М. А. Плаксин,
Национальный исследовательский университет «Высшая школа экономики», Пермский филиал

ПРОПЕДЕВТИКА ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ В ШКОЛЬНОЙ ИНФОРМАТИКЕ. ПАРАЛЛЕЛЬНАЯ ФОРМА АЛГОРИТМА

Аннотация

Описываются понятие «параллельная (ярусно-параллельная) форма алгоритма», порядок ее построения и анализа для управления процессом распараллеливания алгоритма. В качестве примера дается разбор задания из конкурса «ТРИЗформашка-2014».

Ключевые слова: информатика, средняя школа, начальная школа, методика обучения, параллельные алгоритмы, параллельная форма алгоритма, ярусно-параллельная форма, параллельные вычисления, пропедевтика, ТРИЗформатика, ТРИЗформашка.

Контактная информация

Плаксин Михаил Александрович, канд. физ.-мат. наук, доцент кафедры информационных технологий в бизнесе Пермского филиала Национального исследовательского университета «Высшая школа экономики»; *адрес:* 614070, г. Пермь, ул. Студенческая, д. 38; *телефон:* (342) 205-52-50; *e-mail:* mapl@list.ru

M. A. Plaksin,
National Research University Higher School of Economics, Perm Branch

PROPAEDEUTICS OF PARALLEL COMPUTING IN SCHOOL INFORMATICS. PARALLEL FORM OF THE ALGORITHM

Abstract

There are the description of the conception "parallel form of the algorithm (multilevel structure)", the procedure of its design and analysis for management of process of parallelization of algorithm. As an example, the investigation of task from the contest "TRIZformashka-2014" is given.

Keywords: informatics, secondary school, primary school, teaching methods, parallel algorithms, parallel form of algorithm, multilevel structure, parallel computing, propaedeutics, TRIZformatics, TRIZformashka.

Современный этап развития computer science связан с массовым распространением параллелизма вычислений на всех уровнях (многоядерные процессоры, многомашинные кластеры, многопроцессорные ЭВМ). Это делает актуальным включение пропедевтики параллельного программирования в школьный курс информатики.

В рамках работ над «пермской версией» пропедевтического курса информатики (авторский коллектив: М. А. Плаксин, Н. Г. Иванова, О. Л. Русакова; рабочее название курса «ТРИЗформатика») [6, 7] разработка данной тематики начата в 2013 году. Эффективной площадкой для этого выступил конкурс «ТРИЗформашка» — ежегодный межрегиональный интернет-конкурс по информатике, системному анализу и ТРИЗ (теории решения изобретательских задач) для школьников и студентов [3–5, 8]. В марте 2017 года конкурс состоится в 17-й раз. Возраст участников — от первого класса до четвертого курса. Среднее количество команд — около 100 (рекордное — 202). География конкурса — от Владивостока до Риги. Сайт конкурса: http://www.trizformashka.ru

В процессе подготовки конкурса «ТРИЗформашка» были определены типы задач, направленных на освоение базовых понятий параллельных вычислений, и придуманы по несколько задач этих типов, в том числе задачи на параллельную (ярусно-параллельную) форму алгоритма. Разбору понятия «параллельная форма алгоритма» и задач данного типа посвящена данная статья.

Далее приводится текст задачи конкурса «ТРИЗформашка-2014», теоретический материал (описание понятия «параллельная форма алгоритма») и разбор задачи.

Задача. Компания «Джин-строй»

(Краткое содержание предыдущих задач. В задачах предыдущих «ТРИЗформашек» было рассказано о том, как любитель экстремального отдыха сэр Джон отправился на каникулы из Оксфорда в гости к своей тетушке Марфе в подмосковную деревню Ромашкино. В соседней деревне Васильково он встретил красавицу Глашу. Сэр Джон и красавица Глаша подружились. Они вместе путешествовали по миру, встречали в России Новый год, разыскивали в родовом замке сэра Джона карту его прадедушки пирата, на которой указано, где зарыты пиратские клады, и т. д. В конкурсе «ТРИЗформашка-2011» они поженились, в «ТРИЗформашке-2013» у них родился сын Александр.)

Ответы на вопросы данного задания обязательно должны быть обоснованы.

Красавица Глаша и сэр Джон отправились в гости к тетушке сэра Джона Марфе в подмосковную деревню Ромашкино. Сын Джона и Глаши Александр уже учился ходить. И давно пора было показать его ромашкинским родственникам.

Ходил Александр еще не очень уверенно. Зато хорошо умел залезать во все и всяческие щели, лазы и ходы. Время от времени откуда-нибудь из угла раздавался возмущенный крик ребенка, застрявшего при исследовании очередного заинтересовавшего его таинственного уголка. Родители спешили на помощь, выручали чадо из трудной ситуации и отпускали для дальнейших исследований.

Вынимая сына из одной из щелей за кованым сундуком, которому явно была не одна сотня лет, Глаша обнаружила, что дитятко прихватило с собой, зажав в кулачок, обрывок какого-то листка. Развернув его, Глаша обнаружила следующий текст, написанный старинным шрифтом и испещренный буквами, которые в русском языке не употреблялись уже очень давно.

— О, отец! — воскликнул старший сын. — Доверьте сооружение дворца мне. Я призову великое множество джинов. И они построят дворец за неделю!

— О, отец! — отозвался средний сын. — Зачем нам такое количество джинов? Ведь дворец нужен нам ко дню приезда принцессы. А это произойдет только через две недели. Доверьте стройку мне. Я призову ровно столько джинов, чтобы дворец был готов к тому дню, когда принцесса въедет в наш город!

— О, отец! — промолвил младший сын. — Джины могучи. Каждый из них способен за день вырубить из скалы и уложить одну каменную глыбу, из которых и будет построен дворец. Но за содержание каждого джина приходится платить две золотых монеты в день. Это дорогое удовольствие. А доберется ли принцесса за две недели, еще неизвестно. Я думаю вот о чем. Как только джины достроят дворец, на следующий день к нему хлынут толпы приезжих, чтобы полюбоваться чудесным сооружением. С каждого из них мы будем брать одну очень мелкую медную монету. Но приезжих будет столько, что за день мы наберем медяков на полноценный золотой. Поэтому, если вы поручите стройку мне, я призову такое количество джинов, чтобы строительство дворца обошлось нам как можно дешевле.

На другой стороне листа находился сделанный от руки набросок чертежа какого-то сооружения, сложенного из громоздких каменных глыб неправильной формы.

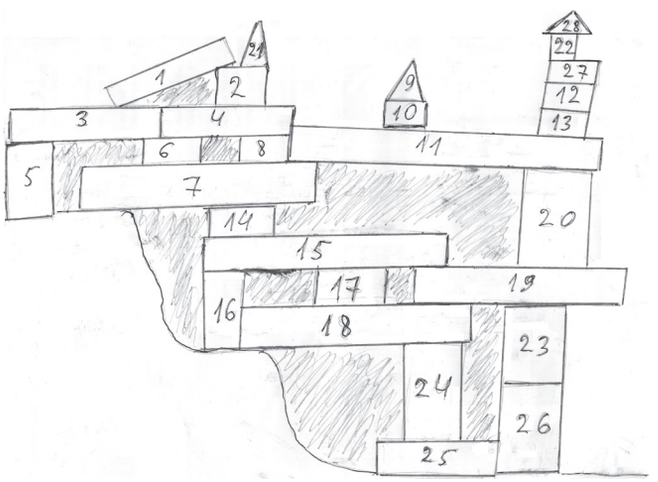

Рис. 1. «Чертеж дворца» для задачи из конкурса «ТРИЗформашка-2014»

— Наверно, это и есть дворец, строительство которого обсуждали братья, — подумала Глаша. — И на то, чтобы вырубить из скалы такую глыбу и уложить ее на стену, даже джину требовался целый день. Зато работали джины, наверняка, без выходных. Они же джины. Интересно, сколько джинов должен был призвать каждый из братьев для того, чтобы выполнить свое обещание?

(Продолжение истории сэра Джона и красавицы Глаши см. в следующем конкурсе.)

Перед продолжением чтения читателю рекомендуется самостоятельно решить предложенную задачу.

Понятие параллельной (ярусно-параллельной) формы алгоритма

Предварительные замечания:

1. Фрагменты программ, приведенные в данном разделе, написаны на языке Паскаль. Выбор языка обоснован единственным фактором: Паскаль — самый распространенный язык в российских школах. Никаких особых свойств Паскаля в примерах не используется, и эти примеры можно совершенно безболезненно переписать на любой другой императивный язык.
2. С аппаратной точки зрения параллельное выполнение программы может быть обеспечено разными путями:
   - за счет совместной работы нескольких однопроцессорных машин;
   - за счет многопроцессорности одной машины;
   - за счет многоядерности одного процессора.

   Эта сторона дела нас сейчас не интересует. Поэтому при обсуждении параллельных вычислительных систем мы будем пользоваться понятием «вычислительное ядро», отвлекаясь от его аппаратной реализации.
3. Автор осознает различие понятий «алгоритм» и «программа» и достаточно сложные отношения между ними (программа реализует алгоритм, представляя его средствами конкретного языка программирования с учетом особенностей этого языка; «любая программа для “обыкновенного” компьютера описывает некоторое семейство алгоритмов. Выбор конкретного алгоритма при ее реализации определяется тем, как срабатывают условные операторы» [2, с. 191]). Но обсуждение таких тонкостей не входит в задачу данной статьи. Поэтому между этими понятиями не делается строгого различия.

Переход от последовательного программирования к параллельному возбудил интерес к новому свойству алгоритмов, которое ранее никого не интересовало. Во времена последовательного программирования главными характеристиками алгоритма являлись правильность, сложность и — для численных методов — погрешность вычислений. Наиболее многоаспектным свойством является сложность. Она делится на структурную (логическую), емкостную (требования к памяти), временну́ю (затраты времени на выполнение алгоритма). Даже длина алгоритма, выраженная в количестве команд, в некоторых случаях используется как оценка сложности алгоритма.

Переход к параллельному программированию вывел на передний план четвертую характеристику алгоритма — способность к распараллеливанию, способность к представлению алгоритма в параллельной форме [1, 2].

Пусть у нас есть программа, которую мы раньше выполняли на последовательной ЭВМ. Теперь мы запускаем эту же программу на машине с двумя вычислительными ядрами. Означает ли это, что программа будет выполнена вдвое быстрее? Увы, не всегда!

Рассмотрим два примера. В них приведены только фрагменты раздела операторов, но опущенные описания легко понятны из контекста.

Пример 1.

```
read(a);
a:=a+1;
a:=a*a;
write(a);
```

Пример 2.

(Номера строк не имеют отношения к программе и добавлены только для удобства обсуждения.)

```
 (1) read(fr1,a);
 (2) read(fr2,b);
 (3) read(fr3,c);
 (4) read(fr4,d);
 (5) a:=a+1;
 (6) b:=b+2;
 (7) c:=c+3;
 (8) d:=d+4;
 (9) d:=7-d;
(10) c:=-c*c;
(11) b:=b*b*b;
(12) a:=a*a;
(13) write(fw1,a);
(14) write(fw2,b);
(15) write(fw3,c);
(16) write(fw4,d);
```

Пусть мы хотим выполнить эти фрагменты на разных вычислительных системах — с одним, двумя, четырьмя и восьмью ядрами. Как скажется количество ядер на времени исполнения фрагментов?

Очевидно, что для первой программы количество ядер никакой роли играть не будет. Все команды данной программы должны выполняться строго одна за другой. Никакой возможности для распараллеливания здесь нет. Поэтому, сколько бы ядер ни имела наша вычислительная система, работать будет только одно из них. Остальные обречены на простой. В данном случае мы зря потратили деньги на покупку сложной вычислительной техники.

Другое дело — вторая программа. Она легко делится на четыре независимые линии (рис. 2).

Каждую такую линию можно запустить на выполнение на отдельном ядре. Поскольку линии друг с другом не связаны, мы можем рассчитывать на то, что на двухъядерной системе программа выполнится быстрее примерно вдвое, на четырехъядерной — быстрее примерно вчетверо. И на этом способность нашей программы к ускорению при увеличении числа вычислительных ядер исчерпывается. Переход к восьмиядерной системе не даст нам никакого выигрыша по сравнению с четырехъядерной.

Алгоритм второй программы мы представили ***в параллельной форме***.

Рассмотрим этот процесс более детально. Что мы должны сделать, чтобы из последовательности команд примера 2 получить четыре независимых линии, пригодных для параллельного выполнения?

Будем просматривать команды друг за другом.

Первую команду поставим в начало первой линии. (Заметим, что первая линия есть всегда. Просто в последовательном алгоритме она останется единственной.)

Вторая команда может выполняться независимо от того, выполнялась первая или нет. Для своего выполнения она не требует никаких данных, никакой информации, которые стали бы известны только после выполнения первой команды. Между этими командами нет никаких информационных связей. Поэтому для второй команды мы создадим вторую линию, которая с этой команды будет начинаться.

Аналогично поступим с третьей командой (она информационно не связана ни с первой, ни со второй) и с четвертой (она информационно не связана ни с одной из предыдущих команд).

Пятая команда зависит от первой: она ссылается на значение переменной *a*, которая была введена в первой команде. Между этими командами существует информационная связь. Поэтому пятую команду мы поместим на первую линию вслед за первой командой.

Аналогично команды 6, 7 и 8 будут размещены на линиях 2, 3 и 4 (команда 6 зависит от команды 2, команда 7 — от команды 3, команда 8 — от команды 4).

Команда 9 информационно связана с командой 8. Она использует переменную *d*, значение которой определено в восьмой команде. Значит, место команды 9 — на линии 4.

Команда 10 использует переменную *c*, вычисленную командой 7. Поэтому она попадает на линию 3.

Команды 11 и 12 зависят от команд 6 и 5 (по переменным *b* и *a* соответственно). Это определяет их место на линиях 2 и 1.

Аналогично распределим по линиям последние четыре команды.

Параллельная форма алгоритма для примера 2 построена.

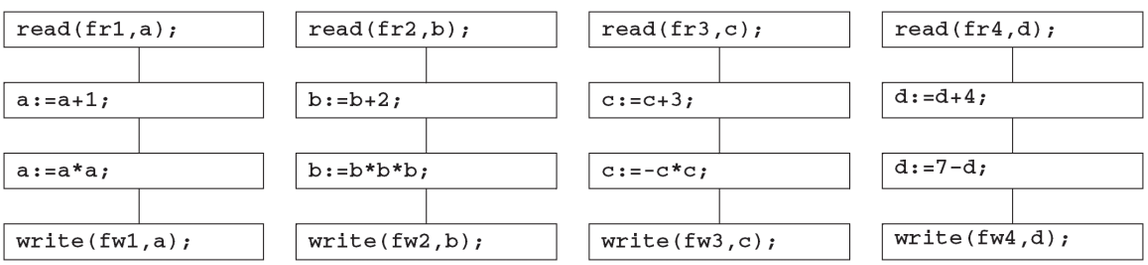

Рис. 2. Параллельная форма программы из примера 2

Заметим, что если мы применим эту же процедуру к примеру 1, то все команды выстроятся строго в одну линию.

В примере 2 все четыре линии оказались совершенно независимы. В реальной программе все будет сложнее. Рассмотрим следующий пример.

Пример 3.

(Номера строк не имеют отношения к программе и добавлены только для удобства обсуждения.)

```
(1) read(a,b,c,d);
(2) a:=a+1;
(3) b:=b+2;
(4) c:=c+3;
(5) d:=c+d+4;
(6) c:=-c*c;
(7) b:=b*b*b;
(8) a:=a*a;
(9) a:=b-a;
(10) write(a,b,c,d);
```

В этом примере информационные связи между командами существенно сложнее. Нам уже не удастся представить параллельную форму в виде четырех независимых линий. Для отображения информационных зависимостей придется построить полноценный граф.

Строить его будем по тем же правилам, что и в предыдущем случае.

Начнем с команды 1, которая не зависит ни от одной другой команды.

Команды 2, 3 и 4 зависят от команды 1. Расположим их ниже уровнем и проведем дуги от команды 1 к командам 2, 3, 4.

Команда 5 будет зависеть уже от двух предыдущих команд: 1 и 3. Расположим ее еще ниже уровнем и свяжем с командами 1 и 3. На этом же уровне поместим команду 6 (зависит от команды 4), команду 7 (зависит от команды 3) и команду 8 (зависит от команды 2).

Команда 9 (зависящая от команд 7 и 8) в одиночку попадет на четвертый уровень.

Наконец, на пятом уровне будет расположена завершающая программу команда 10, связанная с командами 9, 7, 6 и 5.

Параллельная форма примера 3 построена (рис. 3).

В графах программ 2 и 3 вершины располагаются по уровням — ***ярусам***. На схеме графа программы 3 номера ярусов обозначены слева.

На первом ярусе располагаются команды, не требующие для своего выполнения каких-либо предварительных вычислений. Каждая команда, кроме команд первого яруса, требует для своего исполнения некоторых данных, подготовленных другими командами. Все такие «подготовительные» команды будут располагаться на предыдущих ярусах.

Такое представление алгоритма называют ***ярусно-параллельной формой (ЯПФ)***. Количество вершин, расположенных на одном ярусе, называют ***шириной яруса***. Максимальную ширину ярусов называют ***шириной ЯПФ***, а количество ярусов — ***высотой ЯПФ***.

Вершины, расположенные на первом ярусе (не имеющие входных дуг), называют ***входными вершинами***, расположенные на последнем ярусе (не имеющие выходных дуг), — ***выходными вершинами***.

Вершины, расположенные в ЯПФ на одном ярусе, обозначают команды, не зависящие друг от друга, т. е. команды, которые могут быть выполнены параллельно. Это означает, что ширина ЯПФ показывает максимально возможную степень распараллеливания алгоритма — максимальное количество вычислительных ядер, которое может быть задействовано при выполнении этого алгоритма. Увеличение количества вычислительных ядер сверх этой величины дополнительного выигрыша уже не даст.

Для простоты будем считать, что все команды, расположенные на одном ярусе, выполняются с примерно одинаковой скоростью. Тогда команды, находящиеся на одном ярусе, будут выполняться одновременно. А высота ЯПФ характеризует максимальную возможную скорость (минимальное возможное время) выполнения алгоритма при условии его максимального распараллеливания (т. е. при количестве вычислительных ядер, равном ширине ЯПФ).

Так, при наличии четырех вычислительных ядер программа 3 может быть выполнена за пять единиц времени. Быстрее выполнить данный алгоритм невозможно!

(Еще раз обратим внимание на принятое в данной модели положение о равенстве времени выполнения всех команд.)

Важный момент заключается в том, что после построения ярусно-параллельной формы мы можем уже не обращать внимания на конкретные команды, записанные в той или иной вершине графа. Все, что нам надо знать, — это информация о связях между вершинами. А она отражена в структуре графа. Поэтому дальше мы можем рассуждать в терминах теории графов.

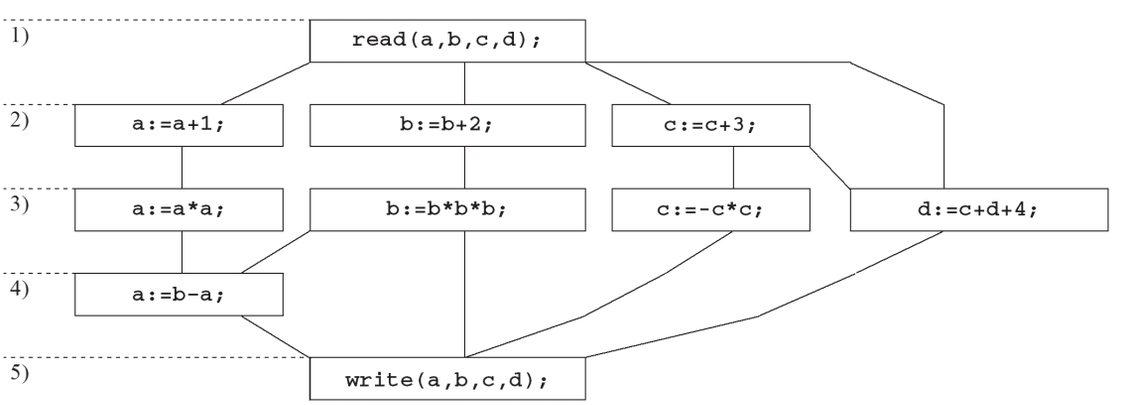

Рис. 3. Параллельная форма программы из примера 3

Вообще говоря, для одного алгоритма можно построить несколько разных параллельных форм. При этом, меняя высоту ЯПФ, мы можем воздействовать на время выполнения алгоритма, а меняя ширину ЯПФ, — на количество требующихся вычислительных ядер.

При построении ЯПФ для примера 3 мы руководствовались правилом:

Каждая команда попадает на ярус, непосредственно следующий за ярусом, на котором закончено вычисление нужных ей данных. Расположение команды на ярусе N означает, что последние необходимые для ее выполнения данные были вычислены на ярусе (N — 1).

Такой подход обеспечил нам построение графа минимальной высоты, но максимальной ширины. Такой параллельный алгоритм будет выполняться максимально быстро, но потребует максимального числа вычислительных ядер.

Легко заметить, что полученный нами граф можно без труда перестроить таким образом, что высота его не изменится, но ширина станет меньше. То есть мы можем при той же скорости выполнения алгоритма уменьшить потребность в вычислительных ядрах! Для этого достаточно переместить с третьего яруса на четвертый вершину «`c:=-c*c;`» (рис. 4).

Высота алгоритма осталась равна пяти, но ширина уменьшилась до трех. Такая перестройка показывает, что для выполнения алгоритма за пять шагов достаточно трех вычислительных ядер.

Дальнейшее уменьшение ширины ЯПФ возможно уже только за счет увеличения ее высоты. Мы можем построить параллельную форму шириной два, но количество ярусов при этом должно возрасти до шести. Это значит, что при наличии двух вычислительных ядер для выполнения алгоритма потребуется шесть шагов.

Заметим, что представление алгоритма в форме графа позволяет легко избежать ошибки, которая в данном случае напрашивается при взгляде на алгоритм как на линейную последовательность команд. «Линейный» взгляд подсказывает следующую цепочку рассуждений: «В алгоритме 10 команд. У нас два вычислительных ядра. За сколько шагов может быть выполнен алгоритм? 10 : 2 = 5. За пять шагов!»

В строго последовательном алгоритме из примера 1 порядок следования команд строгий. Для любых двух команд однозначно определено, какая из них является предшествующей, какая — следующей (непосредственно или через какое-то число посредников). Таков порядок чисел на числовой оси. В параллельных алгоритмах, которые мы построили для примеров 2 и 3, порядок следования команд — частичный. Некоторые команды не связаны отношением «следует за». Зато «непосредственно предшествующих» и «непосредственно следующих» команд может оказаться больше одной. В нашем примере команды, вычисляющие переменную *a* (левая ветвь графа), никак не связаны с командами, вычисляющими переменные *c* и *d* (правые ветви графа). Команды «`a:=b-a;`» и «`d:=c+d+4;`» имеют не одну непосредственно предшествующую команду, а две. А команда «`write(a,b,c,d);`» — целых четыре. Команда «`read(a,b,c,d);`» имеет четыре непосредственно следующие команды, а команды «`c:=c+3;`» и «`b:=b*b*b;`» — по две.

В каждой вершине ЯПФ размещен некоторый «блок вычислений». В нашем примере в роли такого блока выступал один оператор языка Паскаль. То есть в своей работе по распараллеливанию мы остановились на уровне операторов языка Паскаль. Возникают вопросы: можно ли спускаться ниже? Возможно ли распараллеливание внутри одного паскалевского оператора? Внутри одного выражения?

Рассмотрим пример: пусть мы хотим посчитать сумму восьми элементов вектора:

```
s:=a[1]+a[2]+a[3]+a[4]+a[5]+a[6]+a[7]+a[8]
```

На вычислительной машине с одним арифметико-логическим устройством сложение будет выполнено в семь приемов. Но если у нас есть система с четырьмя вычислительными ядрами, мы можем изменить схему счета:

1) сначала сложить попарно рядом стоящие элементы массива: `a[1]+a[2]`, `a[3]+a[4]`, `a[5]+a[6]`, `a[7]+a[8]`;
2) затем сложить попарно четыре полученные на первом шаге суммы;
3) и, наконец, сложить две суммы, полученные на втором шаге.

Вместо семи шагов нам понадобится только три.

Пример показывает, что распараллеливание возможно не только на уровне операторов, но и на уровне более мелких единиц.

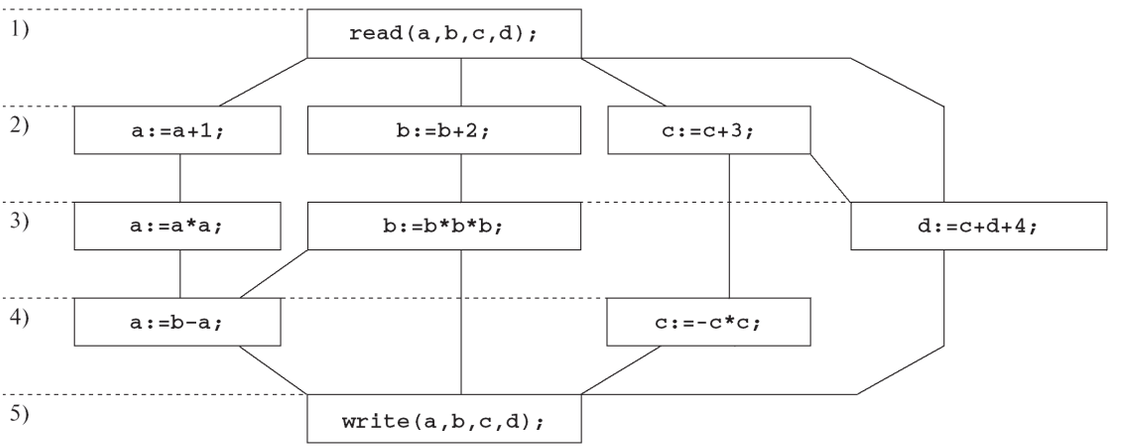

Рис. 4. Параллельная форма программы из примера 3, сжатая до ширины 3

```
s[1]:=0;
for k:=1 to 8 do
 s[1]:=s[1]+a[1,k]
```

```
s[2]:=0;
for k2:=1 to 8 do
 s[2]:=s[2]+a[2,k]
```

```
s[3]:=0;
for k:=1 to 8 do
 s[3]:=s[3]+a[3,k]
```

Рис. 5. Параллельная форма программы из примера 5

```
s[1]:=a[1,1]+a[1,2]+
      a[1,3]+a[1,4]+
      a[1,5]+a[1,6]+
      a[1,7]+a[1,8];
```

```
s[2]:=a[2,1]+a[1,2]+
      a[2,3]+a[2,4]+
      a[2,5]+a[2,6]+
      a[2,7]+a[2,8];
```

```
s[3]:=a[3,1]+a[3,2]+
      a[3,3]+a[3,4]+
      a[3,5]+a[3,6]+
      a[3,7]+a[3,8];
```

Рис. 6. Параллельная форма программы из примера 5, в которой циклы развернуты в линию

С другой стороны, распараллеливание можно «поднять» на уровень единиц более крупных, чем отдельные операторы. Пусть наша программа состоит из последовательности вызовов процедур. Тогда в роли «команды» при построении параллельной формы алгоритма выступит вызов процедуры. И каждая процедура будет запущена на выполнение на своем вычислительном ядре.

Таким образом «вычислительные блоки» для распараллеливания мы можем выбирать сами. При этом логично позаботиться о том, чтобы время выполнения параллельно выполняющихся блоков было примерно одинаковым.

До сих пор все рассуждения мы проводили на примере линейных алгоритмов. Как быть в случаях с более сложными алгоритмическими структурами: развилками, циклами, обработкой исключительных ситуаций и пр.? До какой степени наши рассуждения применимы в этих случаях?

В чистом виде они применимы только тогда, когда программа формально содержит нелинейные структуры управления, но фактически может быть сведена к линейной.

Например, пусть нам надо посчитать суммы элементов каждой строки матрицы и записать их в вектор. Если размеры матрицы известны (например, 3×8), то сделать это можно с помощью фрагмента программы, записанного в примере 4.

Пример 4.

```
for k1:=1 to 3 do begin
  s[k1]:=0;
  for k2:=1 to 8 do
    s[k1]:=s[k1]+a[k1,k2]
end; {for k1}
```

Распараллелить этот фрагмент очень легко, если заметить, что каждая строка матрицы суммируется независимо от других строк. Мы будем иметь три параллельных линии (рис. 5).

Каждая из линий, в свою очередь, представляет собой задачу суммирования. Количество слагаемых — известно. Поэтому ничто не мешает нам развернуть цикл, решить задачу линейно (рис. 6).

Теперь осталось распараллелить вычисление каждой из сумм.

В более сложных случаях, когда линейная развертка программы невозможна (например, при динамическом определении количества повторений цикла), невозможно и статическое построение ЯПФ. Это не значит, что граф программы нельзя построить вообще. Просто строить его приходится динамически. Это предмет отдельного разговора.

Разбор задачи «Джин-строй»

Для ответа на заданные в задаче вопросы необходимо построить ярусно-параллельную форму алгоритма возведения дворца.

Выглядеть она будет так, как представлено на рисунке 7.

Высота ярусно-параллельной формы равна 13, максимальная ширина (на первом и девятом ярусах) — 4.

Напомним, что высота ЯПФ определяет минимально возможное время выполнения алгоритма, максимальная ширина — максимально возможную степень распараллеливания. То есть для нашего случая минимальное время выполнения алгоритма равно 13, а максимально возможная степень распараллеливания (т. е. максимальное количество одновременно работающих джинов) равна 4.

Легко заметить, что ЯПФ можно без труда перестроить таким образом, чтобы сохранить ее высоту, но ширину уменьшить до трех. Для этого достаточно вершины 5 и 3 переместить на более высокие ярусы, ширина которых меньше трех. Как следствие, придется переместить и вершину 1. Но это не страшно. Для этого перемещения места на графе тоже хватает. ЯПФ будет выглядеть, например, так, как показано на рисунке 8.

Дальнейшее сужение графа (до ширины 2) без увеличения его высоты уже невозможно. Камнем преткнове-

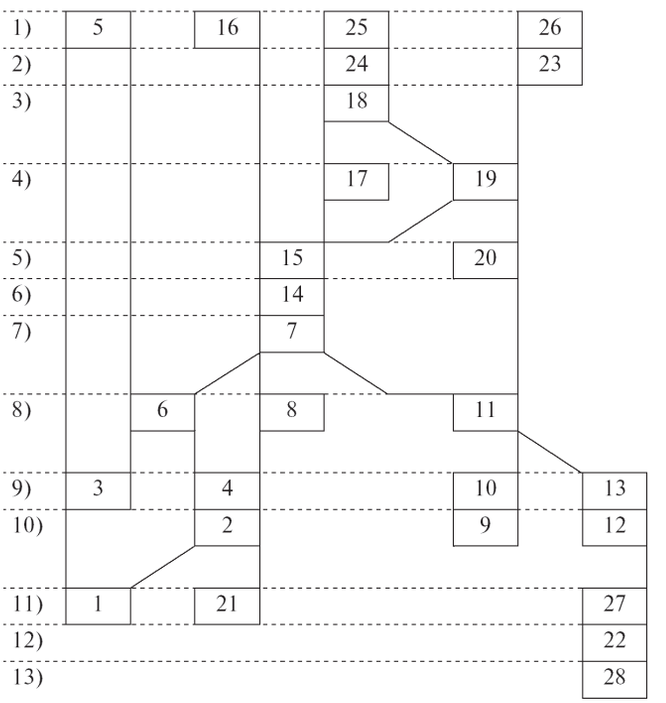

Рис. 7. Параллельная форма алгоритма возведения дворца

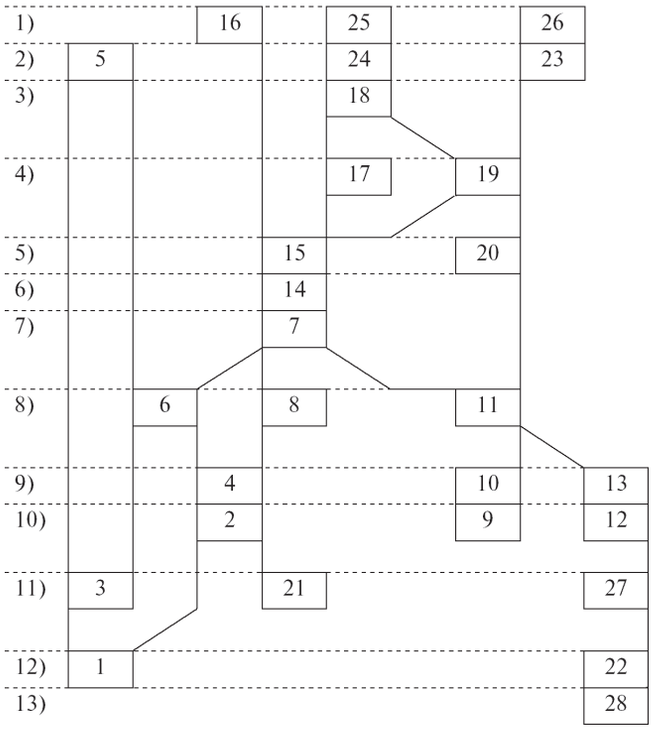

Рис. 8. Параллельная форма алгоритма возведения дворца, сжатая до ширины 3

ния здесь станет ярус 7 (вершина 7). Выше него сужение возможно. В верхней части графа находятся два яруса шириной 3 (ярусы 1 и 2), но и два яруса шириной 1 (ярусы 3 и 6), куда можно передвинуть избыточные вершины со слишком широких ярусов. Да и ярус 7 состоит из одной вершины. Мы можем, например, вершину 16 поднять на уровень 3 (к вершине 18), а вершину 5 — на ярус 6 (к вершине 14). Однако ниже яруса 7 мы имеем четыре яруса шириной три (ярусы 8–11), один ярус шириной 2 (ярус 12) и только один ярус шириной 1 (ярус 13). Нам надо уменьшить ширину четырех ярусов (ярусов 8–11). А увеличить мы можем ширину только одного (яруса 13). Не хватает места для трех вершин! При ширине 2 для их размещения потребуется еще два яруса. То есть при ширине 2 граф будет иметь высоту 15.

Ответим на поставленные в задаче вопросы.

Старший сын свое обещание построить дворец за неделю выполнить не сможет. Высота ЯПФ равна 13. Завершить строительство быстрее невозможно независимо от количества строителей.

На схеме дворца критический путь — от камней 25 и 26 до камня 28.

Длина критического пути — 13:

1) 25, 26
2) 23, 24
3) 16, 18
4) 17, 19
5) 15, 20
6) 14
7) 7
8) 11
9) 13
10) 12
11) 27
12) 22
13) 28

Значит, быстрее, чем за 13 дней, построить дворец нельзя при любом количестве рабочей силы.

Средний сын, чтобы построить дворец за две недели, должен призвать трех джинов.

Выше мы провели сужение ярусно-параллельной формы до ширины 3 при сохранении высоты в 13 и показали невозможность ее сужения до ширины 2 без увеличения высоты до 15 (каковая для нас уже является недопустимо большой).

В задании здесь поставлена ловушка, связанная с «линейным» восприятием процесса строительства. «Линейное» восприятие подсказывает такую цепочку рассуждений: «Нам надо установить 28 камней за 14 дней. 28 : 14 = 2. Ответ: для строительства нужны два джина». Аналогичную ситуацию мы рассматривали при анализе ЯПФ для примера 3.

Ловушка в том, что при движении по критическому пути один раз возникает состояние, когда одновременно может работать только один джин. Второй будет простаивать.

На критическом пути лежит камень 7. Положить его можно только на седьмом шаге. На предыдущих шести шагах могут работать сразу два джина. Для первых пяти шагов эта работа является частью критического пути. Для шестого шага у нас есть независимый камень 5.

Но после этого складывается ситуация, когда *все* остальные камни опираются на камень 7. Положить их ранее или одновременно с камнем 7 невозможно. Значит, на седьмом ходе будет уложен только один камень — камень 7. А всего за первые семь ходов будут уложены 13 камней. То есть останется 15 камней, 7 ходов и 2 джина. За 7 дней 2 джина могут уложить только 14 камней. Один камень остается неуложенным.

Другое дело, три джина.

До седьмого хода включительно третьему джину можно ничего не делать. Там справятся двое. А вот начиная с восьмого хода без третьего работника не обойтись:

1) 25, 26
2) 23, 24
3) 16, 18
4) 17, 19
5) 15, 20
6) 5, 14
7) 7
8) 6, 8, 11
9) 3, 4, 13
10) 2, 10, 12
11) 1, 21, 27
12) 9, 22
13) 28

Впрочем, третий джин может быть задействован с самого начала. Например, так:

1) 16, 25, 26
2) 5, 23, 24
3) 18
4) 17, 19
5) 15, 20
6) 14
7) 7
8) 6, 8, 11
9) 3, 4, 13
10) 2, 10, 12
11) 1, 21, 27
12) 9, 22
13) 28

Младший сын должен призвать двух джинов.

Один джин построит дворец за 28 дней (по одному камню в день) и получит за это 56 монет. То есть через 28 дней дворец будет готов и будет стоить 56 золотых.

Два джина построят дворец за 15 дней и получат за это 60 золотых. После этого дворец начнет приносить доход по одному золотому в день. За (28 – 15) = 13 дней доход составит 13 золотых. То есть через 28 дней окажется, что дворец стоил (60 – 13) = 47 золотых.

Три джина построят дворец за 13 дней и получат за это 78 золотых. После этого дворец начнет приносить доход по одному золотому в день. За (28 – 13) = 15 дней доход составит 15 золотых. То есть через 28 дней окажется, что дворец стоил (78 – 15) = 63 золотых.

Можно посчитать срок окупаемости дворца. В конкурсной задаче этого не требовалось, но можно рассматривать срок окупаемости для оценки стоимости постройки дворца.

Дворец приносит доход в один золотой в день. Через сколько дней окупятся затраты на строительство дворца?

Один джин построит дворец за 28 дней и получит за это 56 монет. Дворец начнет приносить доход через 28 дней. Доход в 56 монет будет получен за 56 дней. Значит, срок окупаемости равен 28 + 56 = 84 дня.

Два джина построят дворец за 15 дней и получат за это 60 золотых. Дворец начнет приносить доход через 15 дней. Доход в 60 монет будет получен за 60 дней. Значит, срок окупаемости равен 15 + 60 = 75 дней.

Три джина построят дворец за 13 дней и получат за это 78 золотых. Дворец начнет приносить доход через 13 дней. Доход в 78 монет будет получен за 78 дней. Значит, срок окупаемости равен 13 + 78 = 91 день.

* * *

Мы разобрали задачу на ЯПФ из конкурса «ТРИЗформашка-2014». Близкие к ней задачи были использованы в конкурсах «ТРИЗформашка-2015» и «ТРИЗформашка-2016». Но эти задачи имеют отличия, заслуживающие отдельного обсуждения.

* * *

В заключение процитируем мысль одного из «отцов-основателей» «суперкомпьютерной науки» в России доктора физ.-мат. наук, профессора, члена-корреспондента АН СССР и академика РАН В. В. Воеводина [1, с. 51]:

«До сих пор специалистов в области вычислительной математики учили, как решать задачи математически правильно. Теперь надо к тому же учить, как решать задачи эффективно на современной вычислительной технике. А это совсем другая наука, математическая по своей сути, но которую пока почти не изучают в вузах.

Для успешного решения задач на вычислительных системах параллельной архитектуры приходится привлекать принципиально новые сведения о структуре алгоритмов на уровне связей отдельных операций между собой. В первую очередь необходимо знать множества операций, которые можно выполнять независимо. Другими словами, нужны сведения как раз о тех параллельных формах алгоритмов, о которых говорилось выше».

Литература

1. *Воеводин В. В.* Вычислительная математика и структура алгоритмов: 10 лекций о том, почему трудно решать задачи на вычислительных системах параллельной архитектуры и что надо знать дополнительно, чтобы успешно преодолевать эти трудности: учебник. М.: Изд-во МГУ, 2010.

2. *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002.

3. *Иванова Н. Г., Плаксин М. А., Русакова О. Л.* Задачи на параллельное программирование в конкурсе «ТРИЗформашка-2013» // Информационные технологии в образовании. XXIII Международная конференция-выставка: сборник трудов. Ч. II. М.: Издательский отдел факультета ВМК МГУ имени М. В. Ломоносова, 2013.

4. *Иванова Н. Г., Плаксин М. А., Русакова О. Л.* Конкурс «ТРИЗформашка» как площадка для апробации заданий на параллельное программирование // Информатика в школе: прошлое, настоящее и будущее: Материалы Всеросс. науч.-метод. конф. по вопросам применения ИКТ в образовании, 6–7 февраля 2014 года. Пермь: Перм. гос. нац. иссл. ун-т, 2014.

5. *Иванова Н. Г., Плаксин М. А., Русакова О. Л.* ТРИЗформашка // Информатика. 2010. № 5.

6. *Плаксин М. А., Иванова Н. Г., Русакова О. Л.* Информатика: учебник для 3 класса. М.: БИНОМ. Лаборатория знаний, 2013.

7. *Плаксин М. А., Иванова Н. Г., Русакова О. Л.* Информатика: учебник для 4 класса. В 2 ч. М.: БИНОМ. Лаборатория знаний, 2013.

8. *Плаксин М. А., Иванова Н. Г., Русакова О. Л.* Набор заданий для знакомства с параллельными вычислениями в конкурсе «ТРИЗформашка» // Преподавание информационных технологий в Российской Федерации: Материалы Тринадцатой открытой всероссийской конференции «ИТ-Образование-2015» (г. Пермь, 14–15 мая 2015 года). Пермь: Перм. гос. нац. иссл. ун-т, 2015.