ПАРАЛЛЕЛИЗМ В АЛГОРИТМАХ - ВЫЯВЛЁНИЕ И РАЦИОНАЛЬНОЕ ИСПОЛЬЗОВАНИЕ

Ежегодная конференция «СПО: от обучения до разработки»
Тема: Свободное Программное Обеспечение (СПО)
в учебном процессе

Баканов Валерий Михайлович, РТУ МИРЭА / НИУ ВШЭ
915-053-5469, e881e@mail.ru, http://vbakanov.ru/left_1.htm

АКТУАЛЬНОСТЬ ПРОЕКТА:

- Развитие в России процессоров архитектуры, прямо ориентированной на параллелизацию вычислений (“первая ласточкой” можно считать процессоры семейства ЭЛЬБРУС архитектуры VLIW)
- Часто формальное и явно недостаточное для дальнейшего практического применения студентами преподавание кустá дисциплин ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ
- Недостаток “на местах” аппаратной части для освоения данных дисциплин

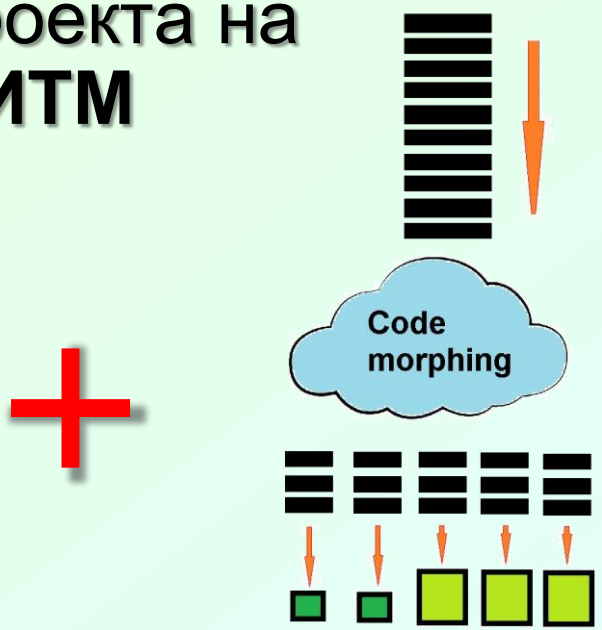
ПРЕДЛАГАЕМОЕ РЕШЕНИЕ (данный проект):

- Разработка набора программных компьютерных моделей для исследования явления параллелизма и его практического использования
- Свободная доступность разработок (как на уровне исходных кодов, так и исполняемых файлов) для всех заинтересованных лиц и учреждений
- Возможно полное покрытие всех сторон данной области знания (от формального выявления параллелизма в алгоритмах до его практического использования в вычислительных практиках)
- Разработка набора методических материалов, направленных на формальную и исследовательскую стороны данной области знания

ОРИЕНТИРОВАННОСТЬ ПРОЕКТА:

- В первую очередь на студентов – будущих разработчиков СИСТЕМНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (конкретно – создание эффективных распараллеливающих блоков компиляторов / интерпретаторов)
- Обеспечение Научно-Исследовательской Работы (*Научных Семинаров*) студентов направлений, связанных с компьютерной обработкой данных

Ориентированность проекта на сущность **АЛГОРИТМ**



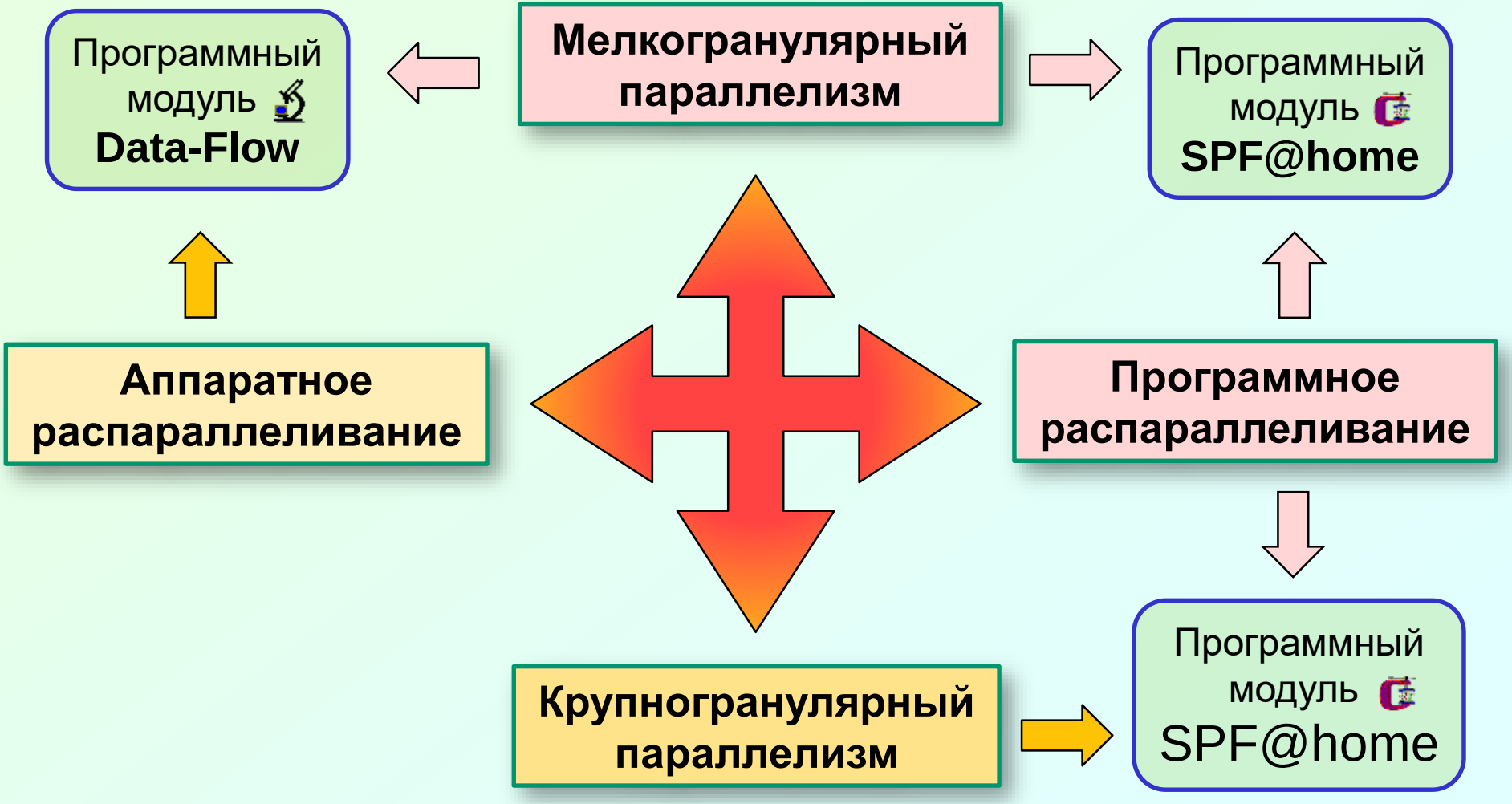
📖 Грань **ВНУТРЕННИЙ (*скры́тый*) ПАРАЛЛЕЛИЗМ** – *относительно новая* исследуемая сторона алгоритма, изучающая наличие в алгоритме собственно параллелизма и его параметров (что необходимо для определения возможности его (параллелизма) использования при обработке данных на параллельных вычислительных системах.

CODE MORPHING (символическое изображение на рис. справа сверху) – преобразование кодовой последовательности из одного вида в другой. Одно из применений ***code morphing*** – преобразование кода из последовательного представления в параллельное с заданными параметрами.

☯ Собственно идея CODE MORPHING, пожалуй, впервые была реализована фирмой Transmeta Corp. при создании процессоров VLIW-архитектуры Crusoe (длина *сверхдлинного слова* 128 бит, 2000 год) и Efficeon (256 бит, 2004 год). В этом же ряду советско/российский проект ЭЛЬБРУС и ITANIUM (последний - совместная разработка Intel Corp. и Hewlet-Packard).

🛠 Анализ часто используемых алгоритмов представлен на WEB-ресурсе AlgoWiki http://algowiki-project.org/ru/. Полное название AlgoWiki - “Открытая энциклопедия свойств алгоритмов”; руководители: Воеводин Вл.В. (НИВЦ МГУ им. М.В.Ломоносова, РФ) и Джек Донгарра (разработчик теста LinPack - университет Теннеси, Knoxville, USA).

Требование учёта максимума сторóн рассматриваемой области знаний при условии минимизации сложности программного обеспечения

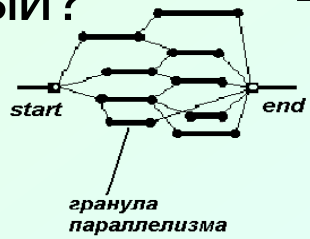


Наивные вопросы по параллельным вычислениям, *количественные ответы* на которые нам необходимо получить в процессе освоения нашего курса

- Каково минимальное время выполнения *заданного (произвольного) алгоритма?*
- При каком именно числе параллельных вычислителей обеспечивается этот минимум времени выполнения?
  - Истинно ли выражение: “чем параллельных вычислителей больше – тем алгоритм вы́полнится быстрее”?
    - Значит ли это, что при стремле́нии числа вычислителей к бесконечности время выполнения алгоритма может быть сколь угодно ма́лым?)
- Как время выполнения алгоритма зависит от числа параллельных вычислителей?

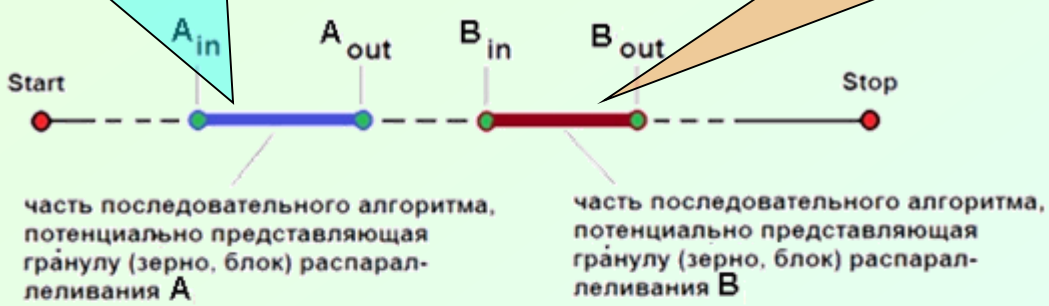
- Более сложные вопросы, ответы на которые может быть эмпирически получен с применением методов моделирования параллельного выполнения алгоритмов:
  - Сохраняется ли присущий данному алгоритму закон вычислительной сложности при параллельном выполнении этого алгоритма?
    - Предложите наиболее простой и максимально показательный эксперимент, подтверждающий или опровергающий предложенную гипотезу по предыдущему вопросу

Каковы *наиболее общие подходы* для преобразования последовательного алгоритма в параллельный?

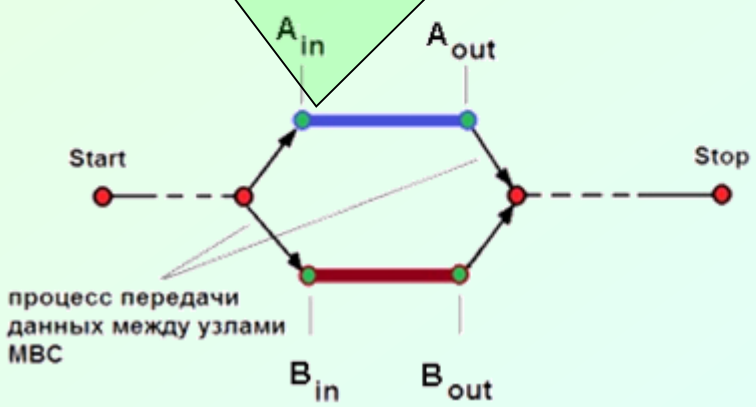


A_{in} – входные данные, A_{out} - выходные (вычисленные) данные последовательности команд A

B_{in} – входные данные, B_{out}^{j} - выходные (вычисленные) данные последовательности команд B



Если входные данные для блока (*зерна, гранулы*) j не зависят от выходных данных блока i, то эти блоки могут выполняться ***независимо*** (т.е. **ПАРАЛЛЕЛЬНО**)



Условием независимого (параллельного) выполнения отдельных частей (блоков, зёрен, гра́нул) алгоритма является их ***независимость по данным*** (если B_{in} зависит от A_{out}, то блок B не может выполняться параллельно с блоком A; в противоположном случае - может).

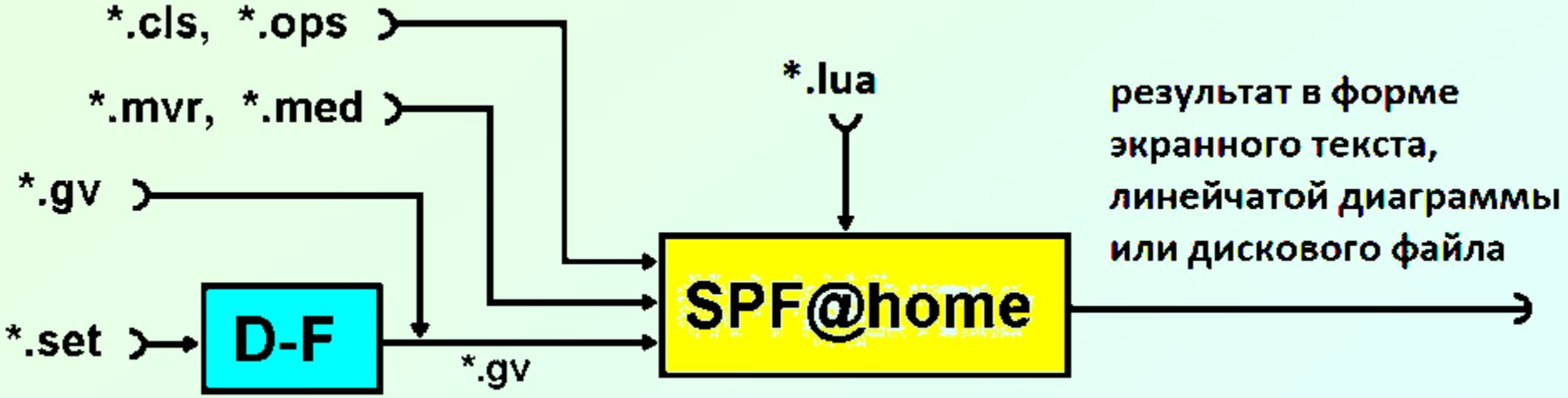
Проблема – такое разбие́ние (*декомпозиция*) на гранулы затруднено, ибо границы блоков априори неизвестны и подлежат определению, при этом распараллеливание каждого изначально-последовательного алгоритма возможно ***множеством способов***, а эффективность (в основном – время выполнения программы) каждого из способов может ***отличаться на порядки***.

Для решения конкретной задачи распараллеливания необходимо решить вопросы:

1) Каков размер *гранулы параллелизма* (от одной машинной команды до тысяч/миллионов)?
2) В каком конкретно месте последовательной программы находятся эти *гранулы*?

Общее число сочетаний таково́, что обычно приводит к *NP*-полной задаче…

Общая схема взаимодействия программных подсистем Data-Flow (D-F) и SPF@home



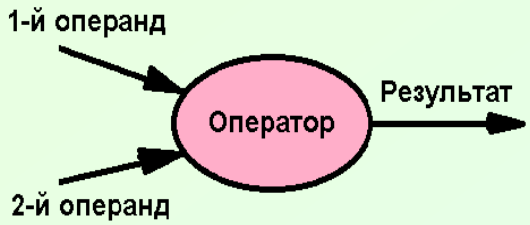
*.set и *.gv – программный файл и файл информационного графа анализируемой программы соответственно,

*.mvr, *.med – файлы метрик вершин и дуг графа алгоритма соответственно,

*.cls, *.ops – файлы параметров вычислителей и операторов программы соответственно,

*.lua – текстовый файл на языке Lua, содержащий методы реорганизации ЯПФ

Общие принципы определение момента выполнения операторов в программе



“По Тьюрингу” сложная программа может быть выполнена как набор простых действий (*операторов, машинных команд*). Если имеется возможность одновременного выполнения нескольких операторов (*аппаратный параллелизм*), то сразу возникает важный вопрос – в какой момент времени выполнять операторы?

Вообще говоря, момент времени выполнения каждого оператора можно определить из следующих соображений:

1. Выполнить оператор как ***можно раньше*** (как только будут “готовы” – определены́ вычислениям или присваиванием – все операнды данного оператора (момент времени t_1).

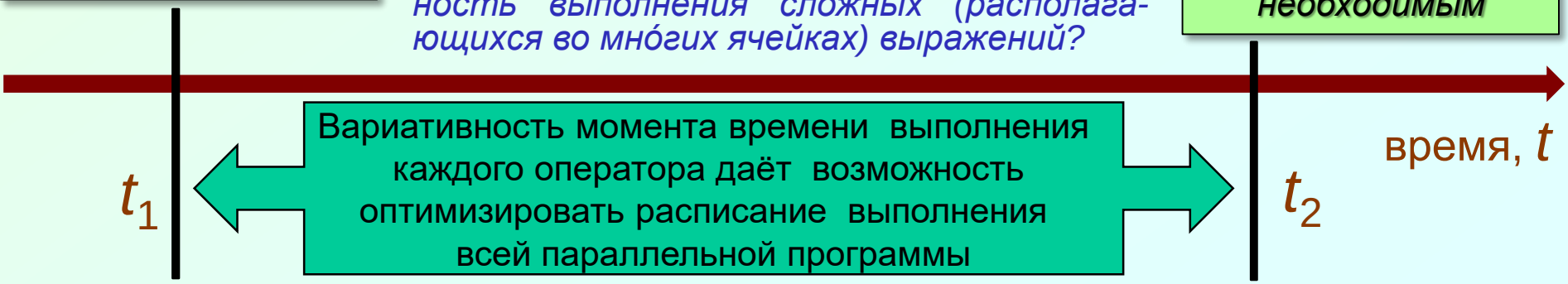
2. Выполнить оператор как ***можно позднее*** (отло́женное, “ленивое”, lazy выполнение); используется в функциональном программировании (момент времени t_2).

3. Выполнить в промежуточный момент между 1) и 2). Именно этот вариант предоставляет возможность оптимизации плана (расписания) выполнения параллельной программы ($t_1 \le t \le t_2$).

“Вопрос на засы́пку” – как EXCEL определяет корректную последователь-ность выполнения сложных (располага-ющихся во мно́гих ячейках) выражений?

Оператор выполняется сразу, как только “готовы” все его операнды

Оператор выполняется тогда, когда результат операции становится *необходимым*

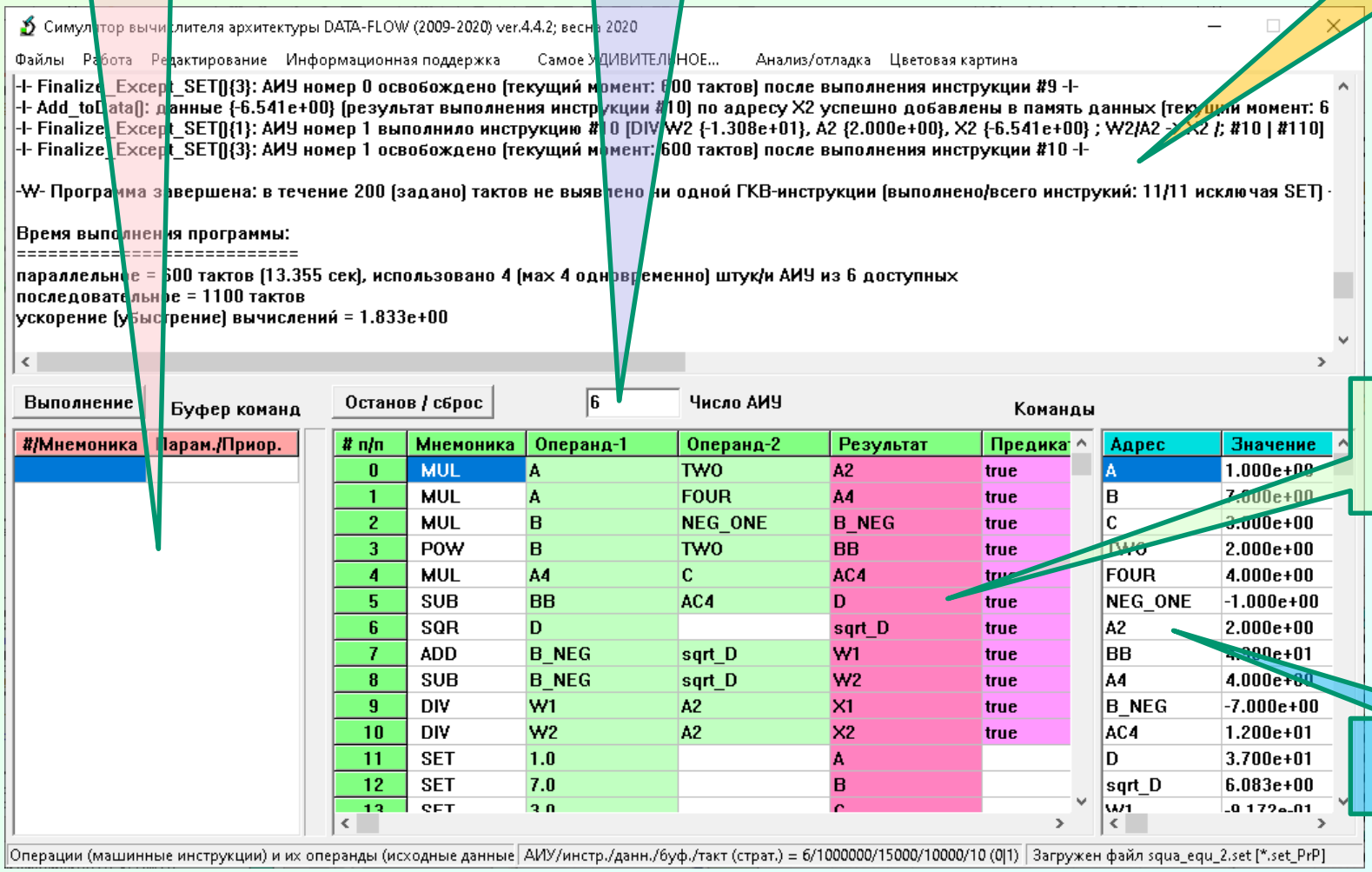


Текущая (версия “зима 2022 г.г.) реализация программы моделирования (*симуляции*) потокового вычислителя DATA_FLOW (платформа Win’32 бит, GUI)

Окно содержимого буфера команд

Число параллельных вычислителей

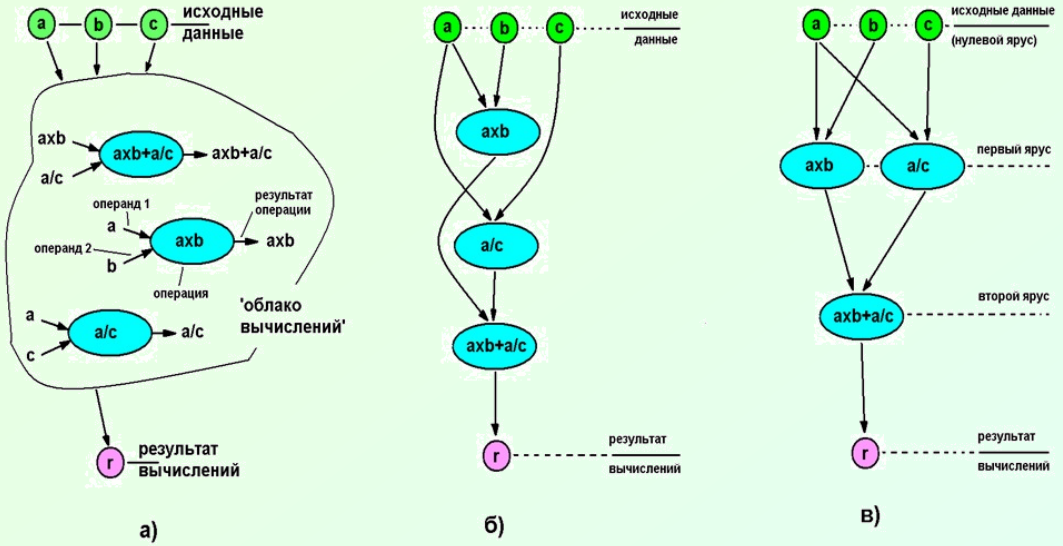
Окно вывода протокола решения задачи



Окно памяти исполняемых инструкций

Окно памяти данных

Работа с программной системой DATA_FLOW (анализ выполнения программы ABC_R.SET)



Назначение программы – вычисление по формуле r=a×b+a/c (исходные данные a, b, c)

Цель исследования – выявить наличие параллелизма в заданном формулой алгоритме.

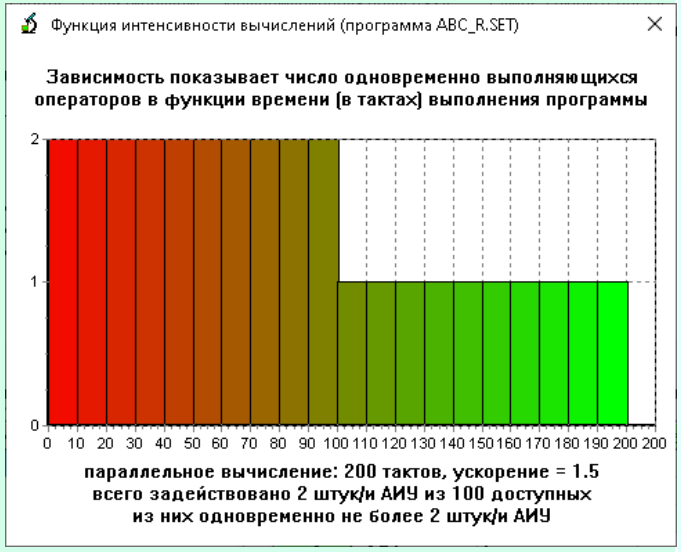
Запись программы для выполнения в системе DATA-FLOW (при условии A=1, B=2, C=3):

```
; Вычисление R = A×B + A/C
MUL A, B, AB ; AB ← A×B
DIV A, C, AC ; AC ←  A/C
ADD AB, AC, R ; R ← AB+AC
SET 1, A ; A ← 1
SET 2, B ; B ← 2
SET 3, C ; C ← C
```

Результат вычислений:

```
A    1    #3 [SET{1}, A ; A ← 1]
B    2    #4 [SET{2}, B ; B ← 2 ]
C    3    #5 [SET{3}, C ;  C ←  3]
AB   2    #0 [MUL A, B, AB ; AB ← A×B]
AC   0.333   #1 [DIV A, C, AC ; AC ← A/C]
R    2.333   #2 [ADD AB, AC, R ; R ← AB+AC]
```

График ИНТЕНСИВНОСТИ ВЫЧИСЛЕНИЙ (в функции времени решения задачи показывает число одновременно выполняющихся действий программы)



Использование предикатов для условного выполнения операторов (программа SQUA_EQU_2.PRED.SET)

```
MUL A, TWO, A2, !false ; A2 ← 2 * A
MUL A, FOUR, A4 ; A4 ← 4 * A
MUL B, NEG_ONE, B_NEG ;  B_NEG ← NEG_ONE * B
POW B, TWO, BB ; BB ←  B^2
MUL A4, C, AC4 ; AC4 ← A4 * C
SUB BB, AC4, D ;  D[iskriminant] ← BB - AC4
SQR D, sqrt_D, IS_re ; ← sqrt(D) -> sqrt_D
ADD B_NEG, sqrt_D, W1, IS_re ; W1 ← B_NEG + D_SQRT
SUB B_NEG, sqrt_D, W2, IS_re ; W2 ← B_NEG - D_SQRT
DIV W1, A2, re_X1, IS_re ; re_X1 ← W1/A2
DIV W2, A2, re_X2, IS_re ; re_X2 ← W2/A2
MUL D, NEG_ONE, NEG_D, !IS_re ; NEG_D ← NEG_ONE x D
SQR NEG_D, sqrt_D, !IS_re ; sqrt_D ← sqrt(NEG_D)
DIV B_NEG, A2, re_X1, !IS_re ; 1-th root (real)
DIV sqrt_D, A2, im_X1, !IS_re ; 1-th root (img)
CPY re_X1, re_X2, !IS_re ; 2-th root (real)
DIV sqrt_D, A2, W, !IS_re ; temp for im_X2
MUL W, NEG_ONE, im_X2, !IS_re ; 2-th root (im)
SET 1, A ; A ← 1
SET 3, B ; B ← 7/3 (re / im)
SET 3, C ; C ← 3
SET 2,  TWO  ; TWO ← 2
SET 4,  FOUR ; FOUR ← 4
SET -1, NEG_ONE ; NEG_ONE ← (-1)
SET 0, ZERO ; ZERO ← 0
PGE D, ZERO, IS_re ; IS_re ← true if D>=0
```

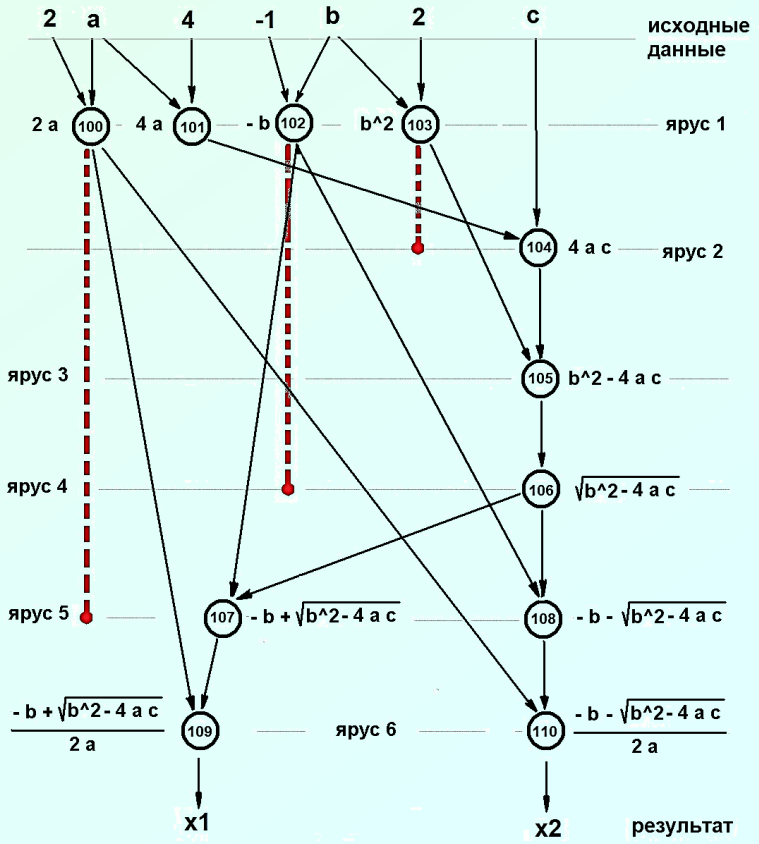
Использована команда PGE, устанавливающая флаг предиката IS_re в зависимости от *соотношения значений переменных **D** и **ZERO***

Пояснения. В качестве флага-предиката здесь используется **IS_re**. Командой **PGE D, ZERO, IS_re** флаг **IS_re** устанавливается в ‘true’ при условии D≥0 (здесь D – дискриминант полного квадратного уравнения) или в ‘false’ в противном случае; далее вычисления производятся в зависимости от значения флага **IS_re** (четвёртое поле команды)

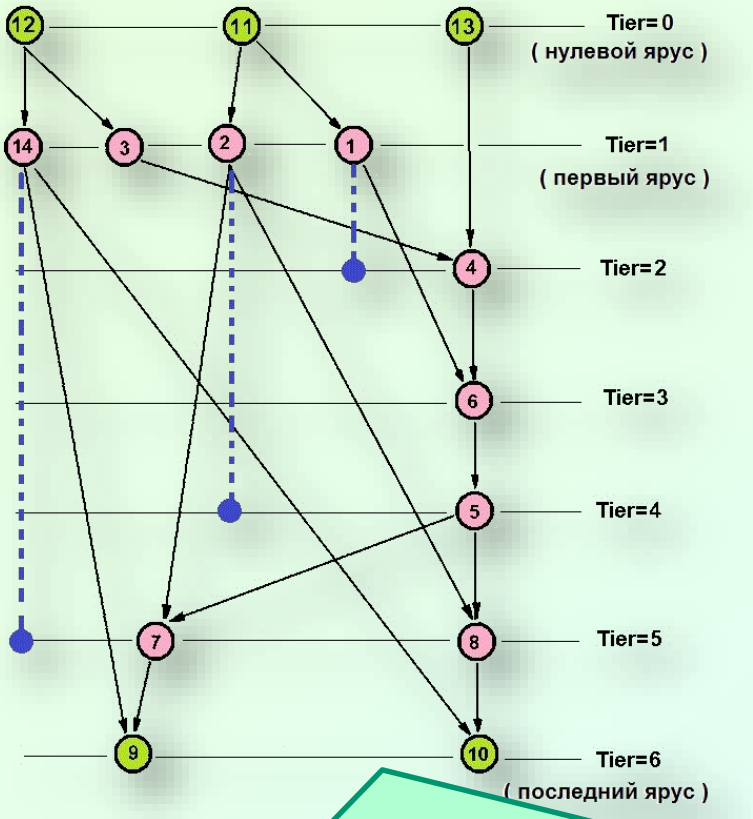
Получение, анализ и возможная реорганизация ЯПФ алгоритма (*программы*)

📖 Для выявления (скрытого) потенциала параллелизма используется метод представле́ния информационного графа алгоритма в специальном виде (сече́нии) – ярусно-параллельной форме (ЯПФ). **При этом на каждом ярусе располагаются операторы, зависимые (*по операндам*) только от операторов (*результатов их выполнения*), находящихся на ярусах *выше данного***. Вычислительная сложность получения ЯПФ $O(N^2)$, где N – общее число операторов (вершин графа).

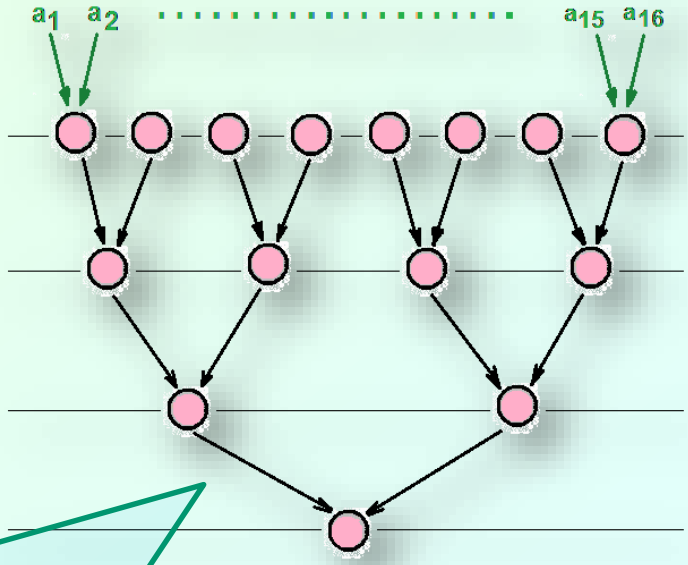
- ЯПФ информационного графа фактически является *начальным* ПЛАНОМ ВЫПОЛНЕНИЯ ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ (далее этот план может быть и усовершенствован).
- Насколько быстрее при этом выполнится программа (по сравнению с последовательным вариантом выполнения)? Ответ ясен – в 11/6 ≈ 1,83 раза!
- **Быстрее выполнить не получится, ибо число ярусов ≡ длина минимального пути в ИГА…**
- Что видим? На 1-м ярусе необходимо задействовать 4 параллельных вычислительных узла, на 5 и 6 – 2 узла, на 2,3,4 ярусах – по одному. **Такая неравномерность использования ресурсов – плохо..!**
- Если чуть подумать (*на рис. справа пунктиром показан диапазон допустимого расположения операторов по ярусам ЯПФ*), то станет ясно, что для выполнения данного алгоритма вполне достаточно и 2-х параллельно работающих вычислительных узлов (причём при этом *время параллельного выполнения остаётся неизменным*)…



Вариативность размещения операторов по ярусам в ярусно-параллельной форме (ЯПФ) графа



ось вре́мени выполнения алгоритма



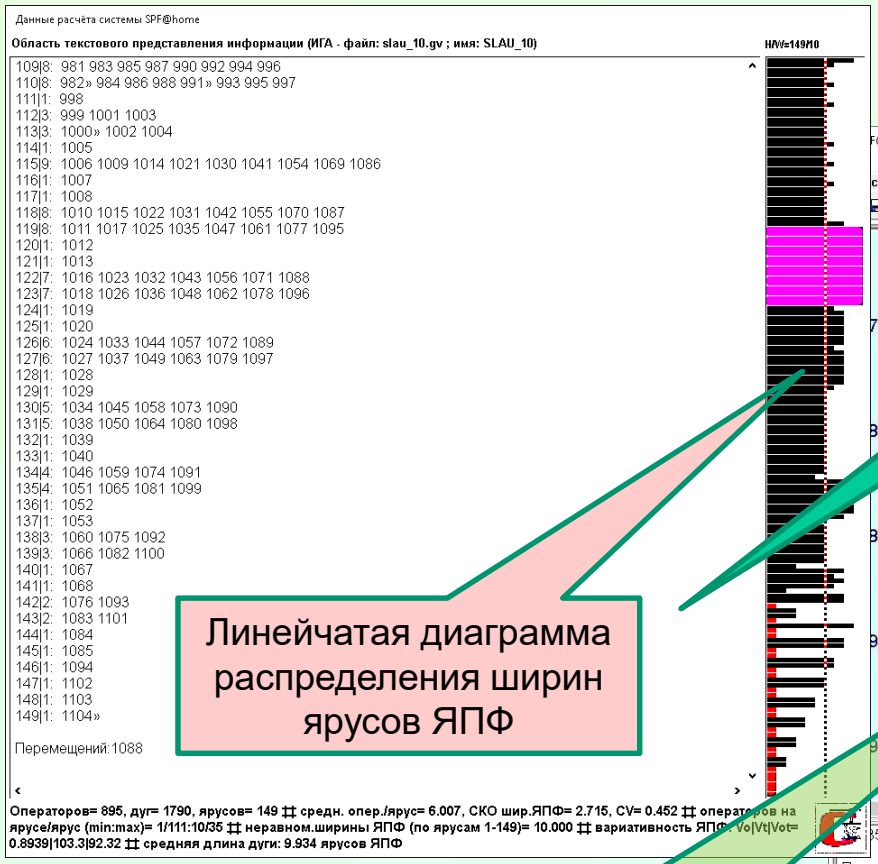
🛠 Показан граф операции вычисления ***методом сдва́ивания*** (справедлив для ассоциативных операций). Важно, что возможности перемещения операторов между ярусами нет (***вариативность нулевая***).

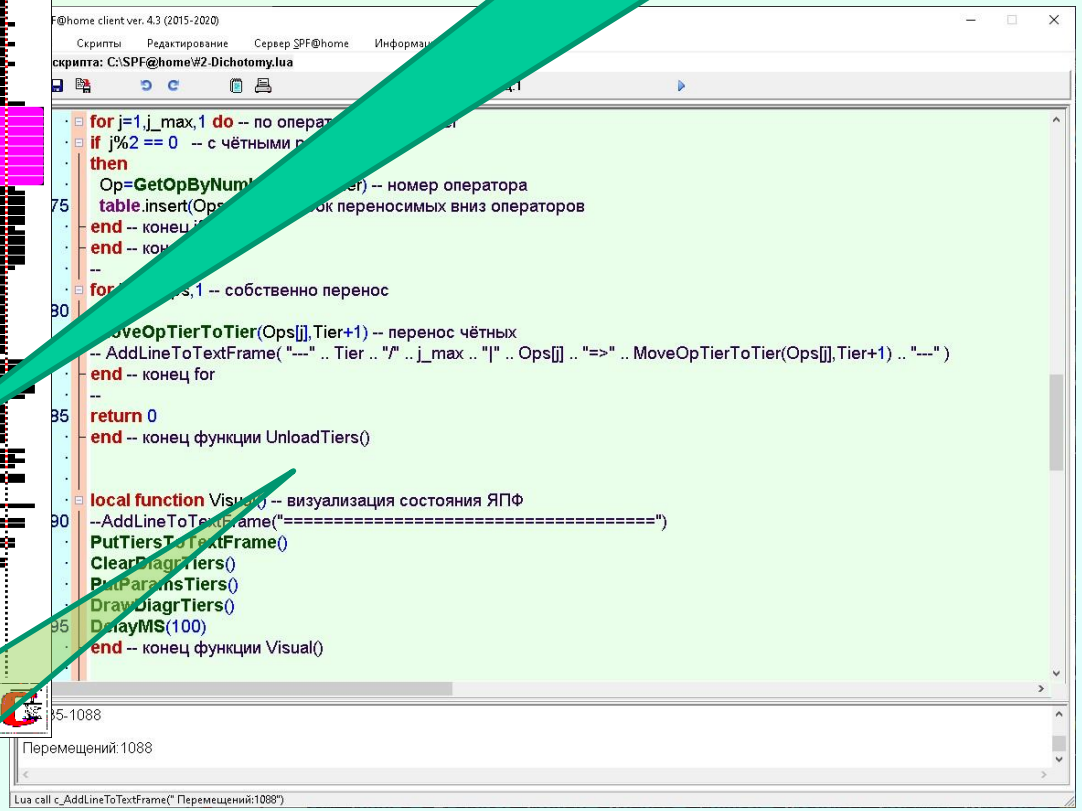
🛠 На рис. показан пример Информационного Графа Алгоритма (ИГА) в Ярусно-Параллельной Форме (ЯПФ), причём исходные данные подаются на входы 12, 11, 13, выходами являются 9, 10. На самом деле это граф программы решения полного квадратного уравнения в вещественных числах (впервые предложен индийским математиком Брахмагупта, 598-670 г.г. н.э.).

🛠 Здесь же синими линиями показан возможный диапазон перемещений операторов с яруса на ярус (***вариативность***). Нетрудно видеть, что благодаря “разгру́зке” 1-го яруса от операторов 14 и 2 данный алгоритм можно выполнить на двух *(вместо 4-х… **SIC !!!**)* параллельно работающих вычислителях !

Текущая реализация клиентской части проекта SPF@home (платформа Win'32, GUI)

Дочернее окно выдачи результатов расчётов в текстовом и графическом видах





Линейчатая диаграмма распределения ширин ярусов ЯПФ

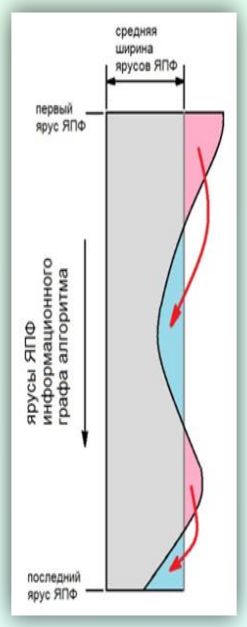
Главное окно разработки и отладки Lua-скриптов и управления их выполнением

Примеры реализации стратегий *целенаправленного изменения* ЯПФ

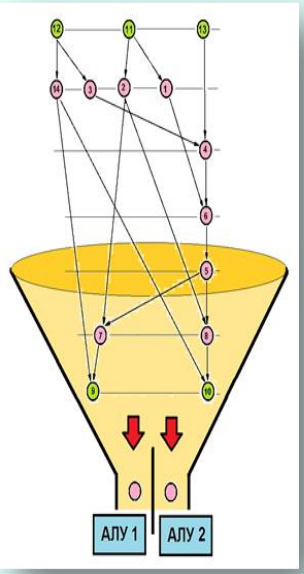
1. Задание – разработать план (расписание) параллельного выполнения программы с максимально равномерным распределением операторов по ярусам без увеличения высоты ЯПФ (оптимизация по критерию максимального использования ресурсов параллельного вычислителя при условии не увеличения времени выполнения программы).

2. Задание – разработать план (расписание) параллельного выполнения программы на заданном числе вычислителей при возможном возрастании высоты ЯПФ (оптимизация по использованию заданного количества вычислительных ресурсов при возможном увеличении времени выполнения программы).

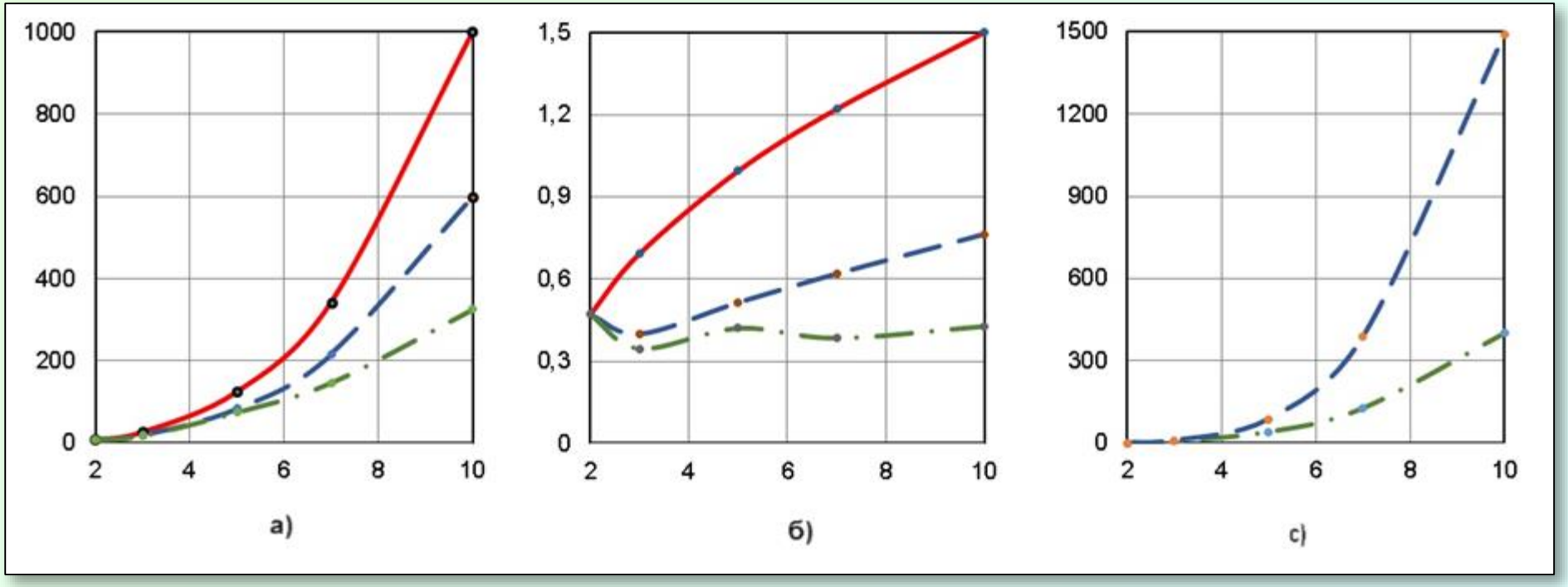
Представим ЯПФ в вертикальном направлении как поверхность земли. Поверхность не гладкая – имеются возвышенности и впадины. Наша цель – сгла́дить поверхность (по возможности привести ширину всех ярусов к среднеарифметической величине). ***Метафора*** – отва́л бульдозера сгреба́ет землю с холмов во впадины…



Нам надо “сжать” ЯПФ по ширине, получив величину не более заданной. ***Метафора*** - предста́вим ЯПФ в виде “куска" продукта, поступающего в мясорубку (размер её выходного конуса – число параллельных вычислителей). На выходе – иско́мое… Вариант с раска́ткой теста на доске также годится!..

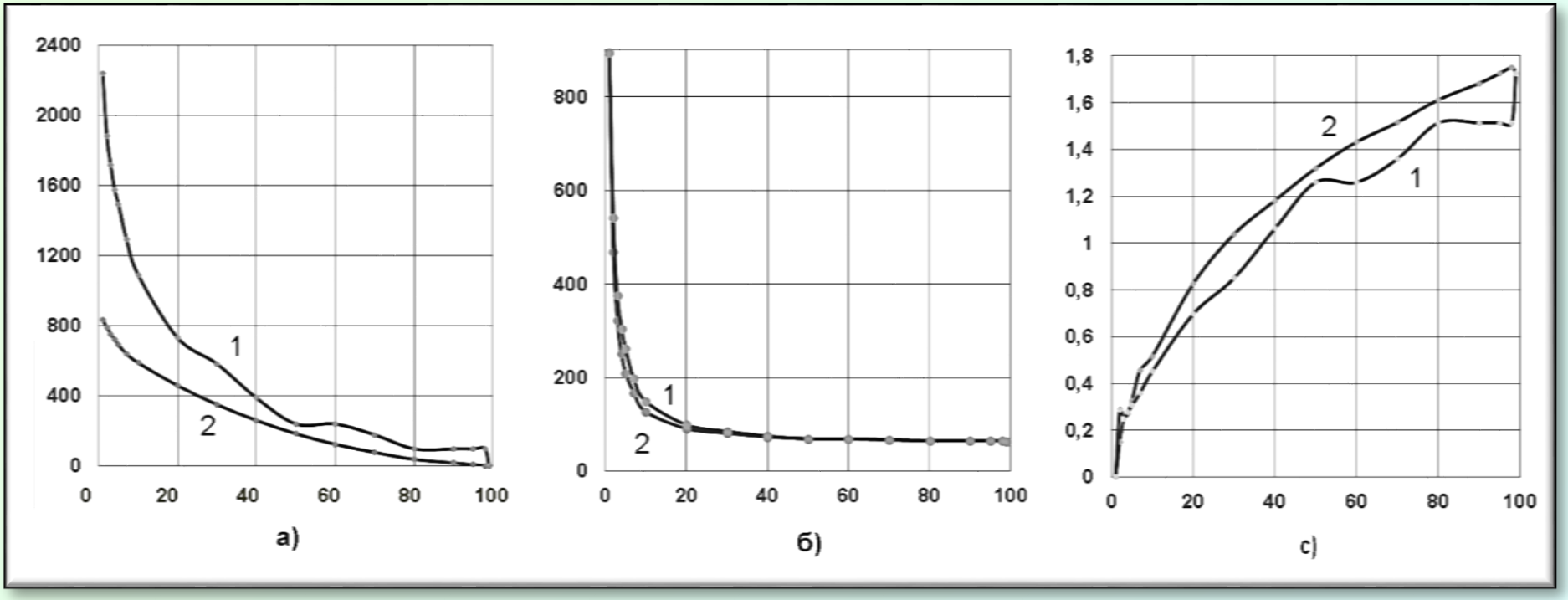


Пример исследования: сравнение эффективности сценариев построения плана выполнения программы при минимальном времени её выполнения (условие неувеличения высоты исходной ЯПФ)



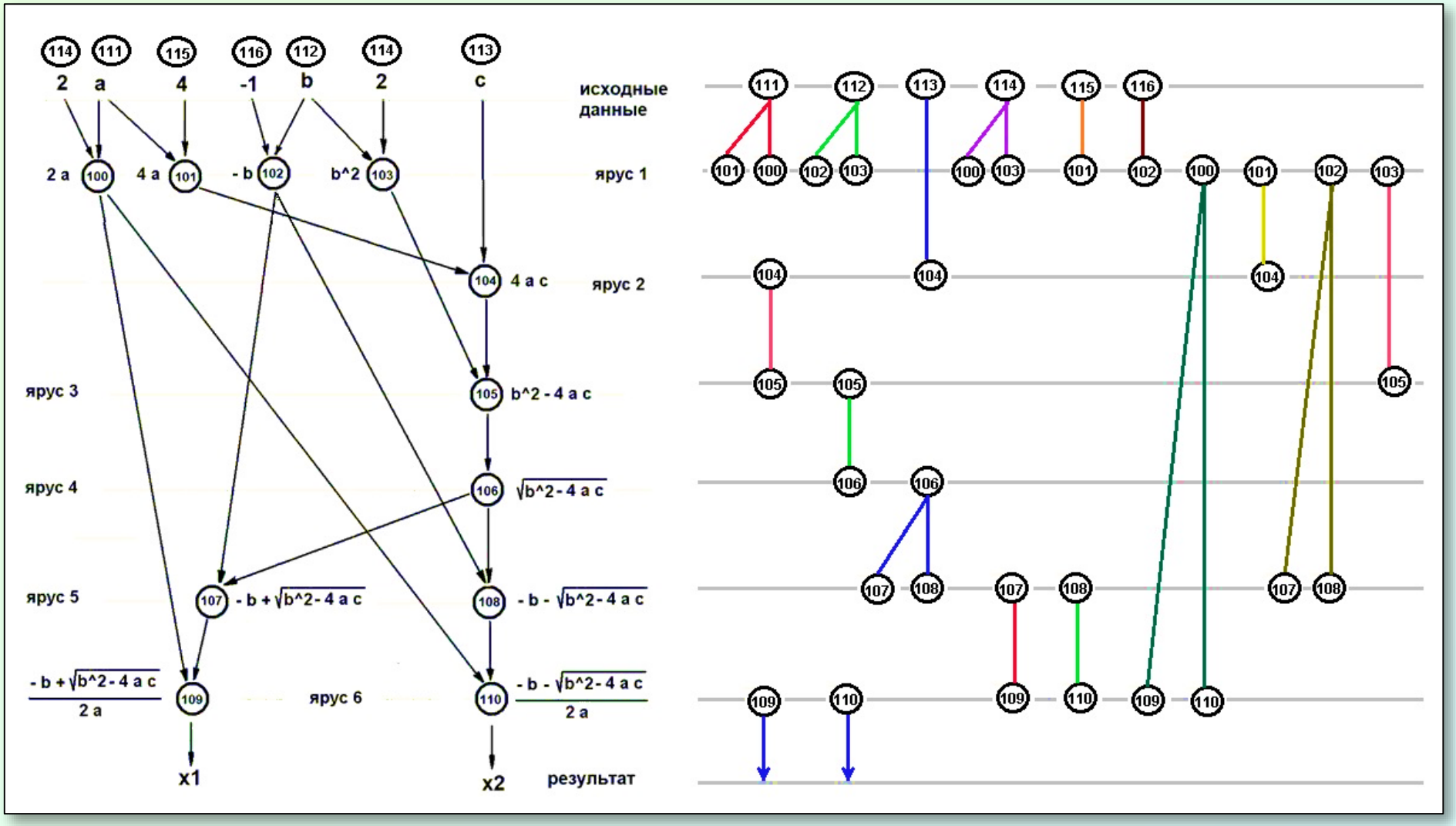
- Исследуемый алгоритм – умножение квадратных матриц порядков 2-10 (ось абсцисс) прямым (безитерационным) методом Гаусса
- На рисунках: а) – ширина ЯПФ, б) - коэффициент вариации (CV) ширин ярусов ЯПФ, с) - вычислительная сложность сценария (в единицах перемещения операторов между ярусами ЯПФ)
- Сплошные (красные) линии – исходная ЯПФ, пунктир (синяя) и штрих-пунктир (зелёная) – результат применения Lua-сценариев *01_bulldozer* и *02_bulldozer*

Пример исследования: сравнение эффективности сценариев построения плана выполнения программы на заданном числе параллельных вычислителей



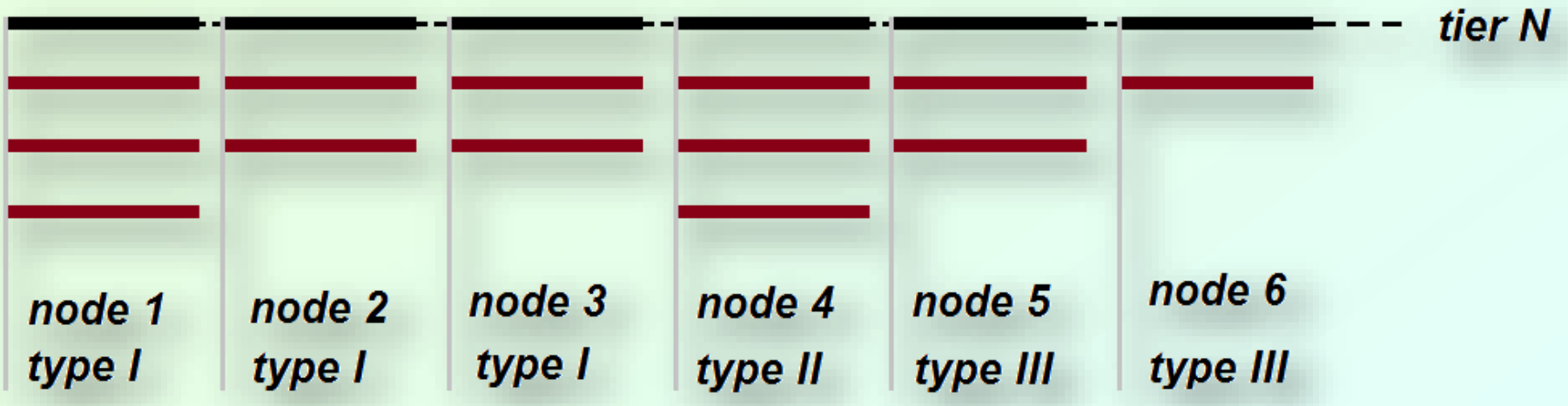
- Исследуемый алгоритм - решение систем линейных алгебраических уравнений (СЛАУ) 10-го порядка прямым (безитерационным) методом Гаусса
- На рисунках: а) – вычислительная сложность сценария (в единицах перемещения операторов между ярусами ЯПФ), б) - высота ЯПФ, с) - коэффициент вариации (*CV*) ширин ярусов ЯПФ (*CV*) в функции заданного количества параллельных вычислителей
- Преобразования ЯПФ проводились согласно Lua-сценариям *01_Strategy* и *02_Strategy* (кривые 1 и 2 соответственно)

Время жизни данных и проблема оптимизации использования регистров общего назначения (РОН) для передачи данных



🛠 В процессе работы алгоритма при срабатывании операторов создаются новые данные, которые затем используются иными операторами в качестве операндов (*правая часть рисунка*). Эти данные надо где-то (обычно в регистрах процессора – РОН) временно хранить. РОН всегда не хватает и поэтому стоит з***адача оптимизации использования РОН для временного хранения данных*** (см. цвет стрелок передачи данных)…

Метод образования подъярусóв при учёте гетерогенности поля параллельных вычислителей



- здесь *tier N* – ярус номер *N* для данной ЯПФ, *node* – конкретный вычислитель типа *type* заданного гетерогенного параллельного вычислительного поля.

В этом случае общее время *T* решения задачи суть сумма по всем ярусам максимальных значений времён выполнения операторов на подъярусах:

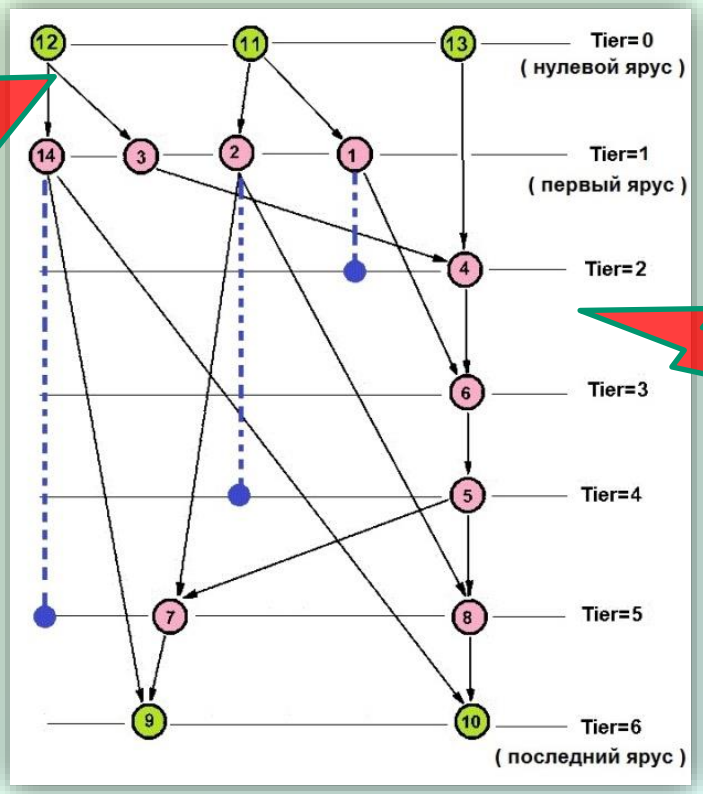
$$T = \sum_j \left(\max_{k_j} \sum_i t_{ik} \right)$$

где *j* - число ярусов, *i* - число подъярусов на данном ярусе, k_j - типы вычислителей на *j*-том ярусе, t_{ik} - время выполнения оператора типа *i* на вычислителе типа *k*.

Задача минимизации общего времени решения усложняется в случае возможности выполнения каждого оператора на нескольких вычислителях вследствие неоднозначности t_{ik} в вышеприведённом выражении; здесь необходима балансировка выполнения по подъярусам, реализуемая в процессе распределения операторов по типам вычислителей для каждого яруса ЯПФ.

Этапы работы с системой SPF@home *(к применению методов искусственного интеллекта при решении задачи)*

Получить (с помощью *информацион-ных функций*) данные о ЯПФ графа



Выполнить скрипт на Lua (акционные функции**), реализующий выбранную стратегию преобразования ЯПФ**

Распозна́ть ситуацию и выбрать наиболее эффективную стратегию преобразования ЯПФ для этой ситуации

Проект “Одома́шненный BEOWUF” (направление исследований: введе́ние технологий параллельных вычислений в быт – *персонализация кластерных технологий*)

Цель проекта – создание типовой модели дешевого (стоимость не более 200 $US на узел) общедоступного ***персонального*** **вычислительного кластера** для освоения принципов системного администрирования и различных технологий параллельного программирования).

Созданное оборудование включает вычислительный блок (размер 760×250×280 мм, вес ~17 кГ) из 5 стандартных системных плат формата µATX с процессорами Celeron 2,0 GHz, HDD 20 Gb, RAM 256 Mb, L2-кэш 256 kb и управляющую ПЭВМ (обычный персональный компьютер), операци-онная система – Slackware 9.1.

Программное обеспечение – библиотека MPI (в версии LAM), пакеты AZTEC и ScaLAPACK; устанавливается система DVM.

Внешний вид МВС (*вычислительный кла́стер*) кафедрального уровня (реализованный вариант)

Основные технические данные:

1. Всего вычислительных узлов – до 24 (размещение – по 6 узлов на каждой полке).
2. Монитор технического обслуживания (дисплей 10" и клавиатура) располагаются на второй полке сверху.
3. Максимальное энергопотребление – до 2 kW (ток 10 А при напряжении 220 вольт); тепловыделение до 7 Мега Джоулей/час).
4. В настоящее время МВС как вычислительный ресурс включена в локальную сеть кафедры КБ-5 МГУПИ / МИРЭА под IP-адресом **192.168.47.61**.

Список литературы

1. AlgoWiki. Открытая энциклопедия свойств алгоритмов. Под ред.: Воеводин В., Донгарра Дж. URL: http://algowiki-project.org (дата обращения: 15.04.2022).

2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2004. - 608 с.

3. Баканов В.М. Управление динамикой вычислений в процессорах потоковой архитектуры для различных типов алгоритмов. // Журнал “Программная инженерия", — М.: 2015, № 9, с. 20-24.

4. Федотов И.Е. Параллельное программирование. Модели и приёмы. — М.: СОЛОН-Пресс, 2018. - 390 с.

5. V.M.Bakanov. Software complex for modeling and optimization of program implementation on parallel calculation systems. Open Computer Science, 2018, 8, Issue 1, Pages 228–234, ISSN (Online) 2299-1093, DOI: 10.1515/comp-2018-0019/html

6. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. — М.: Мир, Книга по Требованию, 2012. — 420 с.

7. Иерузалимски Роберту. Программирование на языке Lua. — М.: ДМК Пресс, 2014. — 382 с.

InterNet – ресурсы по теме доклада

- **Математическая (компьютерная) модель вычислителя пото́ковой архитектуры для исследования произвольных алгоритмов на наличие внутреннего (*скрытого*) параллелизма и параметров его практического использования (Data-Flow) -**
  - **Описание здесь: http://vbakanov.ru/dataflow/**
  - **Инсталляция здесь: http://vbakanov.ru/dataflow/content//install_df.exe**
- **Программная система для исследования и выбора рациональных методов построения планов (*расписаний*) выполнения программ на заданном поле параллельных вычислителей (SPF@home) -**
  - **Описание здесь: http://vbakanov.ru/spf@home/**
  - **Инсталляция здесь: http://vbakanov.ru/spf@home/content/install_spf.exe**
- **О вычислительном кластере кафедры КБ-5 МИРЭА -**
  - **http://vbakanov.ru/hist_clu/clusters.htm**
- **Тематические статьи на Habr'е -**
  - **https://habr.com/ru/post/530078/** **https://habr.com/ru/post/534722/**
  - **https://habr.com/ru/post/535926/** **https://habr.com/ru/post/540122/**
  - **https://habr.com/ru/post/545498/** **https://habr.com/ru/post/551688/**

"Стихотворные" произведения о ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЯХ (*"околонаучный стёб"*) -

- **Размышления о необходимости параллельных вычислений**
- **О по́льзе пото́ковых (DATA-FLOW) вычислительных архитектур**