ИСПОЛЬЗОВАНИЕ МЕТОДА АНАЛИЗА СТРУКТУРЫ И ЦЕЛЕНАПРАВЛЕННЫХ ПРЕОБРАЗОВАНИЙ АЛГОРИТМОВ В ЗАДАЧАХ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

(опыт занятий в РТУ МИРЭА / НИУ ВШЭ в 2018÷2023 г.г.)

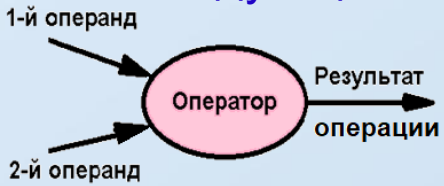
XX юбилейная конференция “Свободное программное обеспечение в высшей школе”. Тема: Использование свободного ПО в учебном процессе: разработка, внедрение, преподавание

Баканов Валерий Михайлович, НИУ ВШЭ
+7-915-053-5469, e881e@mail.ru, http://vbakanov.ru/left_1.htm

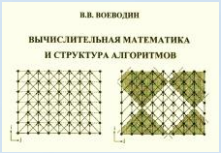
Vector–Magistratum

Воеводин Валентин Васильевич (1934÷2007), российский математик, академик РАН (2000)

Научная деятельность Вал.В.Воеводина была подчинена одной цели: постановке и решению различных математических проблем, связанных с эффективным решением прикладных задач на вычислительных системах различной (в т.ч. параллельной) архитектуры. Его профессиональные интересы связаны со следующими направлениями:



- разработка численных методов,
- ошибки округления и устойчивость,
- **информационная структура алгоритмов**,
- математические модели в вычислительных процессах,
- программное обеспечение для анализа информационной структуры программ (система V-Ray),
- **образование**.



Валентину Воеводину принадлежит формулировка “**теоремы об информационном покры́тии**” (https://agora.guru.ru/abrau2010/pdf/60.pdf, https://parallel.ru/sites/default/files/info/parallel/voevodin/voevodin.pdf). Иные научные работы: Воеводин В.В. Математические модели и методы в параллельных процессах. Наука, 1986. – 296 стр. // Воеводин В.В. Математические основы параллельных вычислений. Изд-во МГУ, 1991. – 345 стр.

Поспелов Гермоген Сергеевич (1914÷1998), российский математик, академик РАН (1984)

Г.С.Поспелов - советский учёный в области автоматического управления, основоположник отечественной школы методов искусственного интеллекта, генерал-майор-инженер. Окончил МГУ, работал в МЭИ, МФТИ; выпустил 20 монографий и более 300 статей.

Под руководством и при непосредственном участии Г.С.Поспелова были внедрены диалоговые и интеллектуальные системы перспективного планирования развития отраслей и комплексов отраслей промышленности (в частности, совместные разработки ВЦ АН СССР и институтов Миноборонпрома, Минрадиопрома СССР и институтов Минмаша СССР). Эти работы были отмечены решением Президиума АН СССР и медалями ВДНХ.

Поспелов разработал комплекс совершенно новых методик построения систем управления на принципе семиотических (логико-лингвистических) моделей представления объекта управления и описания процедур управления ими. Учёный выступил автором аппарата ***ярусно-параллельных форм для решения проблем с организацией параллельных вычислений*** в вычислительных комплексах и сетях. Поспелов заложил основы нового научного направления в области инженерии знаний и моделирования рассуждений специалистов-экспертов, принимающих решения в предметных областях.

Логический стек этапов процесса *количест-венного* моделирования явлений природы

АЛГОРИТМ является **промежуточной стадией** в процессе моделирования между **математической моделью** и созданием **исходного кода программы (и первой стадией, допуска́ющей количественный анализ!)**, находится в логическом стеке этапов количественного моделирования **глубже этапа разработки исходного кода**; соответственно выявле́ние потенциала параллелизма на этом уровне должно́ показать максимальные значения.

Вывод – для выявле́ния максимума потенциала параллелизма в заданном алгоритме *(задача вида “куда стреми́ться?”)* необходимо анализировать именно АЛГОРИТМ..!

Планирование обработки данных при создании расписаний параллельного выполнения алгоритмов (Прихожий А.А., БНТУ)

По структуре разрабатываемого плана

- Обыкновенный (*ordinary*) - каждая операция выполняется на одном шаге управления операции из одного шага выполняются параллельно, а
- Многошаговый (*multi cycling*) - одна операция может выполняться на нескольких соседних шагах управления, а в пределах одного шага все
- Цепочечный (*chaining*) - операции могут образовывать цепочки и выполняться последовательно в переделах одного шага, операции из разных цепочек выполняются в пределах одного шага
- Конвейерный (*pipelining*) - план разбивается на ступени конвейера, обрабатывающие различные наборы данных параллельно во времени

По решаемой оптимизационной задаче

- С ограничением на время реализации плана (*time-constrained*) и минимизацией объёма потребляемых вычислительных ресурсов
- С ограничением на объём потребляемых вычислительных ресурсов (*resource constrained*) и минимизацией времени реализации плана
- С ограничениями на время реализации плана и объём потребляемых вычислительных ресурсов – задача на достижимость (*feasible-constrained*)

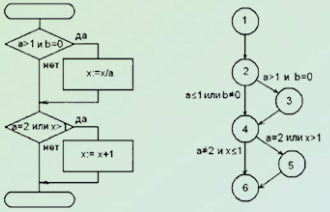
По стратегии планирования

- Назначение операции на наиболее ранний шаг управления (*As Soon As Possible – ASAP*)
- Назначение операции на наиболее поздний шаг управления (*As Late As Possible – ALAP*)
- Спи́сковое планирование (*List Scheduling – LS*)
- Сведе́ние задачи планирования к задаче целочисленного линейного программирования (*Integer Linear Programming Formulation – ILPF*)
- Сведе́ние задачи планирования к свёртыванию графа распараллеленности операций, метод сечений информационного графа и другие стратегии

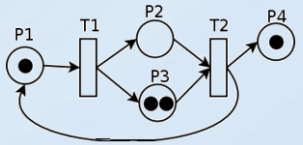
Может быть использовано в качестве индивидуального задания для студентов

Алгори́тмико-ориентированный VS программно-ориентированный подходы к исследованию программ

Алгоритмико-ориентированный подход



Программно-ориентированный подход

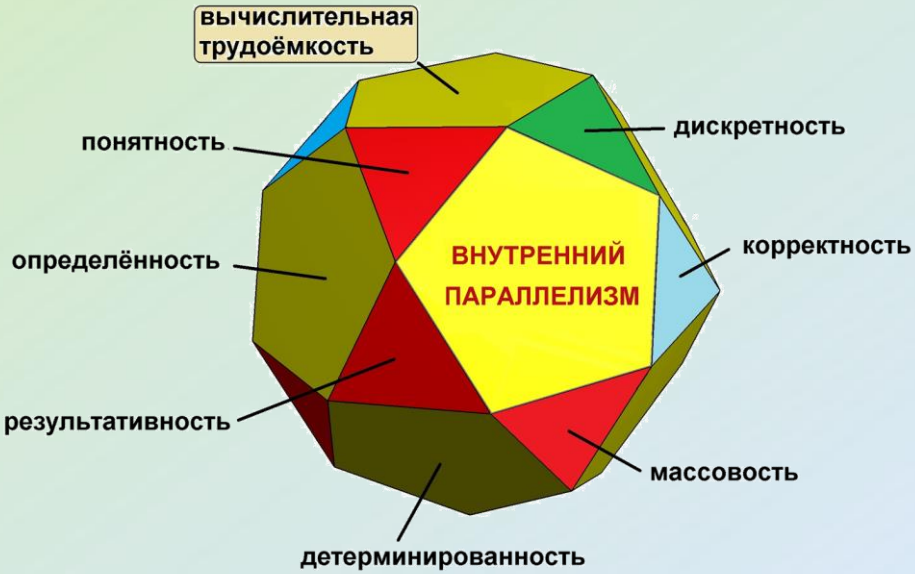


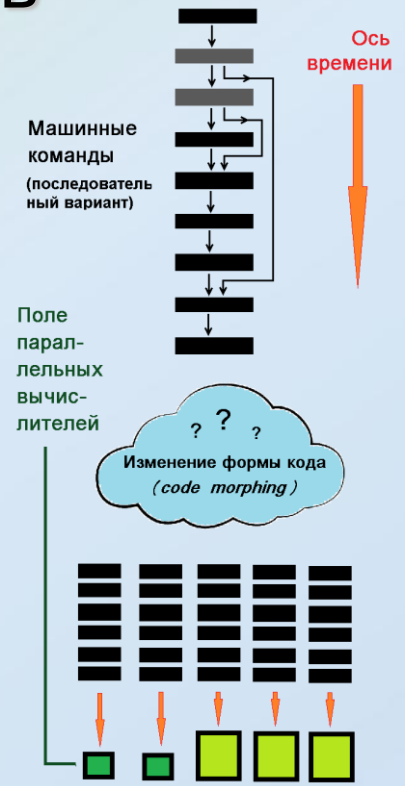
- Абстракция от параметров, незна́чимых для данного представле́ния - напр., *параметров времени* (до́ступа к памяти, выполнения машинных команд); *при анализе отдельные **действия алгоритма не выполняются**, но анализируются **информационные связи** между действиями*
- Возможность представле́ния алгоритма ***в форме графа*** с последующей стандартной обработкой (системы METIS/ParMETIS и др.)
- Поддерживается WEB-сайт AlgoWiki под руководством Вл.Воеводина (сын академика) и Дж. Донгарра (автор теста HPL производительности суперкомпьютеров); система анализа AlgoView

- Точное покома́ндное выполнение машинного кода в системах Simics, Ski, вариантах сетей Петри etc
- В особых случаях (напр., параллельная обработка) всё равно требуется предварительное составление плана выполнения программы (а это затруднительно сделать без привлечения методов АОП)
- Длительная обработка с выдачей огромного количества второстепенной информации

Выводы: алгоритмический подход базируется *на более глубоких* (относительно программной симуляции) *принципах информатики* и поэтому *обладает бо́льшей гибкостью* в применении.

Нахождение потенциала параллелизма и целенаправленные эквивалентные преобразовании алгоритмов

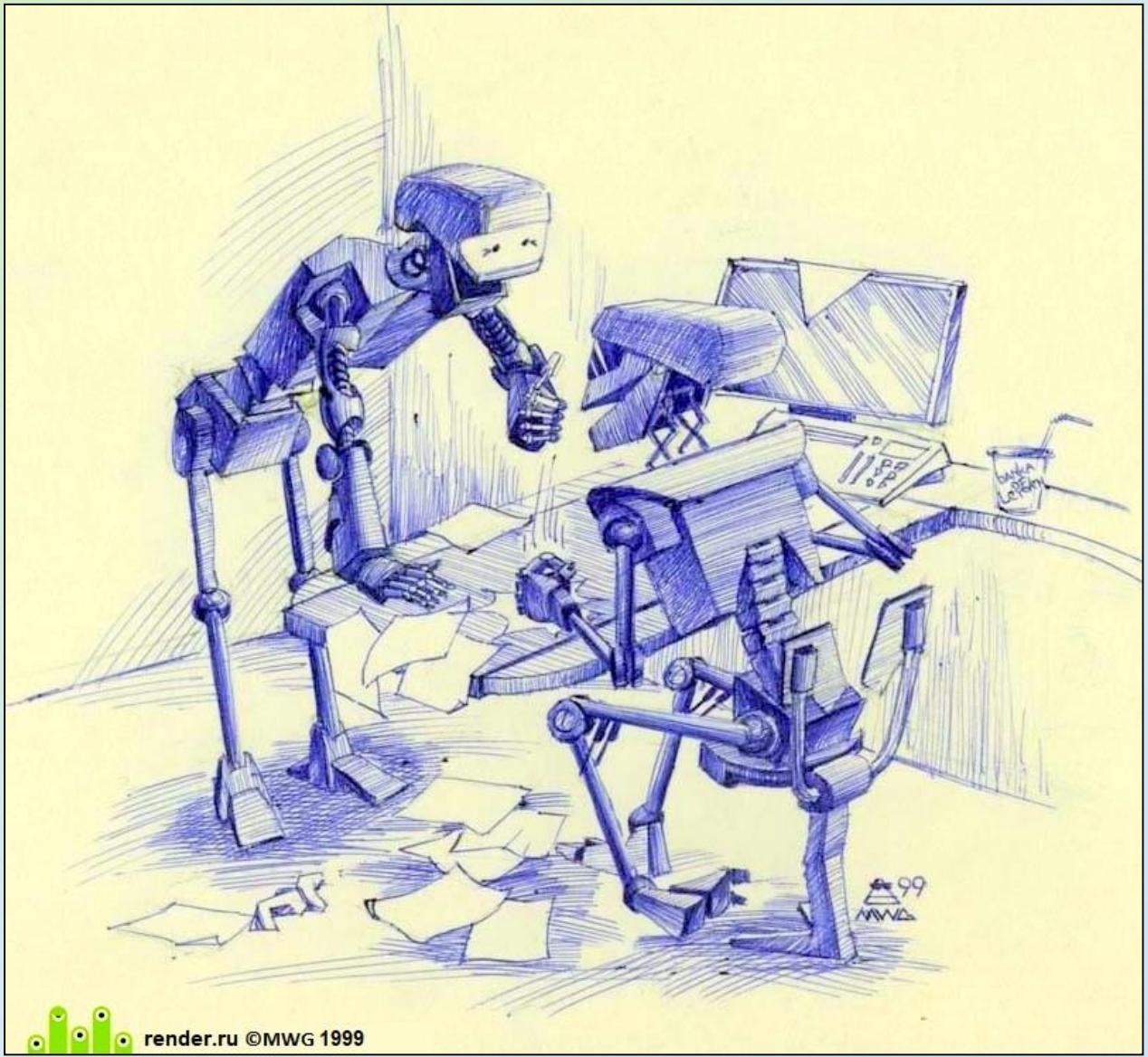




Нахождение потенциала внутреннего (скрытого) параллелизма в произвольных алгоритмах

Целенаправленные эквивалентные (не изменяющие информационных связей) преобразования алгоритмов

“Великие механи́кусы” Трурль и Клапау́ций о роли **АЛГОРИТМА в жизни и творчестве**



📖 – По правде, – бурчал Трурль, – надо бы как-то иначе скомбинировать… Впрочем, **важнее всего АЛГОРИТМ!**

– Тоже мне открытие сделал! Известно, **без АЛГОРИТМА ни шагу ступить!** Ну не́чего, надо экспериментировать!

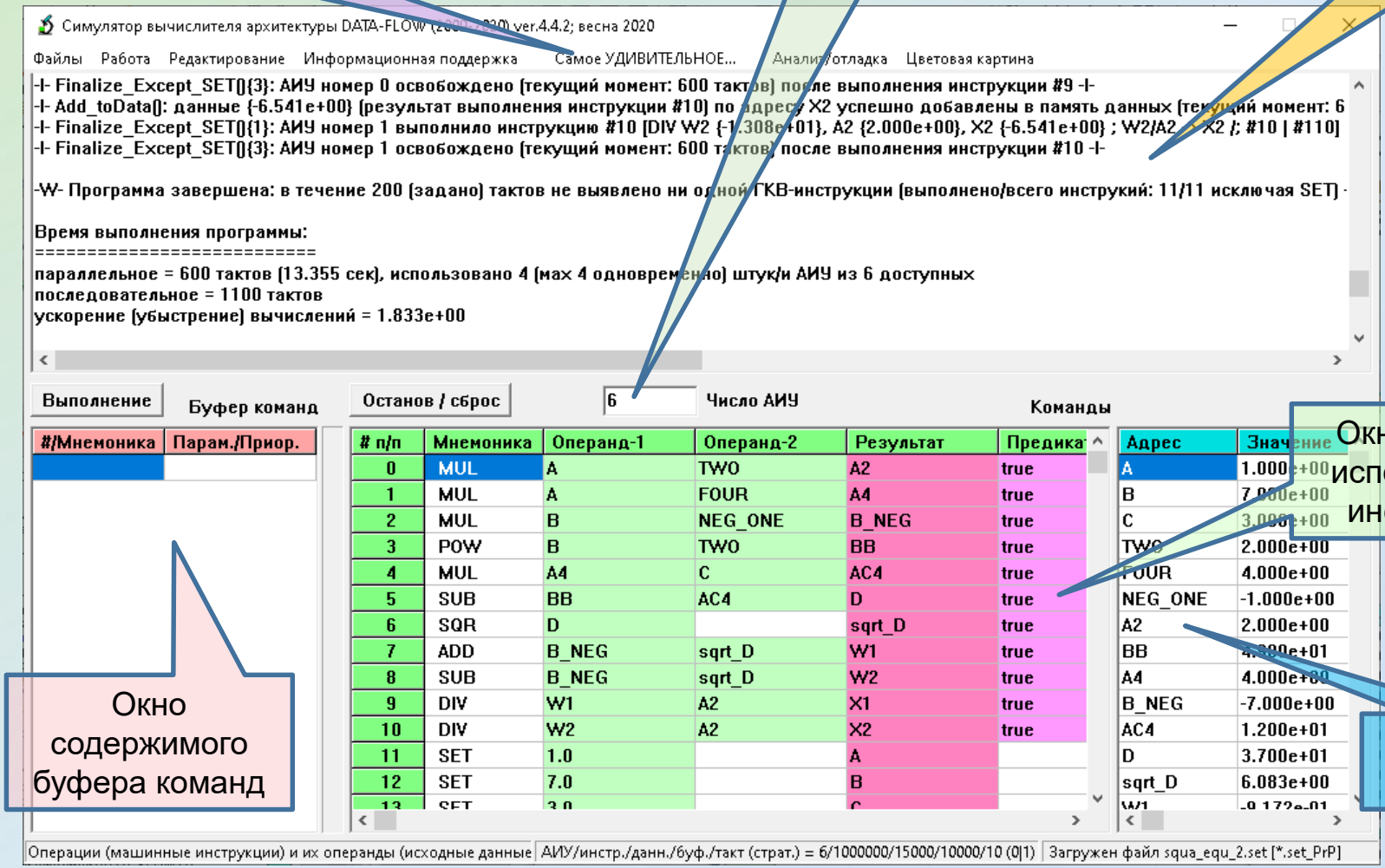
”Семь путешествий Трурля и Клапауция. Путешествие второе, или Какую услугу оказали Трурль и Клапауций царю Жестокусу”. Станислав Лем, КИБЕРИа́ДА. 1965÷1976.

Модель ГРАФ-МАШИНЫ (вычислитель архитектуры Data-Flow) для исследования параметров аппаратного распараллеливания

Кнопка перемешивания операторов программы в случайном порядке

Число параллельных вычислителей

Окно вывода протокола решения задачи



Окно памяти исполняемых инструкций

Окно содержимого буфера команд

Окно памяти данных

Использование предикáтов 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***

📖 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 – необязательное - поле оператора).

🛠 Применение предикатов позволяет ликвидировать ограничения в линейности используемых алгоритмов без применения *регистра-счётчика команд.* Предикатный метод даёт возможность уйти от (крайне высокозатрáтной) практики ***предсказáния переходов*** в программах!

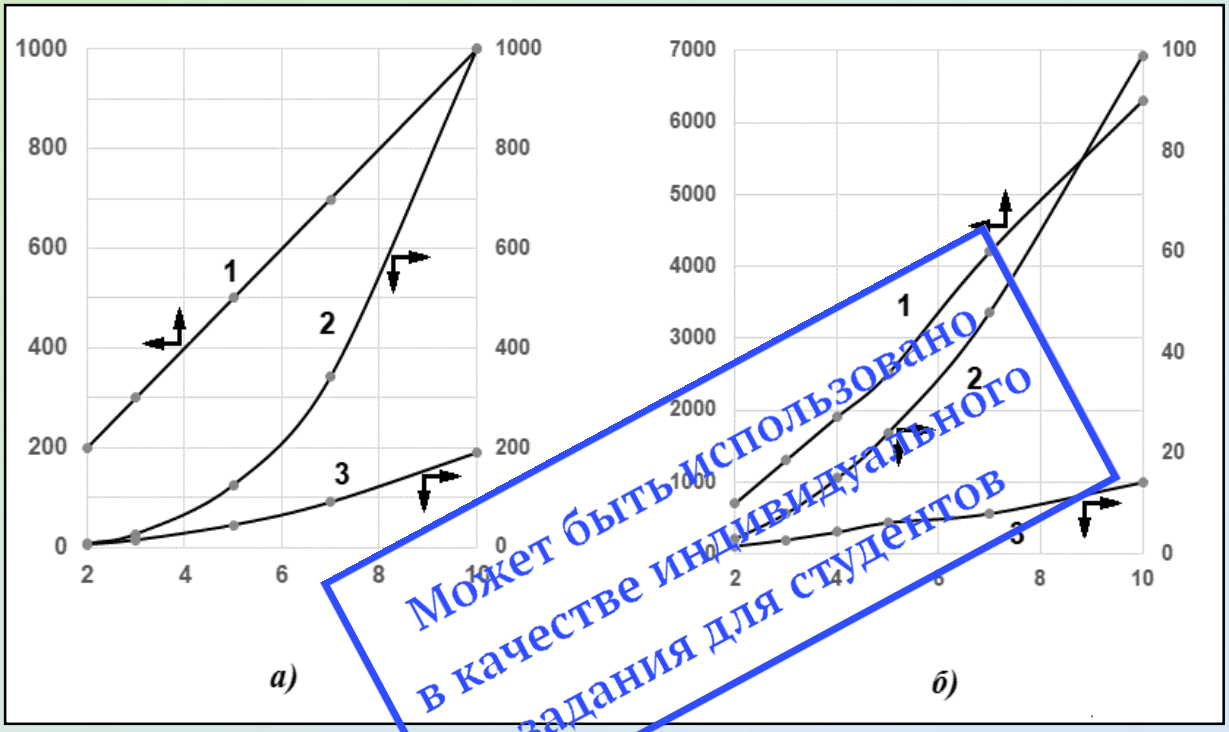
Библиотека алгоритмов для исследований на наличие параллелизма и его параметров

Умножение квадратных матриц классическим способом (общее название m_matr_N.set, где *N* – порядок матриц)	m_matr_2.set, m_matr_3.set, m_matr_5.set, m_matr_7.set, m_matr_10.set
Умножение квадратной матрицы на вектор (общее название m_matr_vec_N.set, где *N* – порядок матриц)	m_matr_vec_2.set, m_matr_vec_3.set m_matr_vec_5.set, m_matr_vec_7.set, m_matr_vec_10.set
Решение систем линейных алгебраических выражение простым (безитерационным) методом Гаусса (общее название slau_N.set, где *N* – порядок системы)	slau_2.set, slau_3.set, slau_4.set, slau_5.set, slau_7.set, slau_10.set
Аппроксимация точек прямой методом наименьших квадратов (общее название mnk_N.set, где *N* – число точек при аппроксимации)	mnk-5.set, mnk-10.set, mnk-15.set, mnk-20.set
Аппроксимация точек параболой методом наименьших квадратов (общее название mnk-2_N.set, где *N* – число точек при аппроксимации)	mnk-2_5.set, mnk-2_10.set, mnk-2_15.set, mnk-2_20.set
Вычисление коэффициента парной корреляции (общее название korr_N.set, где *N* – число точек)	korr_5.set, korr_10.set, korr_15.set, korr_20.set
Вычисление первых 10 чисел Фибоначчи, “трибоначчи”, “квадроначчи”	fibonn_10.set, tribonn_10.set, quadronn_10.set
Вычисление полинома 10-го порядка (метод прямого возведения чисел в степень, метод возведения в степень последовательным умножением)	polinom_10-1.set, polinom_10-2.set
Сложение 32 чисел методом сдва́ивания	doubling_32.set
Нахождение максимума чисел в одномерном массиве из 8 чисел (метод последовательного сравнения пар чисел, алгоритм сдва́ивания)	max-1_mass-8.pred.set, max-2_mass-8.pred.set
Нахождение корней полного квадратного уравнения (только действительные корни, комплексные корни)	squa_equ_2.set, squa_equ_2.pred.set

🛠 Цель 1: создать у обучаемых связь между конкретным алгоритмом и параметрами (в ЯПФ-форме информационного графа) его параллельного выполнения.

🛠 Цель 2: показать наличие и характер зависимости параметров алгоритма от ***размеров обрабатываемых данных*** (обычно ***порядка матриц***).

Пример анализа параметров параллельного выполнения программ на различном числе вычислителей



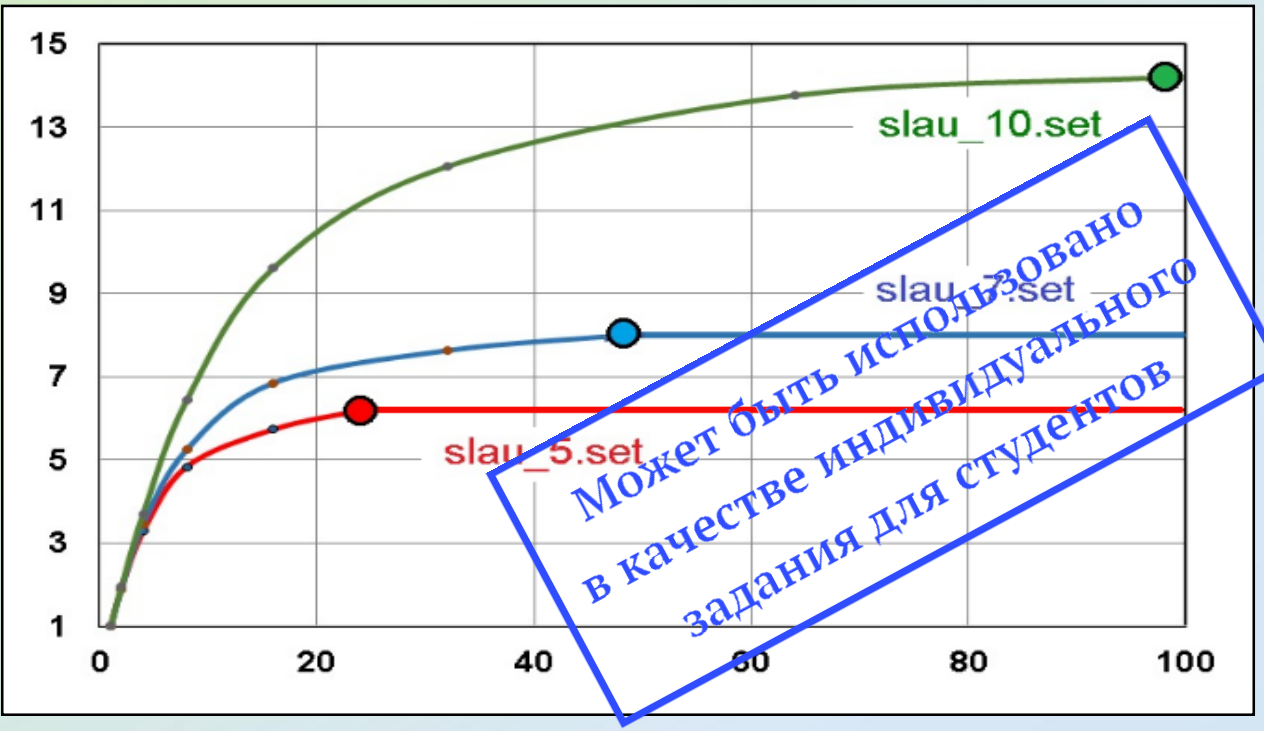
Время выполнения алгоритма является практически линейной функцией от размеров данных, что благоприятно для вычислительной практики. Минимально необходимое для реализации наискорейшего выполнения алгоритма число вычислителей растёт быстрее линейной зависимости, что невыгодно в смысле роста затрат на оборудование. Ускорение вычислений растёт быстрее линейной зависимости (что вы́игрышно).

Программа алгоритма умножения матриц традиционным способом – ***а)*** и решения СЛАУ безитерационным методом Гаусса – ***б)***

Расчёт проведён для *условий неограниченного параллелизма*

Ось абсцисс – размер обрабатываемых данных (порядок матриц); **1** - параллельная сложность алгоритма, **2** – минимально требуемое для реализации наискорейшего выполнения число вычислителей, **3** – ускорение вычислений

Пример анализа параметров параллельного выполнения программы решения систем линейных алгебраических уравнений (СЛАУ) на различном числе вычислителей



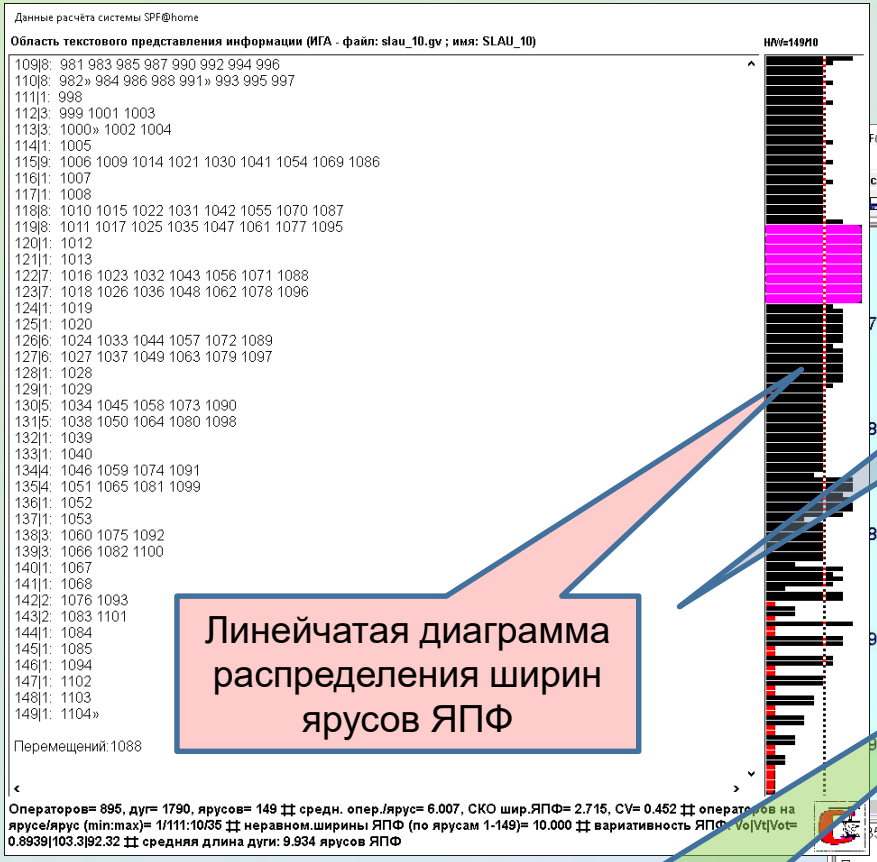
Анализ данных говорит о: а) - ограни́ченности ускорения вычислений при достижении определённого количества вычислителей и б) - возможности значительного сокращения числа вычислителей без существенного снижения производительности (благодаря форме *кривых с насыще́нием*)

Изменение ускорения вычислений (*ось ординат*) от количества параллельных вычислителей (*ось абсцисс*) для алгоритма решения СЛАУ при различном размере обрабатываемых данных (программы slau_NN.set, где NN – порядок уравнении; данные компьютерного моделирования)

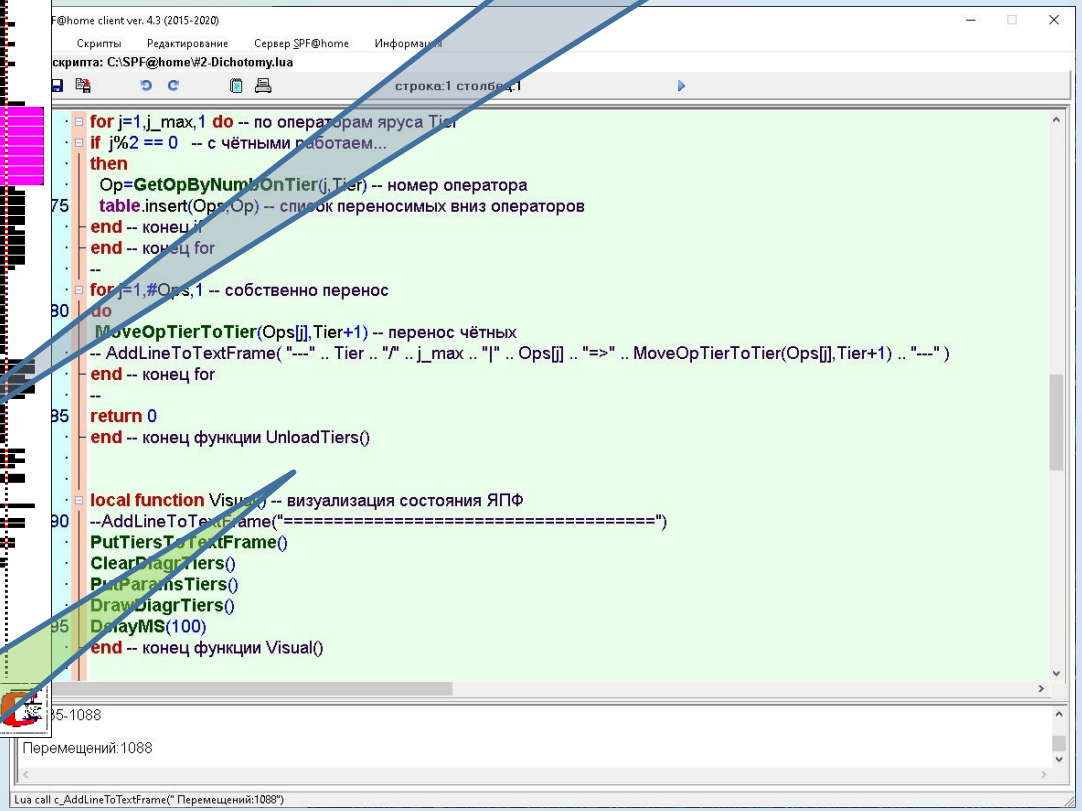
Расчёт проведён для условий неограниченного параллелизма

Программная система SPF@home для построения рациональных планов параллельного выполнения алгоритмов

Дочернее окно выдачи результатов расчётов в текстовом и графическом видах

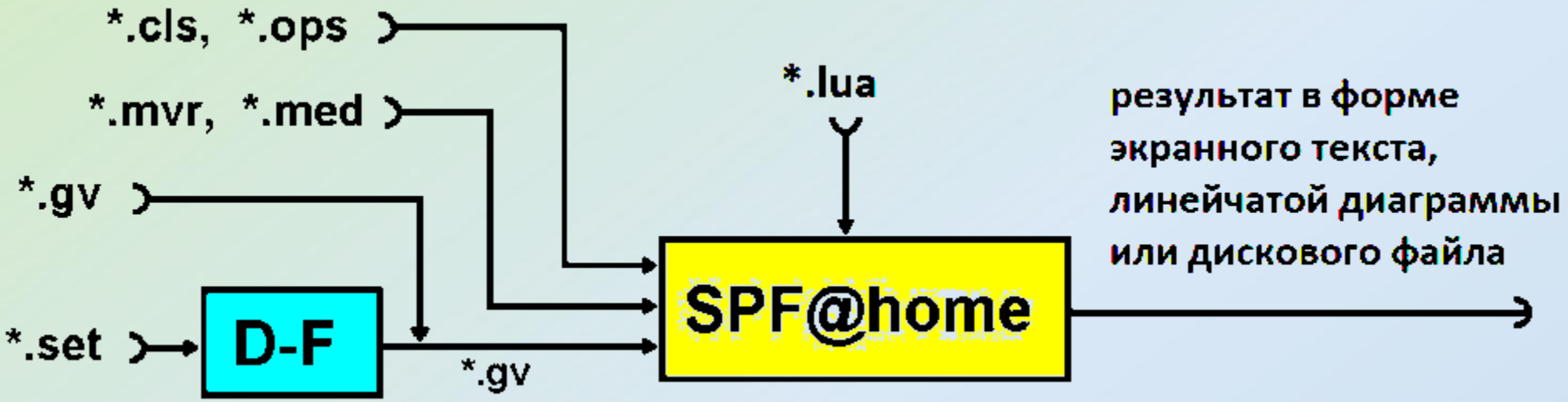


Линейчатая диаграмма распределения ширин ярусов ЯПФ



Главное окно разработки и отладки Lua-скриптóв (“’**эвристик**”) и управления их выполнением

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



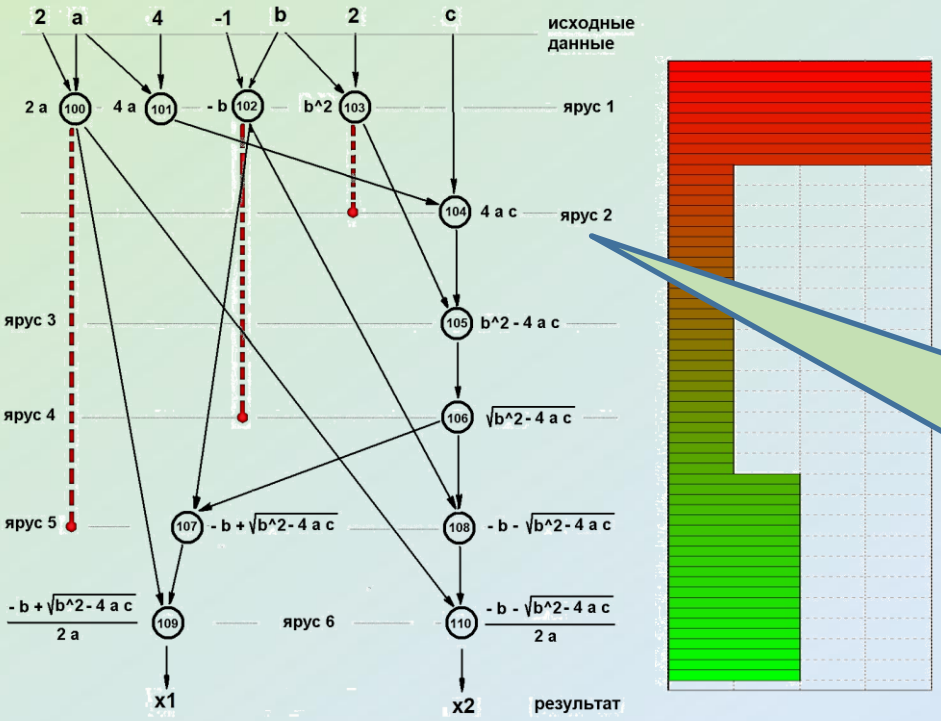
*.set и *.gv – программный файл и файл информационного графа анализируемой программы соответственно,

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

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

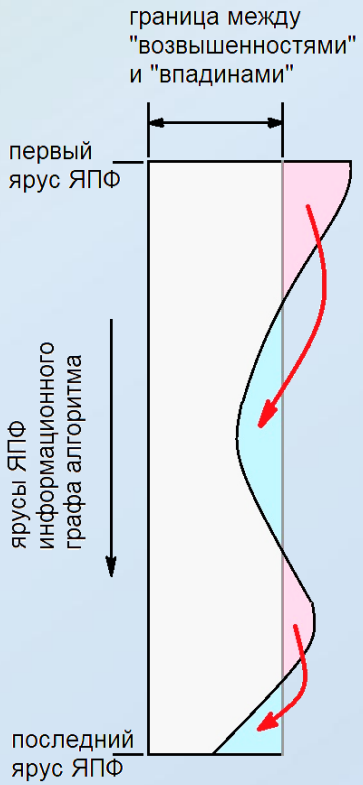
*.lua – текстовый файл на языке Lua, содержащий методы реорганизации ЯПФ

Целенаправленные эквивалентные преобразования ЯПФ информационного графа алгоритма



Вектор времени

Каждый ярус ЯПФ может быть сопоста́влён со списком команд сверхдлинного машинного слова VLIW-процессора



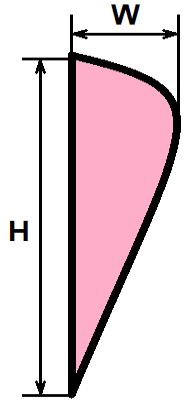
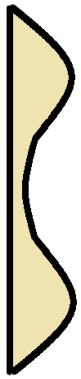
📖 ЯПФ информационного графа алгоритма решения полного квадратного уравнения в вещественных числах

🛠 Красным пунктиром показано возможное расположение операторов после целенаправленного эквивалентного (*не изменяющего информационные связи в алгоритме*) изменения ЯПФ графа

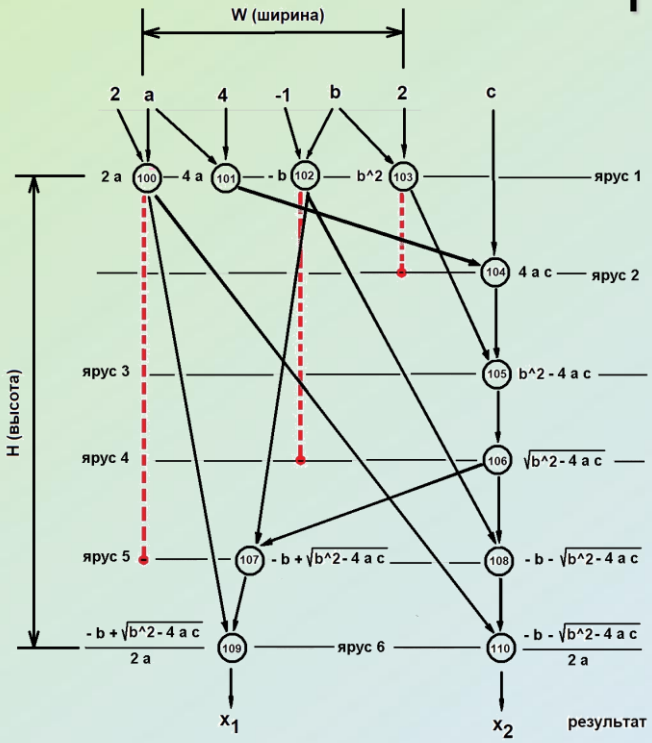
📖 Иллюстрация применения одного из эмпирических методов (“*эмпирик*”) целенаправленных преобразований ЯПФ графа (символическое название “BULLDOZER”)

🛠 Метафора метода – отва́л бульдозера перемещается вдоль ЯПФ графа, сдвига́я при этом почву с возвышенностей во впа́дины (границей между ними является за́данная ширина ЯПФ)

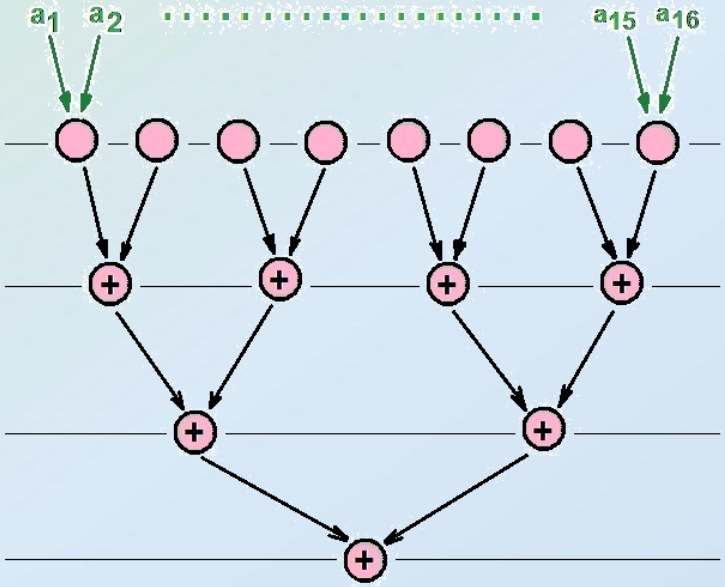
Классификация ЯПФ информационных графов алгоритмов *по их очерта́нию (в целях определения наивыгоднейшего метода целенаправленной модификации)*

Конфигу-рация (форма) ЯПФ					
Описание	Начало ши́ре (*begin wider*)	Конец ши́ре (*ending wider*)	Середина ши́ре (*middle wider*)	Двух-го́рбый (*two-humped*)	Много-го́рбый (*multy-humped*)
Наивыгодней-ший метод целенаправ-ленной моди-фикации ЯПФ	Сгла́жива-ние с начала к концу (“сверху вниз”)	Сгла́жива-ние с конца к началу (“снизу вверх”)	Безраз-лично (оба равноценны)	Безраз-лично	Безраз-лично

Параметры параллелизма некоторых графов алгоритмов



Вектор времени



📖 Ярусно-параллельная Форма (ЯПФ) графа алгоритма решения полного квадратного уравнения в вещественных числах.

Основные свойства:

🛠 W=4 – ширина ЯПФ (режим **неограниченного параллелизма**)

🛠 H=6 – высота ЯПФ (то же)

Допускаемые модификации:

💣 Возможно снижение ширины ЯПФ без увеличение высоты

💣 Возможно снижение ширины ЯПФ до единичной (последовательное выполнение)

📖 ЯПФ графа алгоритма сдва́ивания (примени́м к любым ассоциативным операциями).

Основные свойства:

🛠 высота = $\log_2(N)$, число суммируемых чисел

🛠 ширина = N

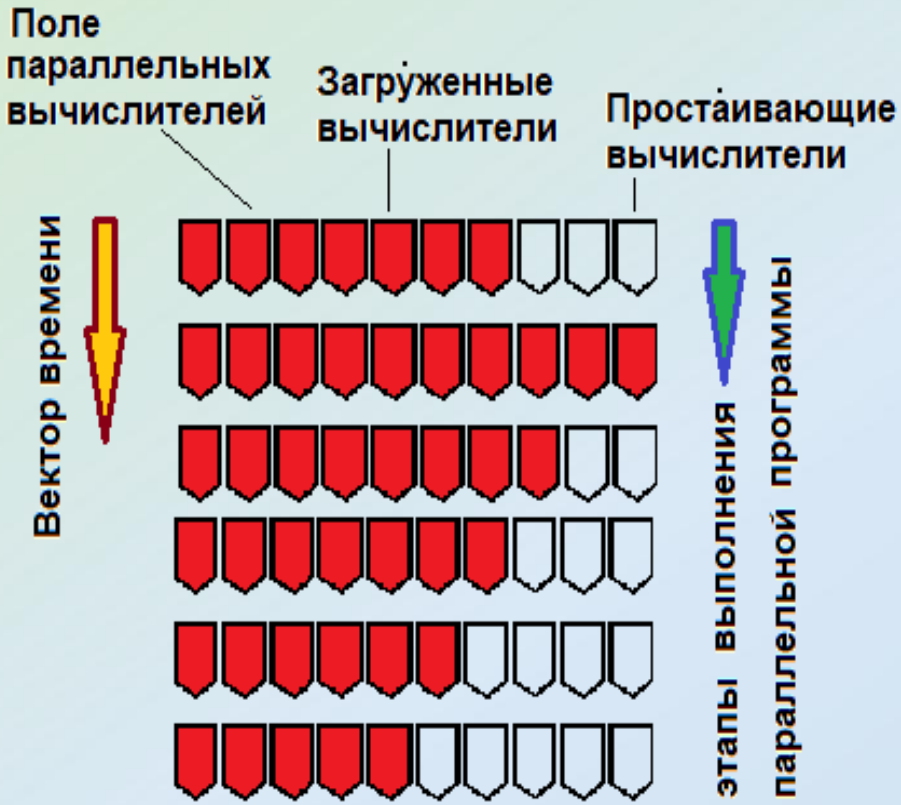
Допускаемые модификации:

💣 Невозможно уменьшение ширины ЯПР без увеличения высоты

💣 Возможно снижение ширины ЯПФ до единичной (последовательное выполнение)

Понятие ***ПЛОТНОСТИ КОДА*** при параллельных вычислениях

📖 Плотность кода – усреднённаая по длительности выполнения алгоритма до́ля за́нятости отдельных вычислителей из общего поля параллельных вычислителей в выполнении операций (некий аналог *коэффициента полезного действия*).

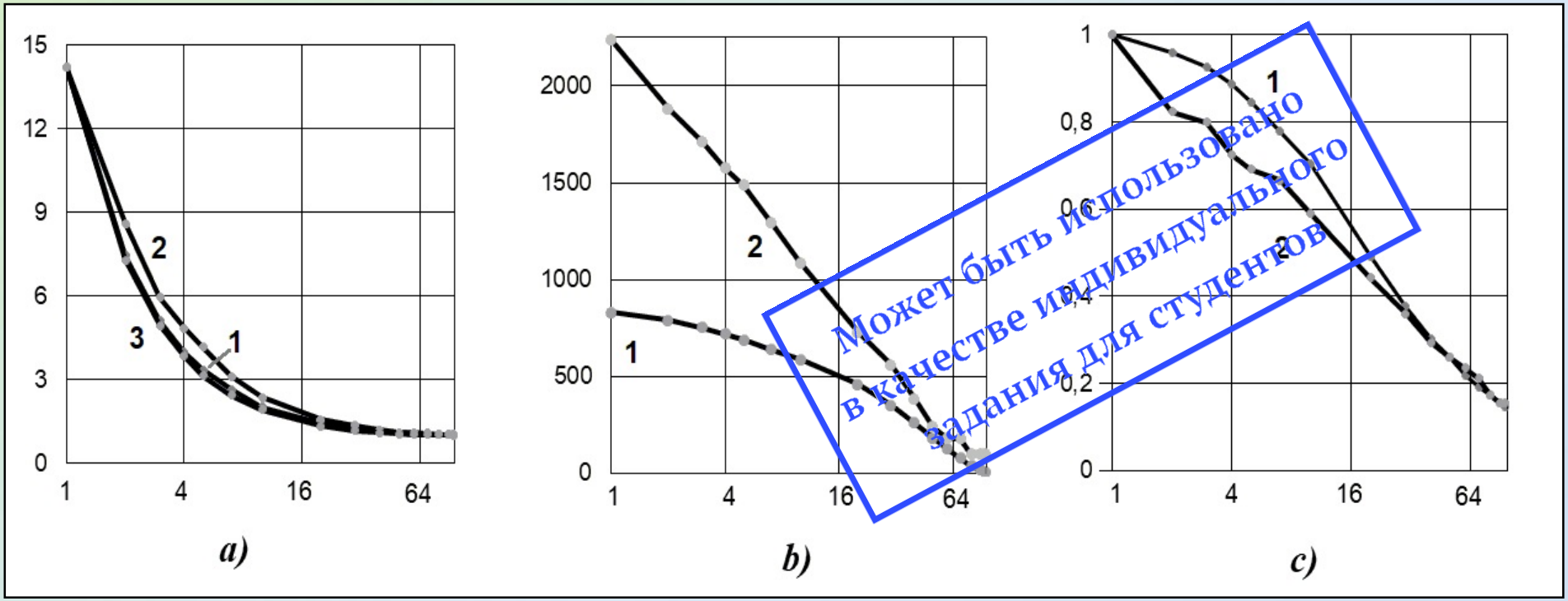


◀ Для показанного на рис. слева плана (*расписания*) выполнения параллельной программы плотность кода ≈ 0,72 (72%).

💣 Причинно-следственные связи в алгоритмах (связи типа “операторы ⇨ операнды”) препятствуют получению плотности кода в 100%. В общем случае получение плана (расписания) параллельного выполнения программы с максимальной плотностью кода является *NP*-полной задачей.

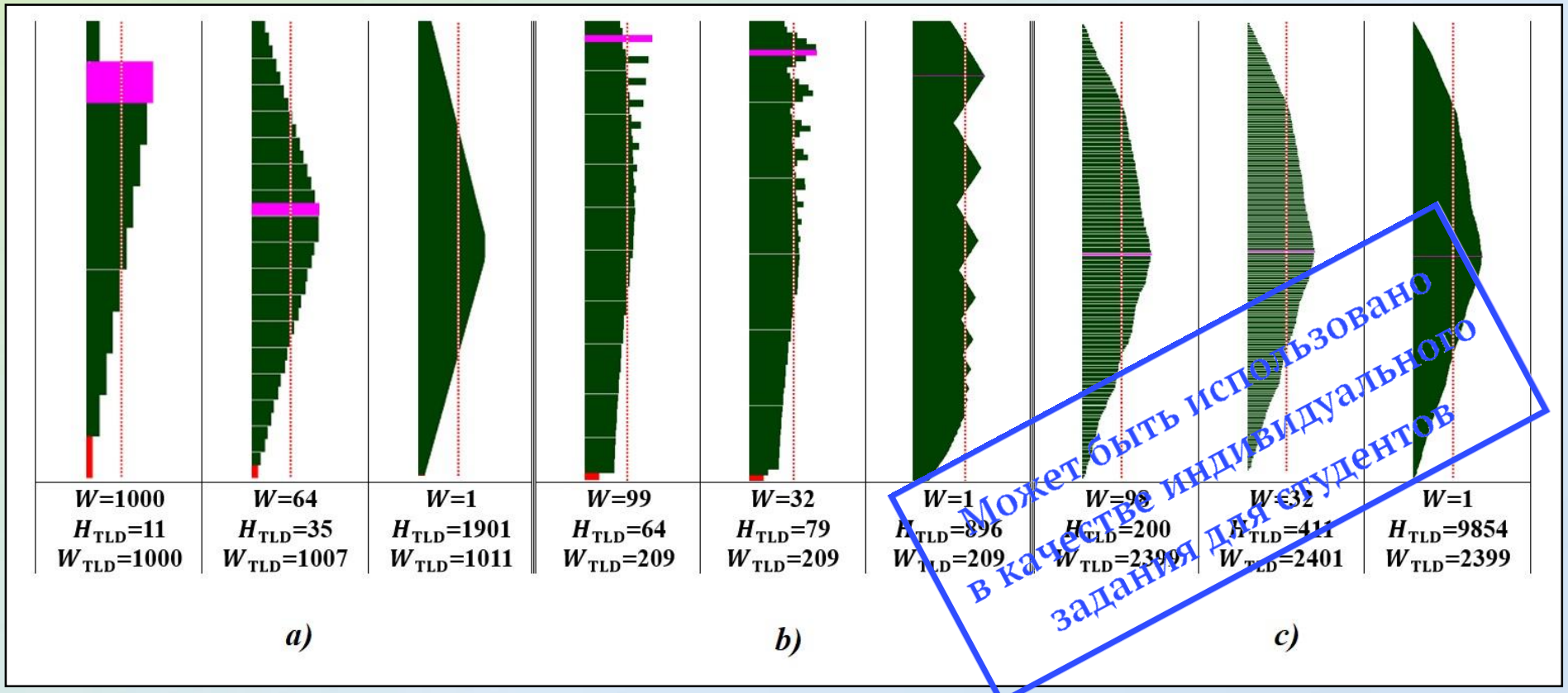
✍ В разрабатываемом методе используется *эвристический подход* к решению задачи *максимизации плотности кода.*

Пример исследования: сравнение эффективности сценариев построения планов выполнения программ на заданном числе параллельных вычислителей для VLIW-процессоров



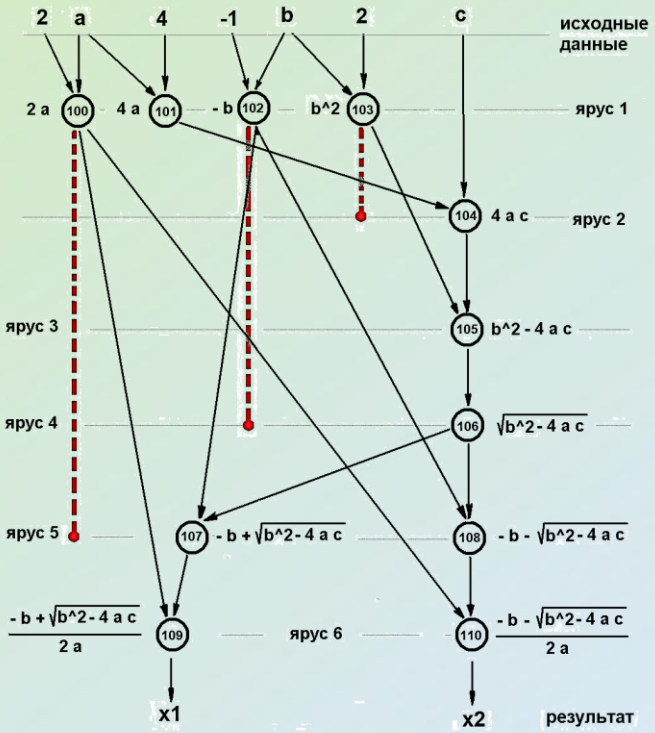
- Исследуемый алгоритм - решение систем линейных алгебраических уравнений (СЛАУ) 10-го порядка прямым (безитерационным) методом Гаусса; *ось абсцисс – ширина ЯПФ после преобразования (число слотов сверхдлинного машинного слова)*
- ***a)*** – параллельная вычислительная сложность (высота ЯПФ, *оценка времени выполнения*), ***b)*** – вычислительная сложность преобразования ЯПФ (в единицах перемещений операторов с яруса на ярус), ***c)*** – плотность кода (в единицах $k_и$)
- **1** – эвристика (*метод преобразования*) ***WithByWith***, **2** – эвристика ***Dithotomy***

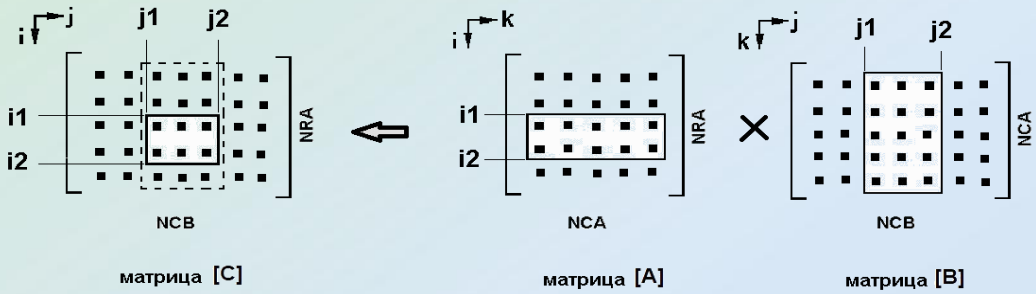
Пример исследования: сравнение форм диаграмм вре́менных локальных данных (***TLD***) и их распределения по высоте ЯПФ

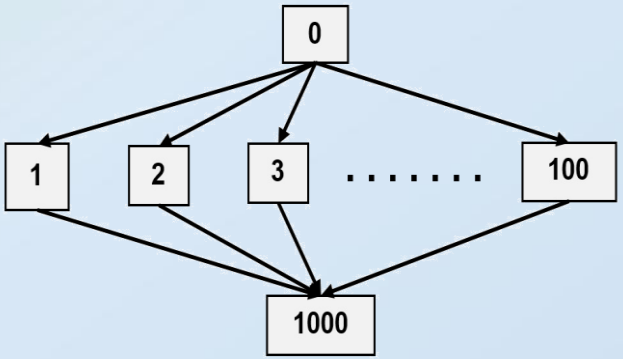


Исследуемые алгоритмы: ***a)*** - умножения квадратных матриц 10 порядка классическим методом, ***b)*** - решение систем линейных алгебраических уравнений (СЛАУ) 10-го порядка прямым (безитерационным) методом Гаусса, ***c)*** - искусственно сгенерированный ИГА алгоритма ***e17039_o9853_t199.gv***; W – ширина ЯПФ, H_{TLD} – высота TLD (=H+1), W_{TLD} – максимум единиц TLD

Микро- VS макропараллелизм применительно к обработке инструментом SPF@home







📖 **Микропараллелизм** характеризуется размером *гра́нул параллелизма* в одну машинную инструкцию (обычно соответствует одному *арифметическому действию*)

🛠 При выявлении микропараллелизма широко используется приём деления информационного графа алгоритма на *ярусы* (получение ЯПФ – *Ярусно-Параллельной Формы*) графа

📖 **Макропараллелизм** хотличается значительным размером (многие тысячи и более машинных инструкций) гра́нул параллелизма, часто используется в многомашинных системах (*вычислительных кла́стерах*)

🛠 Разбиение на гранулы (*декомпозиция*) обычно производится на основе априори известных данных о *топологии программы* (напр., информации об *блочном характере* обрабатываемых матриц)

Общая триеди́ная напра́вленность предлагаемого подхода к преподаванию предмета:

- Слушатели глубже понимают роль АЛГОРИТМА при разработке реальных программ и знакомятся со *стандартными алгоритмами* обработки данных.
- Осозна́ют ПОТЕНЦИАЛ ВНУТРЕННЕГО ПАРАЛЛЕЛИЗМА как неотъемлемую часть (*грань*) сущности АЛГОРИТМ (природу явления, его основные параметры, единицы измерения, ограничения, специфику применения).
- Естественным образом воспринимают принципиальную возможность целенаправленных эквивалентных преобразований алгоритмов с целью *наилучшего использования их в параллельных вычислительных практиках* (в смысле *приспосо́бленности к конкретным вычислительным системам*) и разрабатывают эффективные процедуры таких преобразований (*творческий компонент* работы).

Основные понятия (*дидакти́ческие единицы*) данного курса

- алгоритм
- формы представле́ния алгоритма
- информационная структура алгоритма
- граф алгоритма
- напра́вленный граф
- ацикли́ческий граф
- естественный (внутренний) параллелизм алгоритма
- неограниченный параллелизм
- гра́нула параллелизма, размер гра́нулы параллелизма
- граф-машина
- архитектура пото́кового (Data-Flow) вычислителя
- Ярусно-Параллельная Форма (ЯПФ) информационного графа алгоритма
- каноническая форма ЯПФ информационного графа алгоритма
- высота и ширина ЯПФ
- параллельная вычислительная сложность алгоритма
- ширина ЯПФ
- принцип гото́вности к вычислениям (ГКВ) для операнда
- принцип гото́вности к вычислениям (ГКВ) для оператора
- отло́женные вычисления
- допусти́мое положение оператора на ярусах ЯПФ графа алгоритма
- уровни параллелизма
- границы распараллеливания для данного уровня параллелизма
- алгоритмы, не допускающие распараллеливания
- целенаправленные преобразования графа алгоритма, эквивалентность целенаправленных преобразований графа алгоритма
- плотность кода
- вре́менные локальные данные (*Temporary Local Data* - TLD) алгоритма

Тематические WEB-ресурсы (E-Mail автора: e881e@mail.ru)

 https://vk.com/valery_bakanov

 https://dzen.ru/vbakanov.ru

 https://habr.com/ru/articles/792744/

 https://ok.ru/profile/586055777494

 https://my.mail.ru/mail/e881e

 https://www.facebook.com/ValeryBakanov

 https://github.com/Valery-Bakanov