РАЗРАБОТКА КАРКАСА (РАСПИСАНИЯ ВЫПОЛНЕНИЯ) ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ

Описание работы (конспект)

1.Краткая теория.

1.1.Вводная часть к началу занятия (краткое напоминание о предмете занятий и предыдущем материале). *Реализуется преподавателем или кем-либо из студентов; актуальна при длительном разрыве между занятиями и может быть опущена в противном случае.*

Требования науки и техники в вычислительной мощности компьютеров постоянно растут – появляются новые задачи, требуется повышение точности решения известных задач. Развитие математических методов всегда отстаёт от запросов прикладной науки.

Десятилетиями основным методом повышения производительности компьютеров был путь повышения тактовой частоты процессоров (аналог скорости работы двигателей, от которого зависит скорость движущего средства). Ограничением явилось повышенное тепловыделение процессоров, достигшее в 90-х г.г. величин около 100 W. В самом деле, согласно закону Релэя (1871 г.) в случае упругого рассе́яния (рассеяние называют упругим, если длина волны света не изменяется при рассеянии) и ма́лости размеров неоднородностей, на которых рассеиваются волны по сравнению с длиной волны λ (размер неоднородностей не более чем $0{,}1\lambda$), интенсивность рассе́янной энергии обратно пропорциональна четвёртой степени длины волны $1/\lambda^4 \approx f^4$ (f – частота колебаний). В процессорах электрические колебания рассеиваются на материале микросхемы (условие Релэя соблюдаются – при тактовой частоте 3 GHz длина волны 10 см, что намного больше размера атомов материала микросхемы). Т.о. при увеличении тактовой частоты вдвое тепловыделение возрастёт в 16 раз!

Такое тепловыделение в минимальном объёме корпуса процессора (много менее куб.см.) невозможно компенсировать радиатором, поэтому пришлось искать новые пути повышения вычислительной производительности. В 90-х г.г. ведущие фирмы-производители процессоров сначала начали применять технологию “гипертрединга”, а потом и многоядерности. При возрастании числа ядер вычислительная мощность (теоретически) и тепловыделение растёт всего лишь пропорционально их числу. Эффективное использование многоядерности до сих пор проблематично, ибо большинство программных продуктов создавалось в расчёте на последовательные вычисления. Следует заметить, что современный выбор пути многоядерности – в сильной степени проблема технологий производства процессоров. Если удастся снизить энергопотребление на несколько порядков, от многоядерности (и параллелизма в вычислениях можно) отказаться (скорее всего, временно). Современная 28 нМ технология производ-

ства процессоров характеризуется энергопотреблением около 20×10^{-12}Дж на каждую 64-битную *float*-операцию (значительно больше требуется на пересылку данных); имеются надежды на существенное снижении этой величины (что потребует внедрения новых материалов и технологий). Известен фундаментальный *принцип Ландауэра* (1961) – стоимость изменения состояния одного бита информации не менее 3×10^{-21} Дж.

2.Функциональная часть занятия. *Реализуется преподавателем. Именно здесь даются новые материалы, свойственные данному занятию.*

Итак, создать эффективную (наиболее быстродействию и максимально использующую ресурсы вычислительной системы0 параллельную программу много труднее, чем традиционно-последовательную. Единственно известный на сегодня путь достижения этой цели – тщательный анализ информационных связей в программе (как говорят специалисты, необходим “анализ тонкой информационной структуры”, [1]). Для многих распространённых алгоритмов такой анализ частично произведён – см. поддерживаемый НИВЦ МГУ им. М.В.Ломоносова проект AlgoWiki [2].

Первым этапом (см. ниже схему, представляющую собой фрагмент общего цикла разработки эффективной параллельной программы) является представле́ние алгоритма графом (выявление блоков обработки данных, характеризующихся размером (в машинных командах или блоках таких команд) и входными величинами (*операндами*), обычно стараются выбирать блоки примерно одинакового размера и небольшим (единицы) числом операндов. Это задача *декомпозиции* программы, являющаяся сама по себе достаточно сложной (напр., надо выбрать характерный размер таких блоков – от одной до сотен тысяч машинных команд, причём чем больше, тем лучше). Выявлению таких блоков очень помогает имеющееся представление о конкретном алгоритме (о его функционировании). Практика показывает, что тем большего объёма выбираются эти блоки, тем сложнее их выделить. В дальнейшем будем считать этап декомпозиции выполненным, а блоки называть *операторами* (памятуя при этом, что размер *оператора* может варьироваться от одной машинной команды до очень большого числа таких команд).

Следующий этап – теоретическое исследование свойств алгоритма по графовому описанию – хорошо представлен в упоминавшемся проекте Algowiki.

Сегодня мы будем заниматься дальнейшим этапом – созданием *каркаса параллельной программы* (фактически *расписания выполнения отдельных операторов*), на схеме справа этот этап выделен прямоугольником с двойной границей. Именно эта часть в значительной степени определяет эффективность разрабатываемой параллельной программы. На этой части анализа определяются операторы, мо́гущие выполняться параллельно

(т.е. независимые – ортогональные - по операндам) и, что самое главное, создаётся *расписание* (последовательность) их выполнения применительно к имеющейся параллельной вычислительной системе с учётом максимальной эффективности использования этой системы (а при возможности – минимума времени выполнения программы). Именно эта часть процесса разработки эффективных параллельных программ представляет максимум творчества для Исследователя.

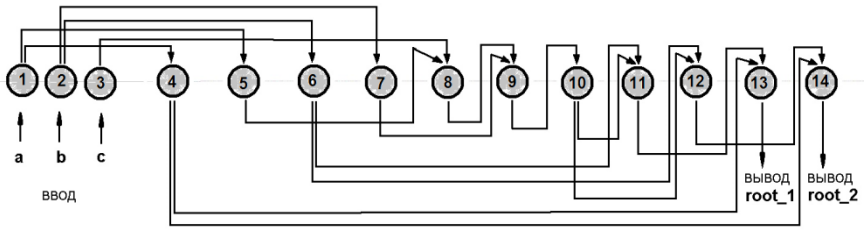
Следующий этап чисто технический – разработка исходного кода параллельной программы (с учётом выбора конкретной технологии параллельного программирования).

Самый простой вид графа алгоритма – информационный граф алгоритма (ИГА, вычислительная модель “операторы - операнды”). ИГА показывает только ИНФОРМАЦИОННЫЕ связи в алгоритме. В ИГА вершины соответствуют вычислениям, дуги – информационным связям (“что от чего зависит”). Простейший ИГА не содержит условных операторов и поэтому *детерминирован*. Т.к. в таком ИГА все дуги напра́влены, граф *ориенти́рован*. Представление алгоритма в виде ИГА может быть дополнено нагруже́нием дуг количеством передаваемых данных (а значит, временем их пересылки от оператора к оператору), а узлы - временем выполнения собственно операторов.

В качестве примера рассмотрим ИГА простой процедуры вычисления обоих действительных корней полного квадратного $ax^2+bx+c=0$. В нижерасположенной таблице приведены 11 арифметических операций (и три операции присваивания) для вычислений (один из вариантов) по известной (индийский астроном и математик Брахмагупта, около 598÷670 г.г. н.э.) формуле $x_{1,2} = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$.

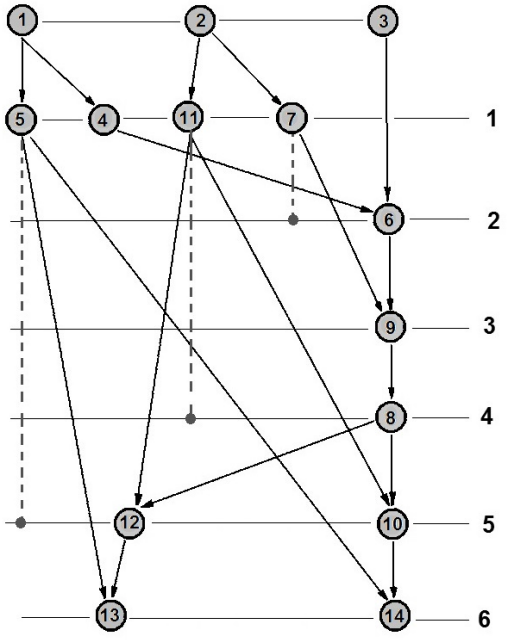
Действие	*Описание действия*	*Порядковый номер оператора*
a←значение	Присвоение значения переменной a	1
b←значение	Присвоение значения переменной b	2
c←значение	Присвоение значения переменной c	3
a2←2×a	a2 – рабочая переменная	4
a4←4×a	a4 – рабочая переменная	5
b_neg←neg(b)	b_neg – рабочая переменная (neg – операция изменение знака - ‘уна́рный минус’)	6
bb←b×b	bb – рабочая переменная	7
ac4←a4×c	ac4 – рабочая переменная	8
p_sqr←bb-a4	p_sqr – рабочая переменная	9
sq←sqrt(p_sqr)	sq – рабочая переменная (sqrt – операция вычисления квадратного корня)	10
w1←b_neg+sq	w1 – рабочая переменная	11
w2←b_neg-sq	w2 – рабочая переменная	12
root_1←w1/a2	root_1 – первый корень уравнения	13

root_1←w2/a2	root_2 – второй корень уравнения	13



Эта последовательность действий в виде ИГА произвольной формы показана на рис. слева. Однако при таком представлении ИГА нет возможности выявить параллелизм в программе (т.е. операторы, мóгущие быть выполненными одновременно = параллельно). Мощным средством для выявления параллелизма является представление ИГА в т.н. *ярусно-параллельной форме* (ЯПФ). При этом совершается более глубокий анализ информационных связей в ИГА и операторы распределяются по *ярусам* (обычно сверху вниз), причём на одном ярусе размещаются операторы, информационно (по данным) зависящие *только от операторов, находящихся на ярусах строго выше данного*. Такое представление ИГА уже является неким расписанием выполнения параллельной программы – находящиеся на одном ярусе операторы *информационно независимы* и поэтому могут быть выполнены одновременно (параллельно). Число ярусов называют *высотой*, а максимум узлов (операторов) на ярусе – *шириной* ЯПФ.

Процедура построения ЯПФ начинается с исходных данных (они помещаются на нулевой ярус), далее каждый оператор сравнивается с каждым и на каждый последующий ярус помещаются операторы, зависящие по данным (операндам) от операторов, расположенных на (уже определённых) ярусах *строго выше данного*; на последнем ярусе располагаются вычисления выходных величин. Можно строить ЯПФ начиная с выходных величин ("снизу"); логика построения такая же, хотя ЯПФ получится несколько иным. Вычислительная сложность построения ЯПФ полиномиальна (*квадратична*) от числа операторов.



На рисунке справа показан ЯПФ (каноническая форма) рассматриваемой операции (в кружках – номера операторов, цифры справа – номера ярусов).

Видно, что максимально нагружен 1-й ярус (4 параллельно выполняющихся оператора, минимально – 2, 3 и 4-й яруса. Такая неравномерность нагрузки ярусов типична для большинства алгоритмов и приводит к большой неравномерности нагрузки на отдельные вычислители параллельной системы (неэффективное использование оборудования).

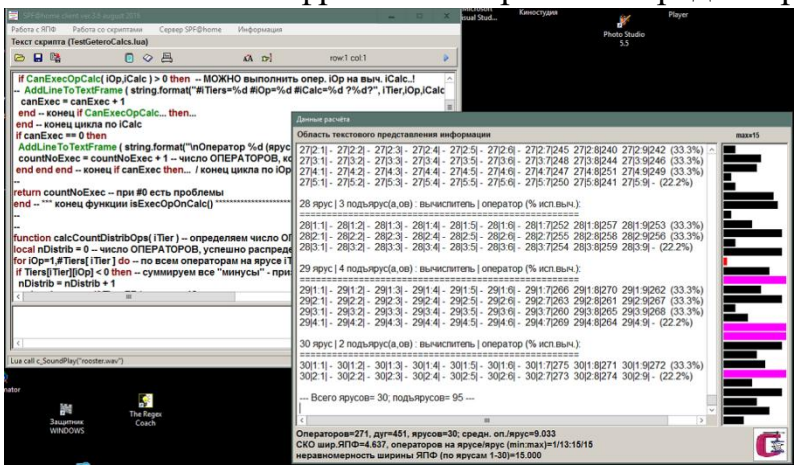
Однако более тщательный анализ ЯПФ даёт возможность устранить эти нежелательные явления. На предыдущем рис. пунктиром показаны диапазоны возможного размещения операторов без нарушения исходного ИГА. Напр., перемещая оператор 7 на ярус 2, а оператор 5 – на ярус 3, получаем ЯПФ шириной 2 (вместо исходных 4); при этом возможно исполнение данной программы на 2-х параллельно работающих вычислителях (вместо 4-х при канонической ЯПФ) при том же времени выполнения (числе ярусов). Заметим, что представленный вариант переноса операторов с яруса на ярус не единственный, при большом числе операторов (и ярусов) это число может быть запредельным.

Однако обсуждаемый потенциал вариабельности в смысле переноса операторов по ярусам не безграничен – ширина ЯПФ не может быть менее некоторого значения (при неизменном числе ярусов). При необходимости выполнения такого алгоритма на ограниченном числе вычислителей можно смириться с увеличением времени выполнения (пойти на увеличение числа ярусов); это достигается введением дополнительных пустых ярусов ЯПФ и переносом на них части операторов с наиболее нагруженных ярусов. При этом получаем комбинационную задачу (при достаточно большом числе операторов и ярусов это *NP*-полная задача). Решение усложняется при учёте запаздываний при передаче данных, времени собственно вычислений и гетерогенности (неоднородности) поля параллельных вычислителей.

Точный (приводящий к абсолютному оптимуму по некоторой функции) метод решения таких задач неизвестен (кроме полного перебора вариантов, что технически невозможно вследствие огромного числа комбинаций), поэтому представляет интерес разработка определённых стратегий в перестановке операторов, позволяющих получить хоть какие-то приемлемые решения задачи (возможна и полезна конкуренция стратегий по сложности и времени их выполнения). Логично разделить такие стратегии на класс (условно) простых и сложных (уровня методов генетических, “ветвей и границ” и т.п.). При этом бессмысленно говорить о нахождении абсолютного оптимума - возможно лишь постепенное к нему приближение (в этом смысле имеет смысл говорить не об *оптимальном*, а о *рациональных* решениях, приближающихся к оптимальному).

Программный комплекс SPF@home (фактически *исследовательский программный стенд*) был создан специально для реализации таких стратегий и сравнения их эффективности [3]. Система разработана на языке C/C++ и представляет собой Window'32 приложение с графическим интерфейсом. Для реализации стратегий генерации представле́ний ИГА использован встроенный скриптовый язык Lua, служащий надстройкой над системой функций API (*Application Program Interface*) собственно преобразований ИГА. Язык Lua выбран вследствие компактности, значительной функциональности и синтаксической близости к C (что упрощает процесс освоения). Симбиоз интерпретируемого языка Lua и компилируемого C очень эффективен – первый даёт системе большую гибкость, второй – скорость обработки данных.

Пользовательский интерфейс системы SPF@home выполнен в Windows-стиле MDI (*Multy Display Interface*, МногоОконный Интерфейс) и представлен главным окном с фреймом встроенного редактора Lua-скриптов и меню управления процессом обработки данных и (немодальным) дочерним окном с фреймами выдачи текстового расчётного материала и ленточного графика распределения числа операторов по ярусам ЯПФ (на рис. слева представлена копия экрана персонального компьютера под управлением Windows 10 при работе данного приложения и частичном перекрытии главного окна дочерним в режиме моделирования гетерогенного поля вычислителей).



Программная система SPF@home свободно распространяется в форме исполняемого модуля под Windows и может быть выгружена как (архивированный) ресурс http://vbakanov.ru/spf@home/content/spf@home.rar (дополнительная информация см. http://vbakanov.ru/spf@home). Описание системы (включая API-функции и др. см. во входящем в комплект распространения файле API_User.pdf).

3.Практическое использование системы SPF@home.

Вся работа ведётся во встроенном редакторе языка Lua. Определены *информационные* (служат для получении информации об ИГА и его представлениях), *акционные* (изменяют представления ИГА) и *вспомогательные* (обслуживание файловой системы и др.) функции.

Примеры простого информационного API-вызова – получить общее число операторов (GetCountOps) или дуг (GetCountEdges) ИГА (ИГА представлен в виде двумерного списка смежных вершин (фактически дуг графа) файлами с расширение EDG). На практике ИГА может быть составлен самостоятельно по описанию алгоритма на псевдоязыке, получен из библиотеки графов программ или сгенерирован специальной программой-графогенератором. Автор разработки пользуется им же разработанным программным симулятором пото́кового (DATA-FLOW) вычислителя, в котором программы отлаживаются и далее экспортируется в файловый EDG-формат списка смежных вершин ИГА (см. ресурс http://vbakanov.ru/dataflow/content/dataflow.rar и его описание http://vbakanov.ru/dataflow/).

Мощным из раздела акционных является API-вызов CreateTiersByEdges – получение из ИГА его ЯПФ. Естественно, существуют функции создания пустого яруса под заданным (AddTier), перемещения на него части операторов с

иных ярусов (MoveOpTierToTier), уничтожения пустого яруса (DelTier) и т.д. Важными являются API-вызовы, определяющие диапазон возможного нахождения данного оператора на ярусах ЯПФ (GetMinTierMaybeOp и GetMaxTierMaybeOp).

Вызовы работы с файловой системой служат для считывания исходного ИГА (ReadEdges), параметров вычислителей и операторов (режим гетерогенного поля вычислителей), сохранения рассчитанных данных в разных форматах и др. API-функции тестового (AddLineToTextFrame, PutTiersToTextFrame и др.) и графического (DrawDiagrTiers) вывода данных позволяют реализовать диалог с Исследователем (включая DelayMS и SoundPlay приостанова вычислений и проигрывания аудиофайла с целью звукового информирования о происшедших в программе событиях соответственно).

Простейшая Lua-программа может выглядеть так (синтаксис Lua предусматривает двойной дефис как указатель комментария до конца строки – аналогично двойному слэшу в C++):

```
-- начало программы
--
projectName = "e17_o11_t6" – имя проекта
ReadEdges(  projectName .. ".edg" )  -- читаем файл описания графа
PutEdgesToTextFrame() -- вывод файла дуг ИГА в текстовый фрейм дочернего окна
CreateTiersByEdges() -- создать ЯПФ по ИГА
PutTiersToTextFrame() – выдать ЯПФ в текстовый фрейм дочернего окна
ClearDiagrTiers()  -- очистить окно ленточной диаграммы
DrawDiagrTiers() – отрисовать ленточную диаграмму распределения ширин ярусов ЯПФ
AddLineToTextFrame( string.format("\n--- Всего ярусов= %d ---", GetCountTiers() )
--
-- копируем ЯПФ во внутреннее представление Lua ----------------------------------
Tiers = {} -- создаём список (будущая таблица номеров операторов по ярусам)
for i=1,GetCountTiers() do -- число ярусов ЯПФ
  Tiers[i]={} -- создаём строку массива для 2D-таблица
  for j=1,GetCountOpsOnTier(i) do -- число операторов на ярусе
    Tiers[i][j]=GetOpByNumbOnTier( j, i ) -- взять и запомнить номер оператора
    AddLineToTextFrame ( string.format("i=%d j=%d Op=%d", i, j, Tiers[i][j]))  -- данные об ярусе
  end – конец по j (по порядковому номеру оператора на ярусе i)
end -- конец по i (по номеру яруса ЯПФ)
--
SoundPlay("rooster.wav") – крик петуха возвещает, что работа сделана
--
-- конец программы
```

В этой программе считывается файл ИГА с именем e17_o11_t6.edg, данные ИГА выдаются в текстовый фрейм дочернего окна, создаётся и визуализируется в текстовом и графическом виде ЯПФ, выдаётся число ярусов ЯПФ, далее ЯПФ копируется в двумерную таблицу Tiers[][], данные каждого яруса распечатываются, окончание программы сигнализирует аудиосигналом. Для дальнейшего анализа содержимое текстового фрейма дочернего окна может быть скопировано в ClipBoard, передано в NotePad или MS Word (соответствующее

меню доступно по правой кнопке мыши), ленточная диаграмма распределения ширин ярусов ЯПФ копируется в ClipBoard по щелчку левой кнопкой мыши в области изображения.

Во встроенном редакторе зарезервированные слова языка Lua отображаются красным, имена API-функций – зелёным цветом, текущее положение курсора показывается на верхней панели. Дополнительно представляется PPTX-презентация по системе SPF@home, предназначенная в основном для направлений программистов (описываются некоторые подробности системы в части представления данных, разработки API-системы и т.п.).

Для более полного изучения возможностей Lua рекомендуется книга [4] и многочисленные ресурсы сети InterNet.

4.Использование системы SPF@home в самостоятельной работе студентов (домашняя или исследовательская работа).

В целом применение программного стенда SPF@home имеет огромный потенциал именно для самостоятельной работы студентов, ибо предлагает возможность *творческой работы* (фактически Исследования). Творческая компонента подкрепляется *потенциалом состязательности* (близкая к игровой компонента), заключающимся в стремлении получить лучший (по заданному критерию) результат, чем другие обучающиеся.

Некоторая проблема заключается в необходимости предварительного "вхождения" в систему программирования Lua и получения навыков использования системы API-функций пакета SPF@home, однако для студентов связанных с компьютерными технологиями направлений обучения это особых сложностей не представляет.

Предлагаются варианты самостоятельных заданий (во всех случаях исходный ИГА различной сложности выдаётся преподавателем):

а). Разработать (и реализовать на языке Lua) стратегию приведения ЯПФ к максимально сбалансированному (имеется в виду наибольшая равномерность в распределении оператор по ярусам ЯПФ) виду при условии сохранения высоты ЯПФ.

Неравномерность распределения операторов по ярусам можно характеризовать коэффициентом неравномерности $k = max(W_i)/min(W_i)$ и/или средним квадратичным отклонением $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{i=N}(W_i - \overline{W})^2}$ этой же величины (подсчитываются и выдаются системой); здесь N – число ярусов ЯПФ, W_i и $\overline{W}$ – ширина i-того яруса и среднеарифметическая ширин ярусов соответственно). Эффективность каждой из предложенных стратегий логично оценивать числом перестановок операторов с яруса на ярус (аналогично задачам сортировки), для этого предусмотрены API-функции CountMovesZeroing и GetOpsMoves.

При такой постановке группе учащихся выдаётся один ИГА, а получившим наилучший результат считается достигший минимума k и/или σ при минимуме перестановок (возможно, придётся предложить - при общественном согласии - некий синтетический критерий, учитывающий оба эти параметры).

б). Разработать (и, конечно, реализовать на языке Lua) стратегию приведения ЯПФ к ширине не менее заданной W_0 при возможном возрастании высоты ЯПФ; фактически это составление расписания выполнения данного алгоритма на многопроцессорной вычислительной системе архитектуры сверхдлинного командного слова (VLIW, *Very Long Instruction Word*) - аппаратно-программная идеологема микропроцессорной архитектуры с явным параллелизмом команд (EPIC, *Explicitly Parallel Instruction Computing*) с числом процессоров W_0. При такой постановке требование наибольшей сбалансированности ширин ярусов ЯПФ возникает автоматически. Побеждает тот, которому удастся разработать стратегию, позволяющую добиться результата с минимумом высоты ЯПФ при минимуме перестановок операторов с яруса на ярус (значения k и σ также полезно учитывать коллективно выбранным способом).

Оформление отчёта по Исследованию в общем свободное, но должны быть указаны характеристики исходного ИГА и алгоритма, которому он соответствует (если таковой существует), текстовое и на языке Lua описание стратегии преобразования ЯПФ, информация об интересных (с точки зрения исследователя) моментах и находках при разработке стратегии, информация об исходной и преобразованной ЯПФ (возможно, по этапам преобразования). Значительные по объёму файлы следует представить на электронном носителе.

В качестве отправной точки при поиске лучших стратегий можно воспользоваться результатами автора 2015 г. (не излишне сложные, однако дающие определённый эффект стратегии), приведённые на ресурсе http://vbakanov.ru/spf@home/tables_effect.pdf и в работе [3].

Опыт показывает, что применение Информационно-Коммуникационных Технологий (ИКТ) доказало свою эффективность - обучаемые поняли важность составления рационального расписания выполнения параллельных программ, освоили подходы к решению этой проблемы и получили начальные практические навыки по реализации её решения; один из обучаемых выбрал данную тему для курсовой квалификационной работы и успешно защитил её.

В настоящее время готовятся описания и остальных занятий по заявленному курсу, которые автор надеется также представить на обсуждение научно-педагогической общественности.

Список использованных источников

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. — 608 с.

2. AlgoWiki. Открытая энциклопедия свойств алгоритмов. [Электронный ресурс]. Дата обновления: 03.09.2015. URL: http://algowiki-project.org/ru/ (дата обращения: 10.09.2016).
3. Баканов В.М. Программный инструментарий анализа информационной структуры алгоритмов по их информационным графам. // Параллельные вычислительные технологии (ПаВТ'2016): труды международной научной конференции (г.Архангельск, 28.03–01.04.2016). —Челябинск: Издательский центр ЮУрГУ, 2016, с. 432-441.
4. Иерузалимски Роберту. Программирование на языке Lua. — М.: ДМК Пресс, 2014. — 382 с.