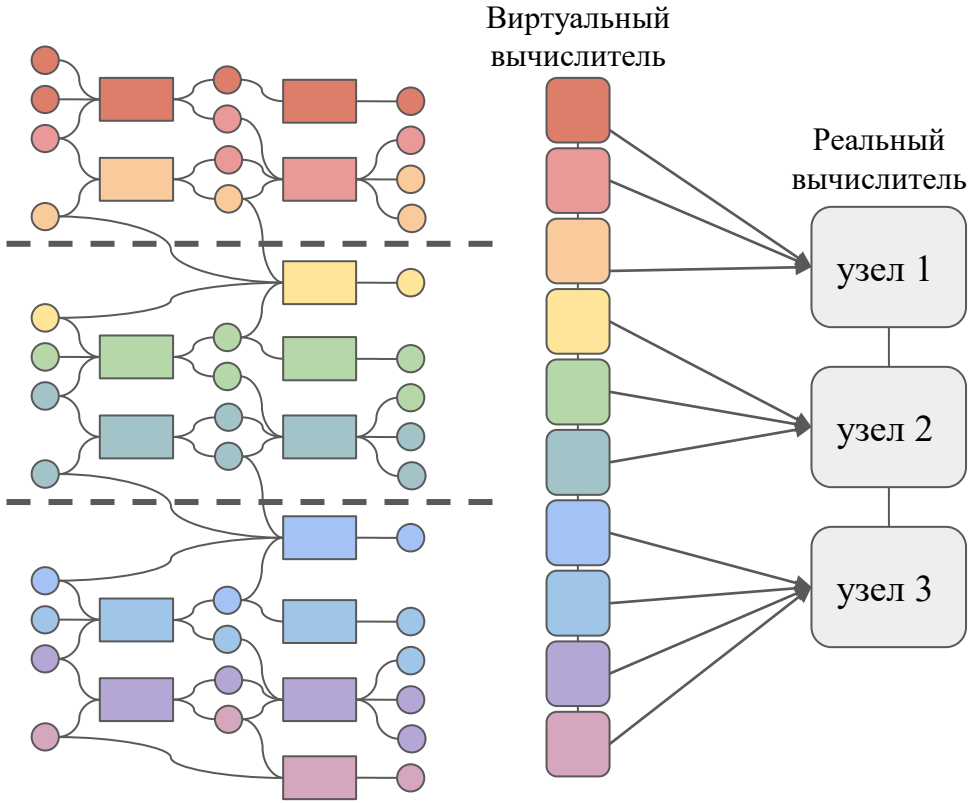
The Basic Distributed Interpretation Algorithm

- Begin: spawn CFs, defined by FA’s operators
- All objects are to be placed and moved according to the location function:
  - CF – by CF id
  - DF – by DF id
  - DF request: by DF id
  - Response to request: by CF id
- On CF spawn: create a request for every input DF
- When a request and the requested DF are on the same computing node replace the request with the response



The Rope of Beads Algorithm: Decentralized Dynamic Distribution of Fragments Which Takes Data Locality into Account

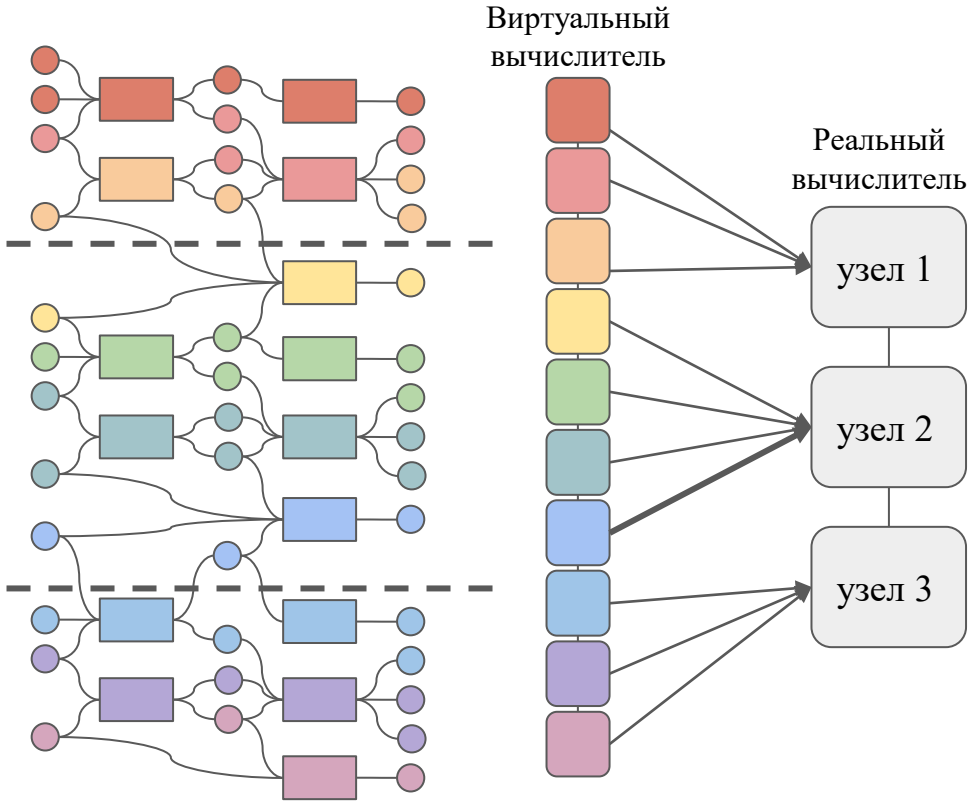
- С помощью рекомендаций статически задаётся отображение множеств ФВ и ФД на виртуальный вычислитель.
- Виртуальный вычислитель динамически отображается на реальный вычислитель
  - Соседство ФВ и ФД на виртуальном вычислителе сохраняется и на реальном
- Переназначение виртуального узла на другой физический влечёт миграцию соответствующих фрагментов на другой узел и позволяет балансировать нагрузку без нарушения соседства данных
- Распределение реализуется полностью децентрализовано и практически не требует накладных расходов.



Улучшенные версии алгоритмов для поддержки многомерных случаев разработаны Г.А. Щукиным

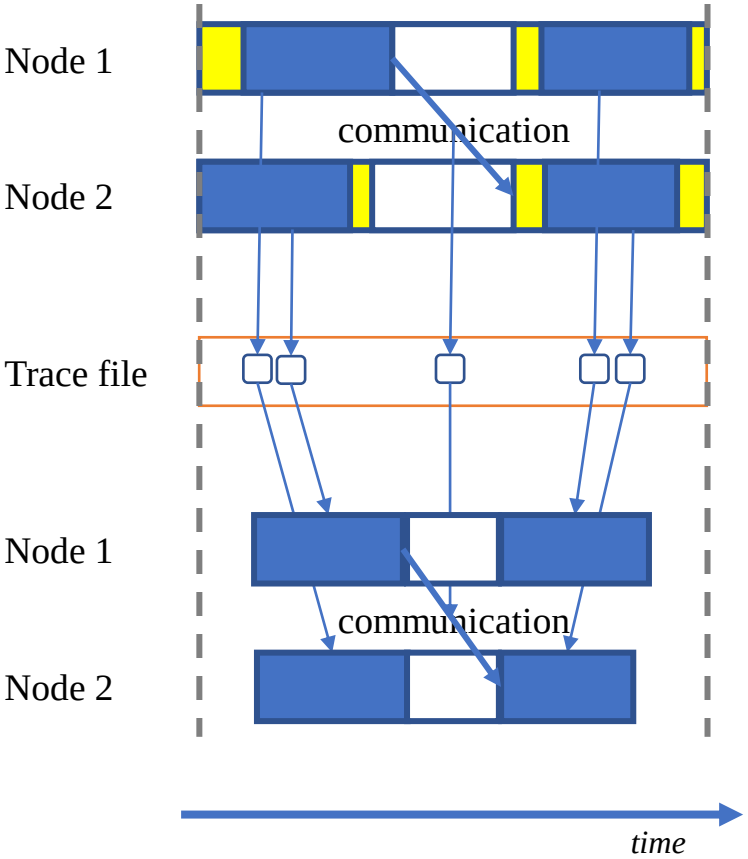
The Rope of Beads Algorithm: Decentralized Dynamic Distribution of Fragments Which Takes Data Locality into Account

- С помощью рекомендаций статически задаётся отображение множеств ФВ и ФД на виртуальный вычислитель.
- Виртуальный вычислитель динамически отображается на реальный вычислитель
  - Соседство ФВ и ФД на виртуальном вычислителе сохраняется и на реальном
- Переназначение виртуального узла на другой физический влечёт миграцию соответствующих фрагментов на другой узел и позволяет балансировать нагрузку без нарушения соседства данных
- Распределение реализуется полностью децентрализовано и практически не требует накладных расходов.



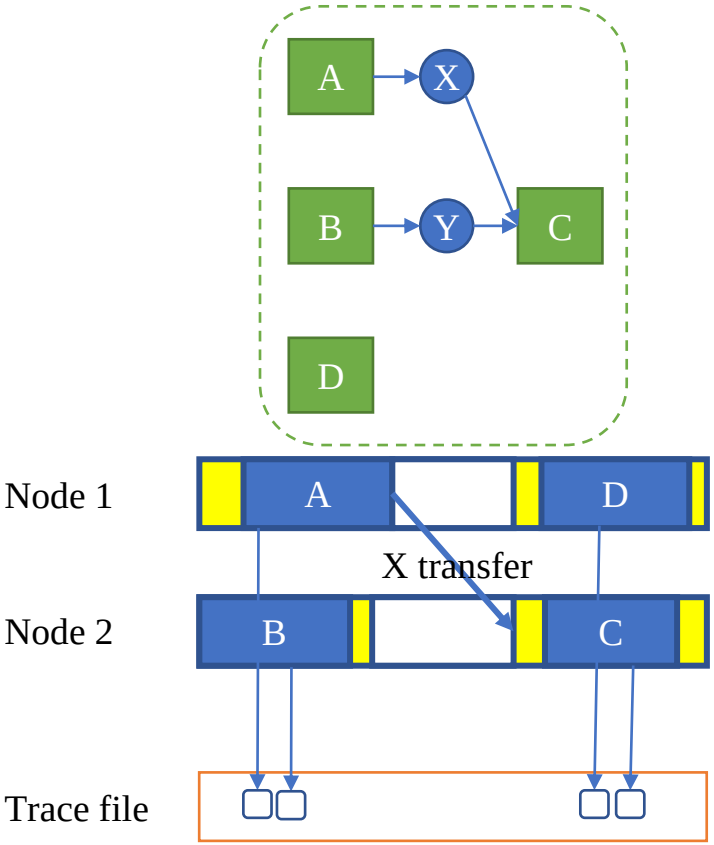
Улучшенные версии алгоритмов для поддержки многомерных случаев разработаны Г.А. Щукиным

The Trace Playback Technique



- A program is first run on some characteristic input data
- Execution events are recorded and can be reproduced for other input data
- Much of the run-time system overhead can be excluded
- Non-determinism of program execution is mostly eliminated

Trace Playback in LuNA System



- Only CF execution start/stop events have to be recorded
  - CF arguments are known from LuNA-program
  - DFs lookup and garbage collection are no problem: all DF consumptions are explicit
- Playback options:
  - Generate a “rigid” MPI program
  - Use a primitive trace repeater
- No need in exact match, provided no information dependencies are violated

Профилирование: дифференцированная диагностика узких мест в программах и версиях системы

Время выполнения программы — слишком скудная характеристика

Нужно контролировать параметры исполнения на разных узлах и в течение времени

Чтобы «не потонуть» в потоке информации её необходимо «уложить»

Analysis of FA and Algorithms

- Обеспечена возможность описывать произвольные алгоритмы в виде ФА в соответствии с идеей фрагментированного программирования
- Обеспечено исполнение произвольных ФА на мультикомпьютерах
- Обеспечена возможность настройки исполнения ФА на основе рекомендаций и специализированных системных алгоритмов

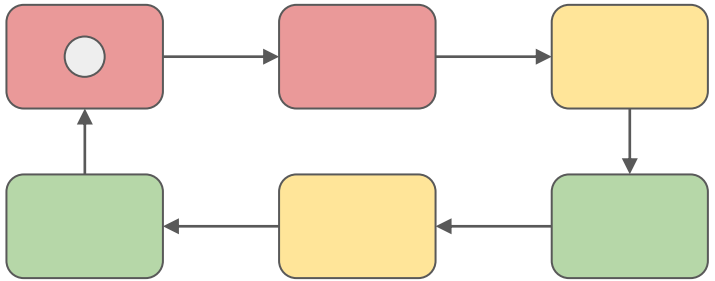
Алгоритм обнаружения завершения работы системы

Задача: децентрализованное обнаружение ситуации завершения работы всеми узлами

Шаги алгоритма:

(каждый узел содержит флаг работы; все узлы логически связаны в кольцо)

- поместить маркер со значением 0 на произвольный узел;
- если на узле с маркером нет работы, то снять флаг работы, переместить маркер на следующий узел в кольце и увеличить его значение на 1;
- при появлении работы на узле устанавливается флаг работы;
- при прохождении маркера по узлу с установленным флагом работы флаг снимается, а значение маркера сбрасывается в 0
- когда значение маркера достигает удвоенного числа узлов, система останавливается



The LuNA System

LuNA (**L**ang**u**age for **N**umerical **A**lgorithms): the Syntax

Описание модуля (внешней C++ процедуры):

```
import proc_name(int, value, name) as comp;
```

Описание фрагмента вычислений:

```
cf a[i]: comp(10, x[i], y[i]);
```

Оператор арифметического цикла:

```
for i=1..10 cf a[i]: comp(10, x[i], y[i]);
```

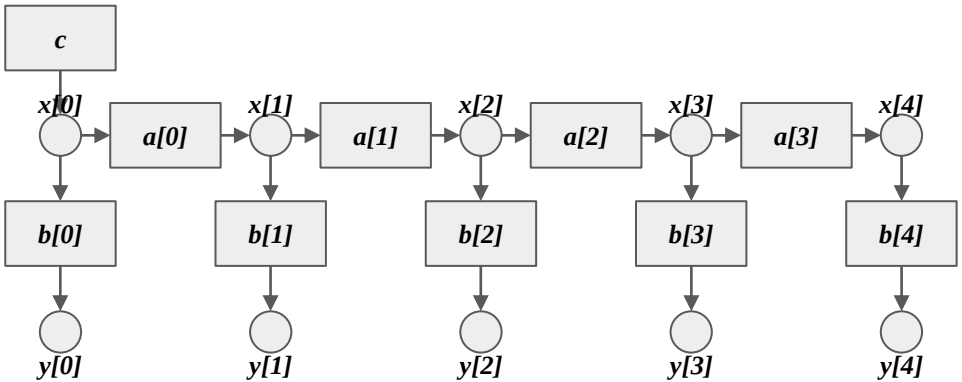
Оператор цикла с предусловием:

```
while x[i]>=0, i=1..out N cf: a[i]: comp(10, x[i], x[i+1]);
```

LuNA-Program Example

```
import init(name);
import calc(value, name);
import conv(int, name);

sub main() {
  df x, y, N;
  while x[i]>0, i=0..out N
    cf a[i]: calc(x[i], x[i+1]);
  for i=0..N {
    cf b[i]: conv(x[i], y[i]);
  cf c: init(x[0]);
}
```



LuNA Peculiarities

Основные свойства:

- Возможность описывать фрагментированные алгоритмы;
- Возможность управления исполнением с помощью расширяемого набора рекомендаций (распределение фрагментов по узлам, сборка мусора, динамическая балансировка нагрузки);
- Возможность определять фрагменты кода на обычных последовательных языках программирования;
- Простая грамматика для разбора существующими инструментами.

Вспомогательные языковые средства:

- комментарии
- макросы
- средства модульности по файлам
- базовые типы данных и операции
- логические операции и условный оператор
- подпрограммы
- вложенные пространства имён

LuNA System: Design

LuNA-компилятор:

- Транслирует программу с языка LuNA в исполняемое представление,
- Лексический и синтаксический анализ: flex+bison,
- Основной язык компилятора: Python,
- Последовательные модули компилируются обычным компилятором.

Исполнительная система:

- Распределённо исполняет скомпилированную LuNA-программу,
- Язык: C++,
- Параллельные средства: MPI, потоки C++.

Другие вопросы проектирования

- Исполняемое представление LuNA-программы на основе мультиагентного подхода и использование языка C++ как языка программ агентов
- Расширяемость компилятора и исполнительной системы системными алгоритмами
- Форматы представления данных и программ в компиляторе и исполнительной системе
- Дополнительные инструменты: профилировщик, отладчик, специализированные исполнительные системы, отладочный стенд для алгоритмов динамической балансировки вычислительной нагрузки
- Технические вопросы трансляции и исполнения LuNA-программ

Comparation of Parallel Programming Using LuNA and MPI

<table>
<tr><th>LuNA</th><th>MPI+Threads</th></tr>
<tr><td colspan="2">Разработать схему параллельной программы</td></tr>
<tr><td colspan="2">Описать последовательные части вычислений в виде C++/Fortran-процедур</td></tr>
<tr><td rowspan="5">Описать и отладить схему параллельной программы на языке LuNA (в виде множества ФВ и ФД)</td><td>Запрограммировать и отладить схему параллельной программы средствами MPI+Threads, в т.ч. коммуникации и синхронизацию процессов и потоков</td></tr>
<tr><td>Обеспечить передачу сообщений на фоне вычислений</td></tr>
<tr><td>Обеспечить настройку программы на размер вычислителя и сетевую топологию</td></tr>
<tr><td>Обеспечить статическую и/или динамическую балансировку нагрузки на узлы</td></tr>
<tr><td>Организовать взаимодействие со спецвычислителем (GPU)</td></tr>
</table>

Преимущества системы LuNA:

- Параллельное программирование как таковое отсутствует, параллельная программа конструируется и исполняется автоматически.
- Отсутствует отладка параллельного кода при модификации программы.
- Параллельная программа по-прежнему продумывается в общих чертах программистом, но описывается не низкоуровневыми средствами, а на предметно-ориентированном языке LuNA.

Demo

Experimental Study

Тест: автоматическое обеспечение динамической балансировки нагрузки на узлы

Идея теста: ускорить вычисления за счёт автоматической динамической балансировки нагрузки на вычислительные узлы

Приложение: моделирование самогравитирующего протопланетного пылевого диска методом частиц-в-ячейках в трёхмерной области*.

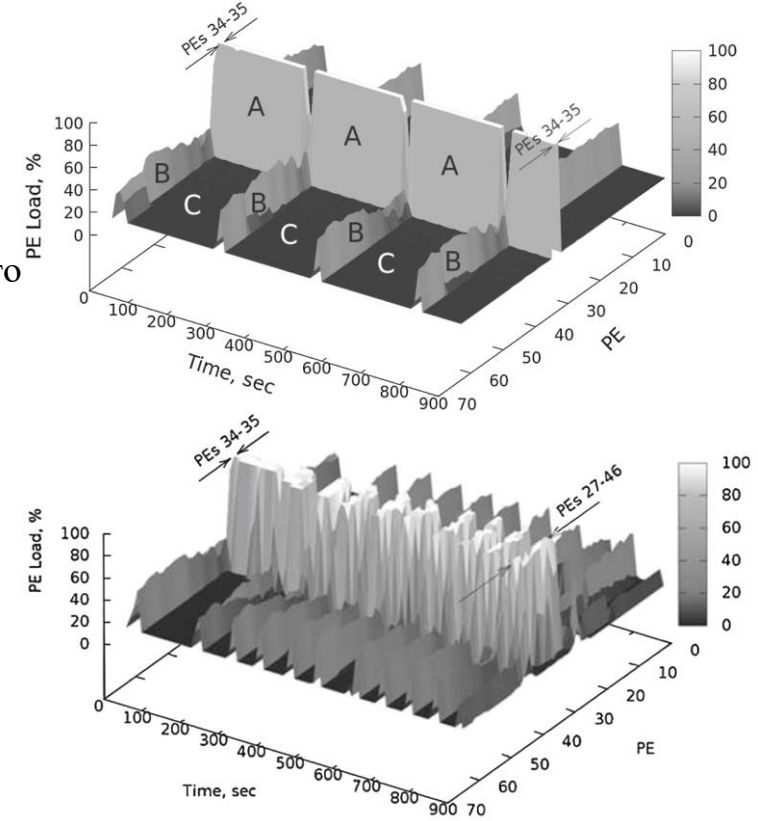
Размер сетки: 64^3, **кол-во частиц:** 10^7

Фрагментация: 16^3

Время работы без балансировки: **2320 сек.**

Время работы с балансировкой: **860 сек.**

Вывод: динамическая балансировка нагрузки обеспечена автоматически



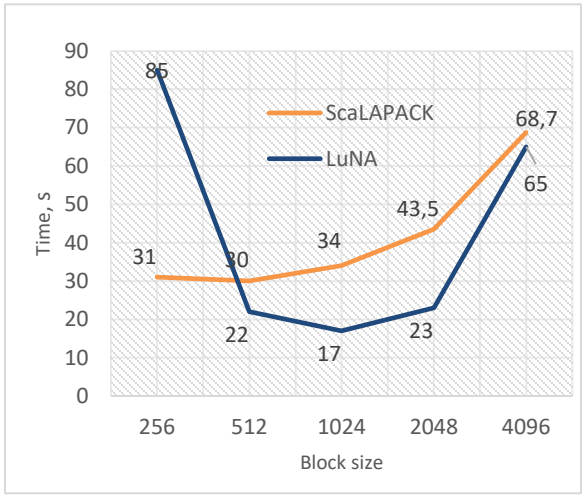
A — сдвиг частиц

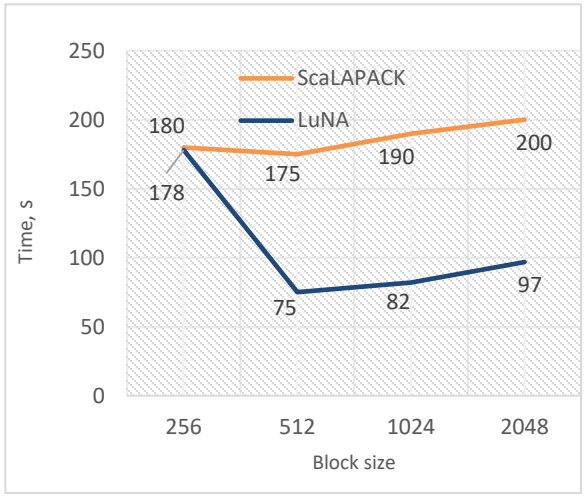
B — решение уравнения Пуассона

C — простой оборудования

*) Victor E. Malyshkin, Vladislav A. Perepelkin. The PIC Implementation in LuNA System of Fragmented Programming // The Journal of Supercomputing, Special Issue on Parallel Computing Technologies. Springer, 2014. pp. 89-97. DOI: 10.1007/s11227-014-1216-8

Тест: разложение Холеского, сравнение специализированной исполнительной системы и ScaLAPACK





Зависимость времени выполнения LuNA-программы и реализации ScaLAPACK для различных размеров блока. Размер матрицы: 32768 (слева) и 65536 (справа).

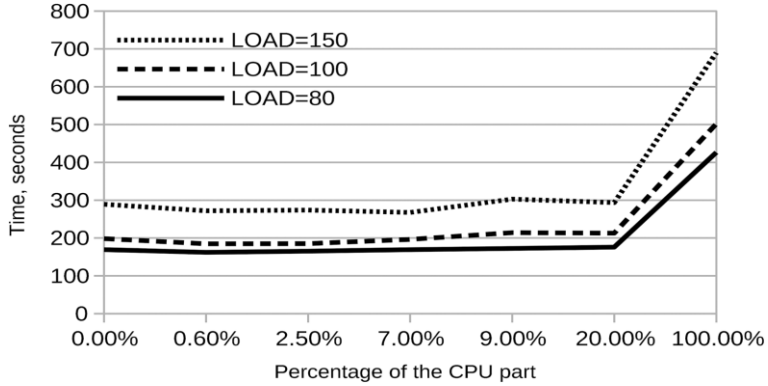
Достигнута производительность, не уступающая таковой ScaLAPACK

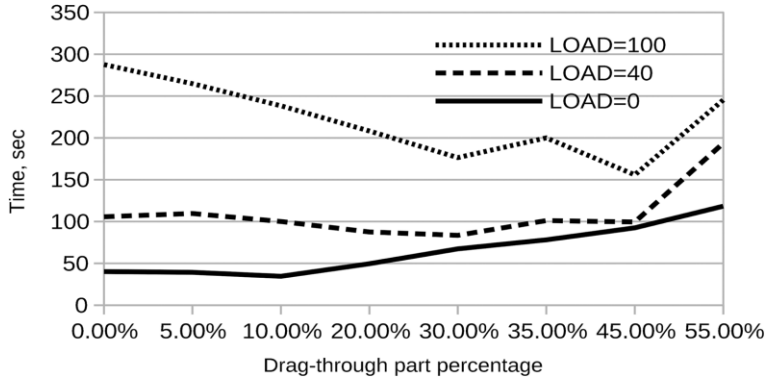
Тест: Автоматизация использования спецвычислителей на примере GPU

Идея теста: проверить автоматизацию использования спецвычислителей при выполнении LuNA-программ.

Приложение: модельная задача, решаемая явным методом на регулярной сетке. Объём вычислений на каждый узел сетки является параметром (LOAD).

Результат: задача успешно отрабатывает на гибридном вычислительном узле (CPU+GPU), причём лучшее время достигается совместным использованием и CPU, и GPU.





Поддержка GPU реализована Н.А. Беляевым

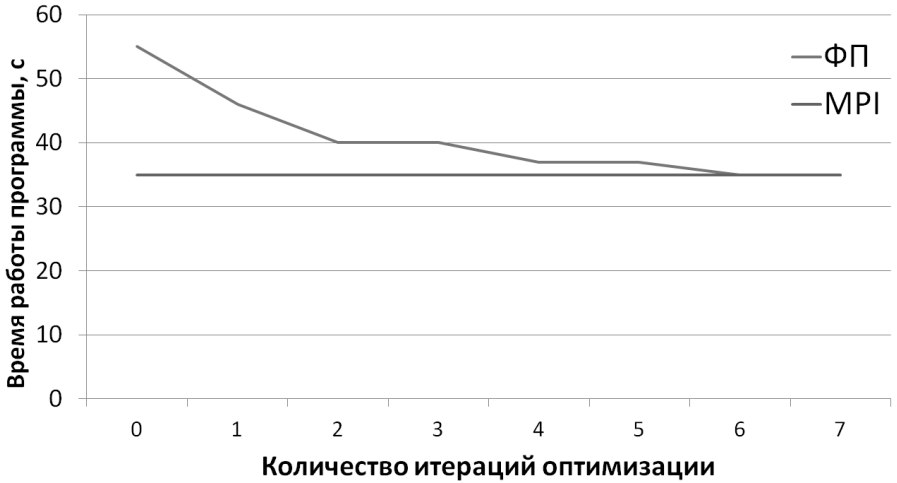
Тест: Автоматическая адаптация исполнения LuNA-программы на основе профилирования

Идея теста: Автоматически настроить отображение фрагментов LuNA-программы на узлы мультикомпьютера на основе профилирования

Приложение: умножение плотных матриц размера 6000×6000

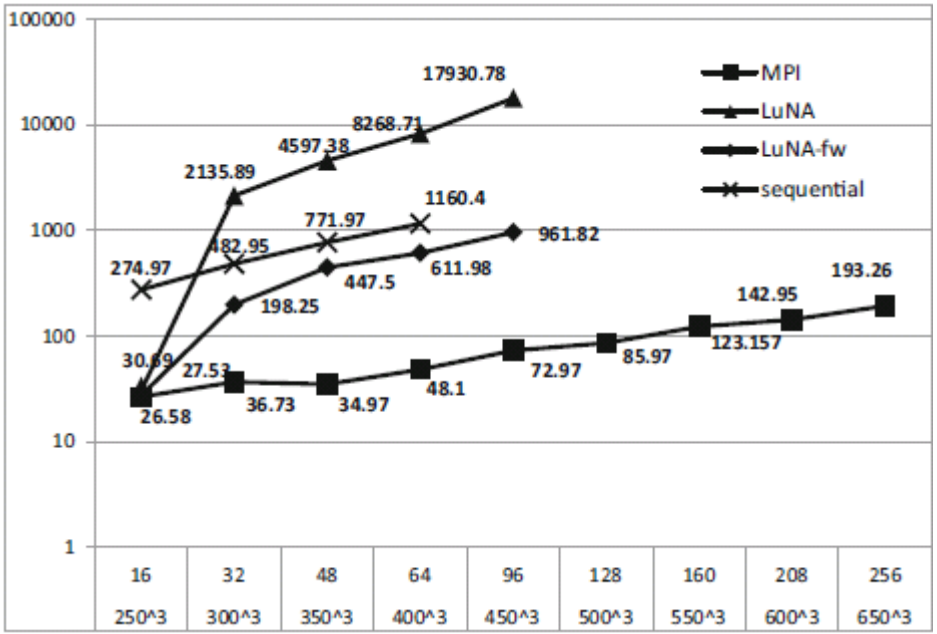
Алгоритм настройки: Путём анализа профиля исполнения программы выявлялись дисбалансы нагрузки, и в моменты дисбаланса часть фрагментов переназначалась (для последующих запусков) с перегруженных на недогруженные узлы.

Результат: За серию прогонов цикла настройки время выполнения сократилось до времени выполнения той же задачи, реализованной вручную средствами MPI.



Тест: решение краевой задачи фильтрации в системе «нефть—вода—газ»

Идея теста: проверить работоспособность и эффективность системы на реальном приложении
Приложение: решатель краевой задачи фильтрации в системе «нефть—вода—газ»*
Результаты: LuNA-программа успешно отрабатывает на тестовых данных. По производительности уступает ручной MPI-реализации. Настройка исполнения LuNA программы позволяет существенно улучшить её производительность.



Приложение разработано на языке LuNA коллегами из КазНУ им. аль Фараби (г. Алма-Ата, Казахстан)

*) Akhmed-Zaki, D., Lebedev, D., Perepelkin, V. Implementation of a three dimensional three-phase fluid flow (“oil–water–gas”) numerical model in LuNA fragmented programming system // Journal of Supercomputing (2017). - 73(2). Springer, 2017. pp. 624-630. DOI: 10.1007/s11227-016-1780-1

Тест: Оптимизация исполнения LuNA-программ на основе воспроизведения трасс

- Приложение: моделирование эволюции самогравитирующего протопланетного диска методом частиц-в-ячейках
- Исполнение на основе воспроизведения трасс показывает существенное улучшение производительности

Parameters			Execution time (sec.)		
Mesh size	Particles	Cores	MPI	LuNA-TP	LuNA
100^3	10^6	64	5.287	13.69	355.5
150^3	10^6	64	18.896	31.088	732.833
150^3	10^7	64	23.594	111.194	2983
150^3	10^7	125	23.352	118.32	3086.411
150^3	10^6	343	33.697	39.651	1008.655

Список приложений, реализованных в системе LuNA

Решение модельного уравнения теплопроводности в двух- и трёхмерном случае

- Akhmed-Zaki, D., Lebedev, D., Perepelkin, V. Implementation of a 3D model heat equation using fragmented programming technology // J Supercomput. 2019. pp. 7827-7832. DOI: 10.1007/s11227-018-2710-1

Реализация операций редукционного типа на примере модельной задачи суммирования множества массивов данных

- V.E. Malyshkin, V.A. Perepelkin, A.A. Tkacheva. Control Flow Usage to Improve Performance of Fragmented Programs Execution // In Proc 13th International Conference on Parallel Computing Technologies. LNCS 9251. Springer, 2015. pp. 86-90. DOI: 10.1007/978-3-319-21909-7_9.

Решение уравнения Пуассона методом Зейделя в трёхмерной области

- Malyshkin. V., Perepelkin. V., Schukin G. Scalable Distributed Data Allocation in LuNA Fragmented Programming System // Journal of Supercomputing, S.I.: Parallel Computing Technologies - 2017. Springer, 2017. pp. 1-7. DOI: 10.1007/s11227-016-1781-0

Решение двумерного эллиптического уравнения попеременно-треугольным методом

- B. Daribayev, V. Perepelkin, D. Lebedev, D. Akhmed-Zaki. Implementation of the Two-Dimensional Elliptic Equation Model in LuNA Fragmented Programming System // 2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT). 2018. pp. 1-4.

Решение модельного уравнения теплопроводности методом IADE_RB_CG

- С.Е. Киреев, В.А. Перепёлкин. Исследование производительности реализации метода IADE в системе фрагментированного программирования LuNA // Параллельные вычислительные технологии (ПаВТ'2016): труды международной научной конференции (28 марта - 1 апреля 2016 г., г. Архангельск). Челябинск: Издательский центр ЮУрГУ, 2016. с. 780

Блочно-синхронная клеточно-автоматная модель реакции окисления монооксида углерода на поверхности палладия

- В.А. Перепелкин, А.А. Ткачёва. Особенности параллельной реализации синхронных и блочно-синхронных клеточных автоматов в технологии фрагментированного программирования // XIII Всероссийская конференция молодых ученых по математическому моделированию и информационным технологиям. Новосибирск, 2012. с. 29.

Клеточно-автоматное моделирование просачивания влаги сквозь пористую почву

Заключение

Разработаны язык и система LuNA автоматического конструирования параллельных программ численного моделирования на мультикомпьютерах:

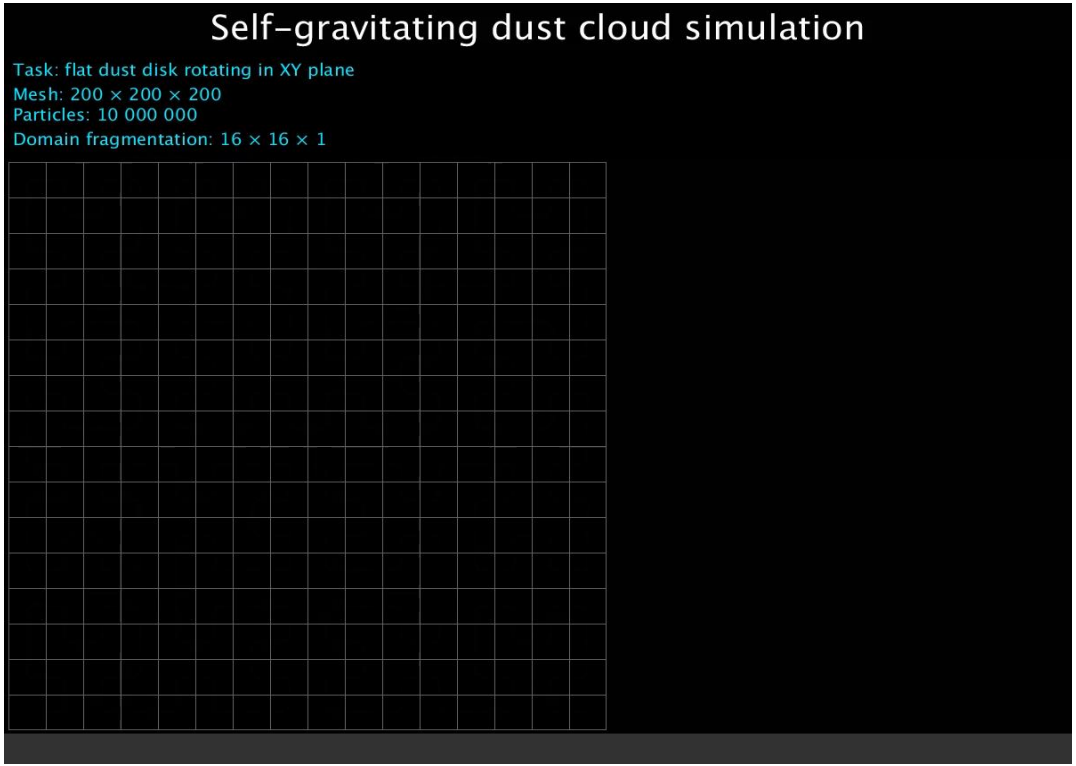
- Разработан язык LuNA описания фрагментированных алгоритмов,
- Разработаны системные алгоритмы трансляции и распределённого исполнения фрагментированных алгоритмов, описанных на языке LuNA,
- Спроектирована и реализована система LuNA, автоматического конструирования параллельных программ для мультикомпьютеров.
- Выполнено экспериментальное исследование системы, которое подтвердило свойства исследуемого подхода

Дальнейшие планы

- Доработка и разработка новых системных алгоритмов, обеспечивающих более высокое качество генерируемых программ для прикладных алгоритмов различных классов.
- Доработка программного кода системы с целью повышения удобства пользователя.
- Использование системы LuNA для поддержки технологии активных знаний*.

*) Malyshkin V.E. Active Knowledge, LuNA and Literacy for Oncoming Centuries // Springer, LNCS, Vol. 9465, pp. 292-303.

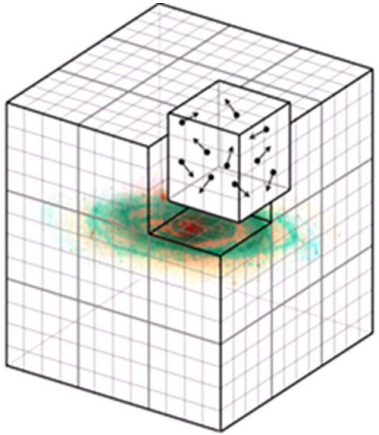
Спасибо за внимание



Анимированная иллюстрация подготовлена С.Е. Киреевым

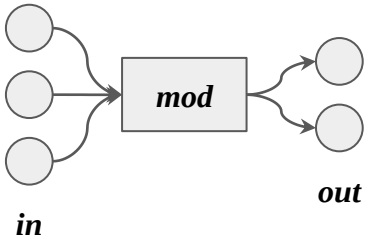
Дополнительные слайды

Язык **LuNA**
(**L**ang**u**age for **N**umerical **A**lgorithms)

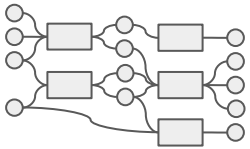


Переменные прикладного алгоритма представляются множеством фрагментов данных (ФД) — блоков совместно используемых переменных. Размер ФД ограничен константой, не зависящей от размера задачи.

Операции представляются множеством фрагментов вычислений (ФВ) — триплетами вида ⟨***in***, ***mod***, ***out***⟩, определяющими, что значения множества ФД ***out*** может быть вычислено из значений множества ФД ***in*** с помощью некоторого модуля ***mod*** (например, последовательной процедурой без побочных эффектов).



Множества ФВ и ФД образуют граф фрагментированного алгоритма (ФА). Исполнение ФА состоит в исполнении всех ФВ по готовности их входных ФД.



Представление прикладного алгоритма

Фрагментированный алгоритм — это четвёрка конечных множеств ⟨N, I, A, O⟩, определяющих рекурсивно-перечислимое множество фрагментов данных и фрагментов вычислений.

- N и I — множества *имён* и *индексных переменных*, из которых конструируются *ссылки* вида n, n[m], n[i][m[j]] и т.п., где n,m ∈ N, i,j ∈ I.
- A — множество *фрагментов кода;*
- O — множество операторов одного из трёх видов:
  - оператор исполнения $\langle a,r_1,...,r_L\rangle$, a ∈ A, $r_1,...,r_L$ — ссылки;
  - оператор арифметического цикла ⟨i,f,l,B⟩, i ∈ I, f и l — ссылки, B — конечное множество операторов;
  - оператор цикла с предусловием ⟨i,c,b,e,B⟩, i ∈ I, c, b и e — ссылки, B — конечное множество операторов.

Пример LuNA-программы

```
#!/usr/bin/luna
/*
 Matrix multiplication.
*/

import c_init(int, name) as init;
import c_init_submatrix(int `file, int `row, int `col, int `height,
import c_mult_matrix(value `a, value `b, name `c) as mult_mat;
import c_sum_matrix(value `a, value `b, name `c) as sum_mat;
import c_save_submatrix(value `a, int `row, int `col) as save_mat;
import c_copy_matrix(value `a, name `b) as copy_mat;

#define FG_SIZE 4
#define FG_COUNT 10

sub calc_mat(name A, name B, name C, int i, int j, int N)
{
>---df Ctmp, Csum;

>---for k=0..N-1
>---{
>--->---cf f[i][j][k]: mult_mat(A[i][k], B[k][j], Ctmp[k]);
>---}

>---if N>1
>--->---sum_mat(Ctmp[0], Ctmp[1], Csum[1]);
>---if N==1
>--->---copy_mat(Ctmp[0], Csum[0]);

>---for k=2..N-1
>--->---cf sum[k]: sum_mat(Ctmp[k], Csum[k-1], Csum[k]);

>---copy_mat(Csum[N-1], C);
}
```

```
extern "C"
void c_copy_matrix(const InputDF &a, OutputDF &b)
{
    b.copy(a);
}

extern "C"
void c_sum_matrix(const InputDF &a, const InputDF &b, OutputDF &c)
{
>---struct Matrix A, B, C;
>---attach(A, a);
>---attach(B, b);
>---assert(A.Height==B.Height && A.Width==B.Width);
>---create_matr_buf(c, A.Height, A.Width);
>---C.Height=A.Height;
>---C.Width=B.Width;
    const size_t *data=static_cast<const size_t*>(c.get_data());
    C.Data=(double*)(data + 2);

    for (size_t i=0; i<C.Height*C.Width; i++) {
>--->---C.Data[i]=A.Data[i]+B.Data[i];
    }
}
```

Алгоритм определения очерёдности запрашивания фрагментов данных фрагментом вычислений

Задача: для каждого оператора определить перечень входных фрагментов данных и порядок их запрашивания у исполнительной системы

Вход: оператор

Выход: набор множеств ссылок

Шаги алгоритма:

- Положить множество фрагментов данных с известными значениями пустым
- Рекурсивно разобрать все выражения и ссылки в операторе;
- Выбрать все ссылки, значения индексных выражений которых известны, а значения самих ссылок — нет;
- Добавить это множество ссылок в выходной набор и во множество известных ФД
- Повторять до тех пор, пока значения всех ссылок не окажутся известными

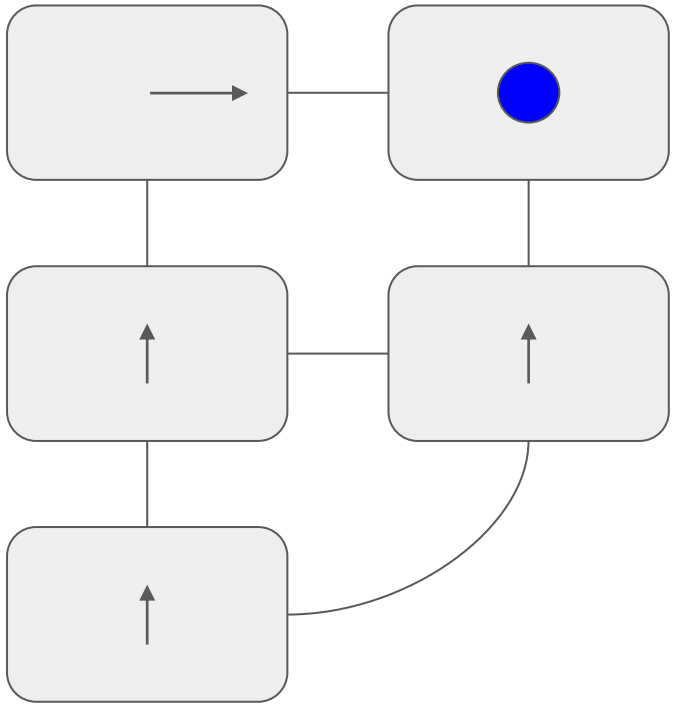
show(x[N][y[M]]);

Очерёдность запрашивания и пример значений:

1.	N, M	N=5, M=7
2.	y[M]	y[7]=12
3.	x[N][y[M]]	x[5][12]

Механизм динамического поиска объектов в распределённой системе

Задача: обеспечивать динамическое децентрализованное обнаружение узла мультикомпьютера, на котором находится именованный объект по имени объекта
Вход: Имя объекта
Выход: Номер соседнего узла в направлении к искомому узлу
Шаги алгоритма: (зависит от алгоритма динамического отображения)



Алгоритм Rope of Beads*

Задача: динамически децентрализованно отображать множество имён на узлы мультикомпьютера с учётом сетевой топологии

Вход: имя объекта

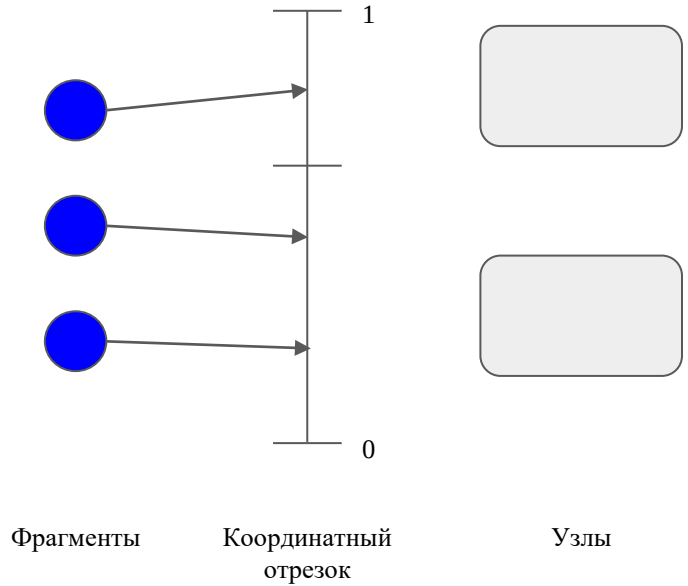
Выход: номер соседнего узла

Шаги алгоритма:

(каждый узел хранит границы [A; B) своего диапазона координат и номера смежных узлов)

- Определить «координату» C имени (задана статически и известна на всех узлах)
- Если A≤C<B, то вернуть номер текущего узла
- Если C<A, то вернуть номер предыдущего смежного узла
- Иначе вернуть номер следующего узла

Граница может динамически сдвигаться парой смежных узлов: [A;B),[B;C) → [A;B'),[B';C), при условии A≤B'<C.



*) Название придумано Щукиным Г.А.

Механизм динамической балансировки нагрузки на узлы методом Job Stealing

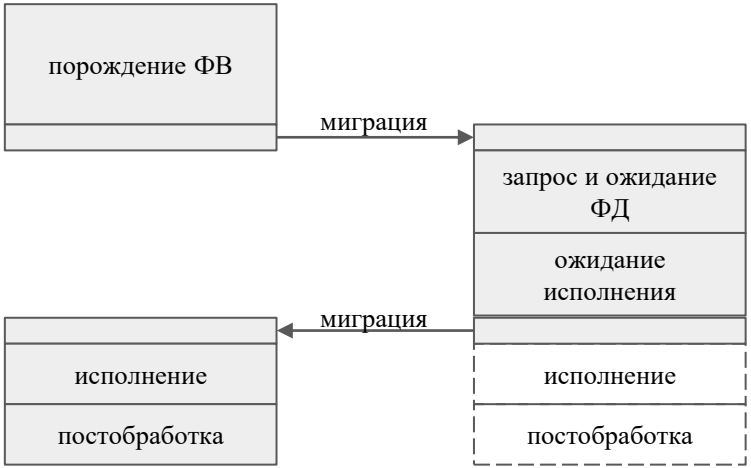
Задача: Динамически балансировать нагрузку на узлы мультикомпьютера

Схема работы:

- При простое узел отправляет запрос на соседние узлы;
- Если при получении запроса на узле есть избыток готовых к исполнению фрагментов вычислений, то часть из них отправить на запросивший узел.

Механизм может дополнять другие алгоритмы балансировки (напр., на основе Rope of Beads).

Балансировке подвергаются только готовые к исполнению фрагменты вычислений



Алгоритмы доставки значений фрагментов данных потребляющим их фрагментам вычислений

Задача: доставлять входные фрагменты данных на узел, где будет исполняться фрагмент вычислений

Доставка по запросу:

- запрос (пара ⟨имя ФД, узел ФВ⟩) генерируется ФВ и передаётся на узел хранения ФД
- ФД, будучи выработанным, передаётся на узел хранения
- когда запрос и ФД оказываются на одном узле, то копия ФД передаётся на узел ФВ

Упреждающая посылка ФД:

- ФД, будучи выработанным, передаётся на узел ФВ
- ФВ не запрашивает, а просто ожидает входных ФД

Алгоритмы распределённой сборки мусора

Задача: освобождать память, занятую фрагментами данных, значения которых более не понадобятся.

Алгоритм 1: удалять по счётчику количества потреблений.

Алгоритм 2: удалять по событию, которое происходит позже последнего потребления.

Алгоритм 3: удалять по завершению исполнения блока, в котором объявлены фрагменты данных

Шаги алгоритма 3:

- при исполнении блочного оператора на узле исполнения размещается счётчик неисполненных дочерних операторов
- при завершении не блочного оператора счётчик родительского блока декрементируется
- при достижении нуля счётчиком декрементируется счётчик родительского блока и удаляются объявленные в текущем блоке фрагменты данных

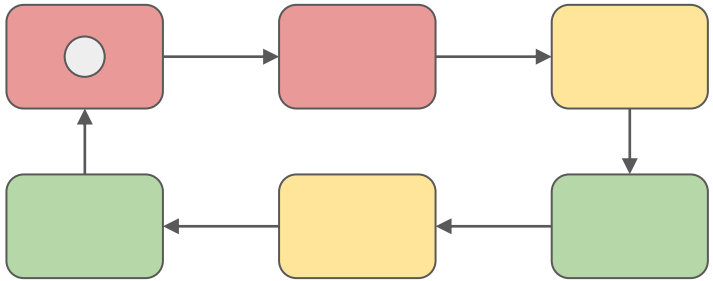
Алгоритм обнаружения завершения работы системы

Задача: децентрализованное обнаружение ситуации завершения работы всеми узлами

Шаги алгоритма:

(каждый узел содержит флаг работы; все узлы логически связаны в кольцо)

- поместить маркер со значением 0 на произвольный узел;
- если на узле с маркером нет работы, то снять флаг работы, переместить маркер на следующий узел в кольце и увеличить его значение на 1;
- при появлении работы на узле устанавливается флаг работы;
- при прохождении маркера по узлу с установленным флагом работы флаг снимается, а значение маркера сбрасывается в 0
- когда значение маркера достигает удвоенного числа узлов, система останавливается



Тестирование производительности базовой исполнительной системы

•Тест: решение уравнения теплопроводности итерационным методом на 3D сетке

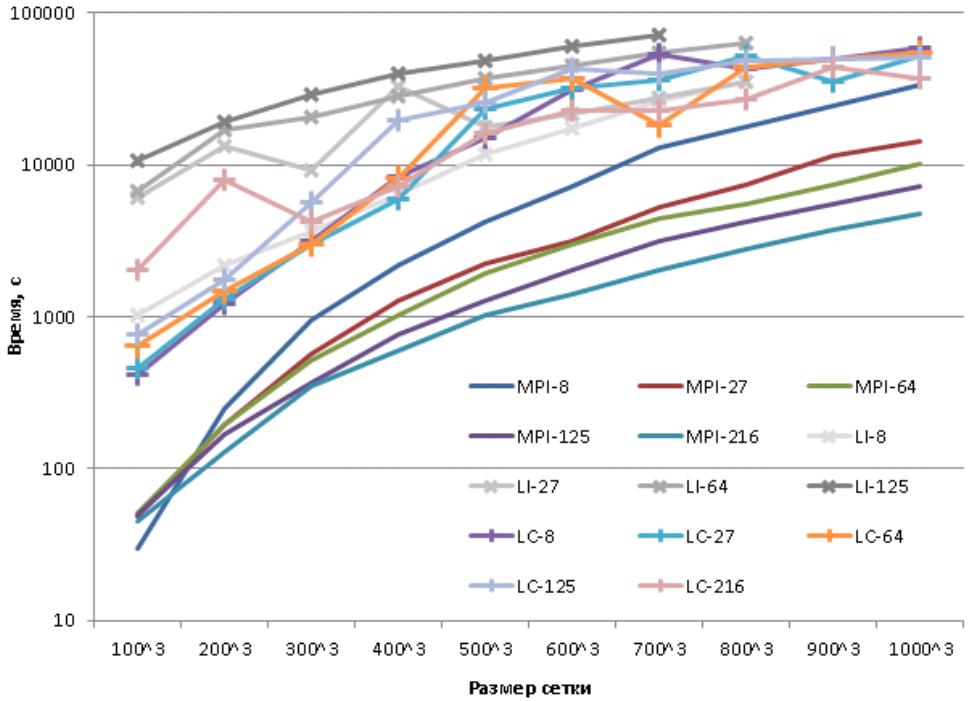
• Реализации (число означает ядра):

–Ручная (MPI)

–LuNA-интерпретатор (LI)

–LuNA-компилятор (LC)

•Получено существенное увеличение производительности, особенно при мелкой фрагментации



Проблемы реализации сложных моделей на примере метода частиц-в-ячейках

Пыль

$$\frac{\partial f}{\partial t} + \mathbf{u}\frac{\partial f}{\partial \mathbf{r}} + \frac{\mathbf{F}}{m}\frac{\partial f}{\partial \mathbf{u}} = 0$$

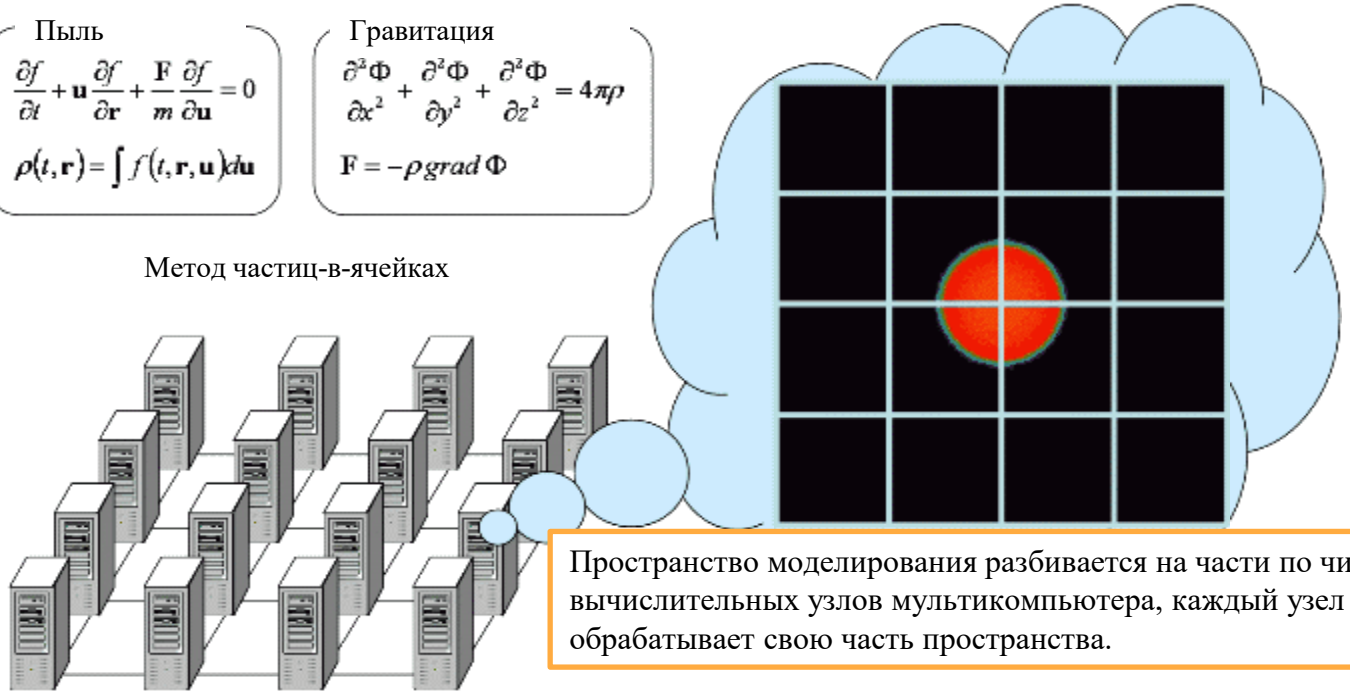
$$\rho(t,\mathbf{r}) = \int f(t,\mathbf{r},\mathbf{u})d\mathbf{u}$$

Гравитация

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + \frac{\partial^2 \Phi}{\partial z^2} = 4\pi\rho$$

$$\mathbf{F} = -\rho \, grad \, \Phi$$

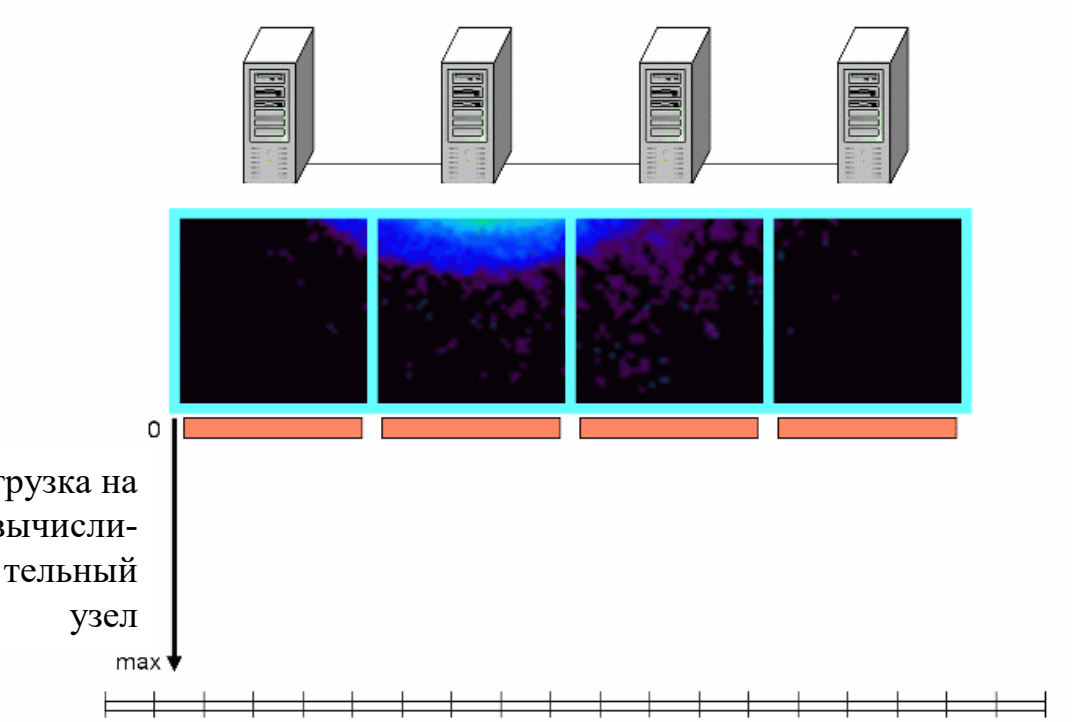
Метод частиц-в-ячейках



Пространство моделирования разбивается на части по числу вычислительных узлов мультикомпьютера, каждый узел обрабатывает свою часть пространства.

Анимированная иллюстрация подготовлена С.Е. Киреевым

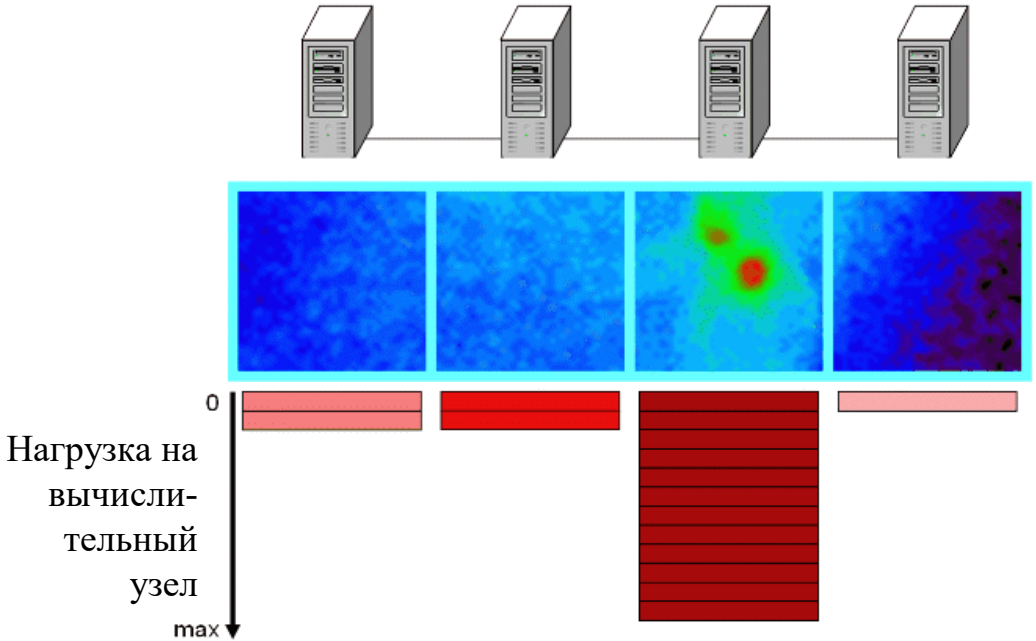
Демонстрация возникновения дисбаланса загрузки вычислительных узлов в процессе вычислений



Нагрузка на вычислительные узлы изменяется динамически в зависимости от количества вещества в соответствующей области пространства моделирования. Это приводит к простою недогруженного оборудования.

Анимированная иллюстрация подготовлена С.Е. Киреевым

Демонстрация устранения дисбаланса путём перераспределения нагрузки



При возникновении дисбаланса нагрузки часть нагрузки передаётся с перегруженных вычислительных узлов на недогруженные, чтобы обеспечить эффективную работу мультикомпьютера

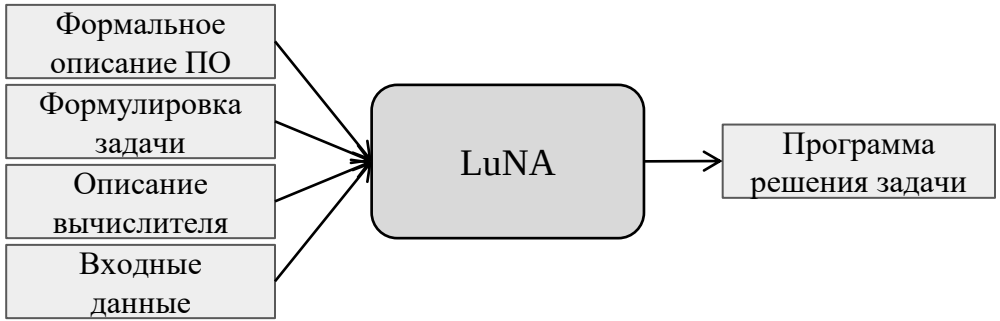
Анимированная иллюстрация подготовлена С.Е. Киреевым

Подход к автоматическому конструированию параллельной программы

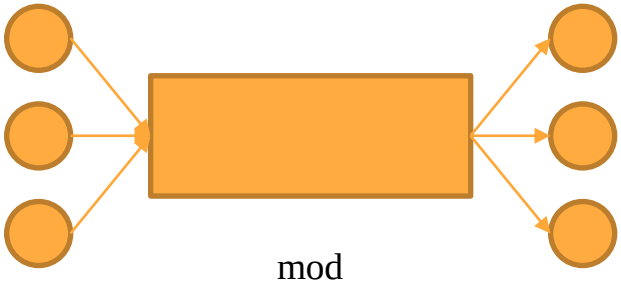
Предметная область (ПО) частично описывается формально в виде множества понятий (переменных) и их связей — возможности вычислять значения одних переменных из других (операций).

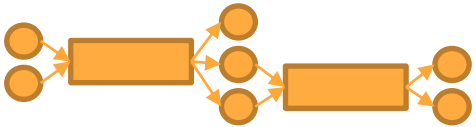
Задача формулируется как два множества — данных переменных и искомых.

Программа решения задачи конструируется системой с учётом свойств вычислителя и входных данных.



Фрагментированный алгоритм





Фрагментированный алгоритм (ФА) – это формальное описание прикладного алгоритма, которое:

- Является адаптированным понятием рекурсивной функции (операторный терм)
- Позволяет описывать вычислительные модели частного вида
- Ориентировано на параллельное исполнение на мультикомпьютерах

В частности:

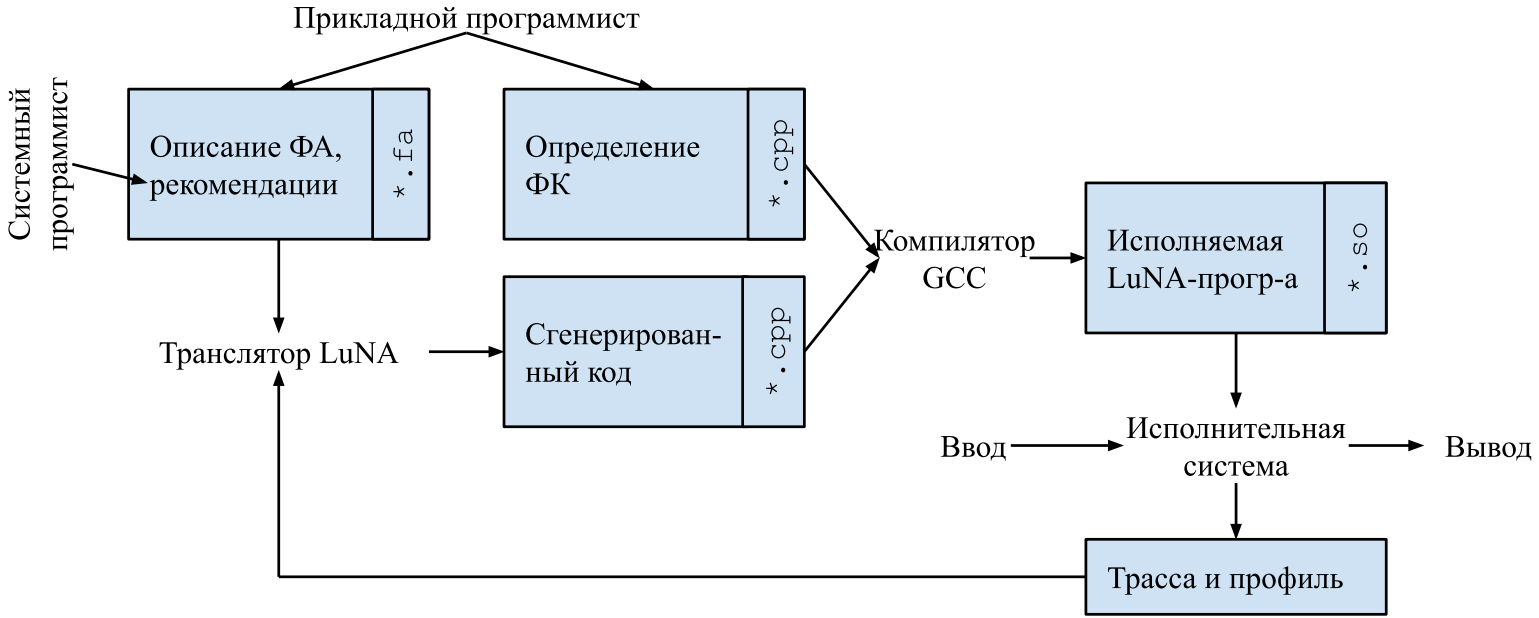
- Промежуточные значения именуются
- Операции могут иметь несколько выходных параметров

Операторы фрагментированного алгоритма

Ссылка – выражение вида $x[a_1,\ldots,a_n]$, где x – имя, а a_i – либо индексная переменная, либо ссылка

- Оператор исполнения (задаёт одиночный фрагмент вычислений):
  - Конечное множество ссылок, задающих входные фрагменты данных
  - Конечное множество ссылок, задающих выходные фрагменты данных
  - Фрагмент кода (модуль, процедура), способный вычислить значения выходных фрагментов данных из значений входных
- Оператор арифметического цикла:
  - Индексная переменная – счётчик цикла
  - Две ссылки, задающие начало и конец диапазона счётчика цикла
  - Тело цикла – конечный набор операторов
- Оператор цикла с предусловием:
  - Индексная переменная – счётчик цикла
  - Ссылка, задающая предусловие

Архитектура системы LuNA



Родственные проекты

Лаб. СПП — одна из нескольких лабораторий РАН, **занимающихся автоматизацией** параллельного программирования для суперЭВМ и развивающих **собственный** академический программный продукт

Организация	Люди	Продукт	Характеристика
ИПМ им. М.В. Келдыша РАН	А.Н. Андрианов	Норма	Генерация параллельных программ по непроцедурному описанию алгоритма
ИПМ им. М.В. Келдыша РАН	В.А. Крюков	DVMH / SAPFOR	Полуавтоматическое распараллеливание аннотированного последовательного кода
ИСИ СО РАН	В.Н. Касьянов	Cloud SISAL / CPPS	Функциональный потоковый язык спецификации вычислений
ИПС им. А.К. Айламазяна РАН	С.М. Абрамов	T-Система	Расширение последовательного кода средствами описания параллелизма задач
ИВМиМГ СО РАН	В.Э. Малышкин	LuNA	Синтез параллельных программ численного моделирования, язык LuNA

Основные зарубежные проекты, занимающиеся сходной проблематикой:

Организация	Люди	Продукт	Характеристика
University of Tennessee	J. Dongarra	PaRSEC / DPLASMA	Исполнительная система с task-based параллелизмом для распределённых неоднородных вычислителей
University of Illinois	L. Kale	Charm++	Параллельное программирование на основе мигрирующих объектов, обменивающихся сообщениями
Stanford University	A. Aiken	Legion / Regent	Параллельное программирование с явной фрагментацией данных и параллелизмом задач
Louisiana State University	H. Kaiser	HPX	Библиотека поддержки task-based параллелизма с глобальным адресным пространством

Обзор средств автоматического конструирования параллельных программ

	DVMH/ Sapfor, HPF	Hadoop (MapReduce)	HOPMA, Sisal	Charm++, T-Система	PGAS (Titanium, UPC, X-10)	PaRSEC	Legion	LuNA
Распределённая память	+	+	-	+	+	+	+	+
Параллельное представление алгоритма	-	+	+	+	+	+	+	+
Крупноблочный параллелизм	-	+	-	+	+	+	+	+
Универсальная модель вычислений	+	-	+	+	+	-	+	+
Управляемое исполнение	частично	-	-	-	-	+	только вручную	+