АВТОМАТИЗАЦИЯ ДЕКОМПОЗИЦИИ ПРОГРАММ ДЛЯ РАСПРЕДЕЛЛЕННОГО ВЫПОЛНЕНИЯ

В. Е. МАРЛЕЙ, В. И. ВОРОБЬЕВ, Р. А. КРЫЛОВ, М. Ю. ПЕТРОВ

Санкт-Петербургский институт информатики и автоматизации РАН

СПИИРАН, 14-я линия ВО, д. 39, Санкт-Петербург, 199178

<vmarley@iias.spb.su>

УДК 681.3

Марлей В. Е., Воробьев В. И., Крылов Р. А., Петров М. Ю. **Автоматизация декомпозиции программ для распределенного выполнения** // Труды СПИИРАН. Вып. 3, т. 2. — СПб.: Наука, 2006.

Аннотация. *Рассматриваются принципы построения средств автоматической декомпозиции последовательных программ на основе графа их максимально распараллеленной формы, получаемой из графа информационных потоков. Разработан макет системы автоматизированной декомпозиции программ, написанных на языке С. На исходный текст накладываются ограничения структурного программирования.* — Библ. 11 назв.

UDC 681.3

Marley V. E., Vorobiev V. I., Krylov R. A. , Petrov M. Y. **Automation of software decomposition for its distributed execution** // SPIIRAS Proceedings. Issue 3, vol. 2. — SPb.: Nauka, 2006.

Abstract. *Principles of design of automated decomposition tools for sequential software are considered on the basis of its parallels form, reached from information streams graph. Template (prototype) of automated decomposition system for software compiled in C is developed, source text being under restrictions of structure programming.* — Bibl.11 items.

1. Введение — обзор существующего положения

Интенсивное развитие технологий высокопроизводительных параллельных вычислений привело в настоящий момент к тому, что даже рядовые пользователи, не имеющие дорогостоящих высокопроизводительных вычислителей, получили возможность решать ресурсоемкие задачи.

Один из технологических подходов к программированию для параллельных вычислительных систем заключается в явном описании всех возможных параллельных ветвей в соответствующей программной системе. С помощью такого подхода осуществляется программирование на стандартных и широко распространенных языках программирования с использованием высокоуровневых коммуникационных библиотек и интерфейсов (API) для организации межпроцессного взаимодействия. Наиболее распространенными здесь являются системы программирования на основе передачи сообщений (MPI, PVM).

Другой подход заключается во введении специальных «распараллеливающих» конструкций в язык программирования. При этом могут создаваться оригинальные параллельные языки или параллельные расширения существующих (с сохранением преемственности). Примерами таких языков являются разработанные в ИПМ им. Келдыша РАН языки НОРМА и DVM.

Третий подход состоит в том, что программист явно не указывает, какие части программы нужно выполнять параллельно. Программирование осуществляется обычным способом на стандартных языках. При этом возможны два варианта:

- в качестве конструктивных элементов используются заранее распараллеленные процедуры из специализированных библиотек,

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

- полученная «последовательная» программа преобразуется в «параллельную» с помощью средств автоматического распараллеливания последовательных программ.

Для проектирования параллельных программ возможно также использование инструментальных средств (CASE-систем), использование которых позволяет осуществлять процесс распараллеливания на этапе проектирования с помощью визуальных средств программирования (языки IDEF, UML).

Технология Грид вычислений подразумевает взаимодействие множества ресурсов, гетерогенных по своей природе и географически удаленных [1]. Количество объединяемых ресурсов может быть от нескольких элементов до нескольких тысяч и более. При этом возникает потенциальная возможность снижения производительности по мере наращивания ресурсов. Следовательно, приложения, которые требуют для своего решения объединения большого числа географически удаленных ресурсов, должны разрабатываться таким образом, чтобы быть минимально чувствительными к времени задержки. При объединении большого количества ресурсов отказы элементов являются не исключением, а правилом. Поэтому управление ресурсами или приложениями должно осуществляться динамически, чтобы извлечь максимум производительности из доступных в данное время ресурсов и сервисов [2]. Таким образом, основными свойствами Грид технологии являются:

- распределенность по природе,
- возможность динамической конфигурации,
- неоднородность (гетерогенность) структуры,
- защищенность программ и данных,
- возможность объединения ресурсов различных организаций.

В течение последних лет на основе многоуровневой модели Грид архитектуры (по аналогии с моделью ISO/OSI) разработаны протоколы, сервисы и инструменты, позволяющие создавать среду разделения ресурсов, обладающую необходимыми свойствами, Структура модели и соотношение с многоуровневой архитектурой Internet-протоколов приведены на рис. 1 [3].

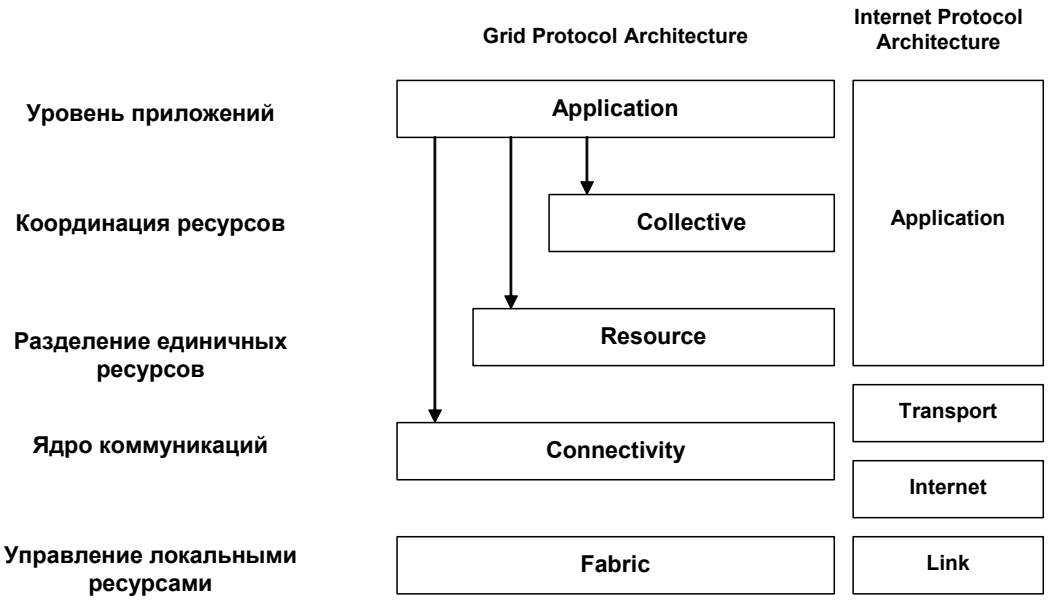


Рис. 1. Многоуровневая модель Грид архитектуры.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Ядром многоуровневой модели являются протоколы Resource и Connectivity, на которые возложены функции обеспечения разделения индивидуальных ресурсов. Уровень Collective отвечает за координацию использования имеющихся ресурсов и безопасность, доступ к ним осуществляется с помощью протоколов Fabric.

В качестве базовой конструкции узла Грид сети могут выступать векторно-параллельные машины или отдельные компьютеры, но чаще всего применяются кластерные системы, которые, по сравнению с векторными машинами, обладают рядом привлекательных качеств, и, в частности, относительно низкой стоимостью при сопоставимой производительности.

Опыт создания и эксплуатации параллельных машин говорит о том, что, как правило, переход от однопроцессорной конфигурации к многопроцессорной системе из N процессоров не приводит к сокращению времени счета в N раз.

Поскольку, программы пишутся не для эффективного использования вычислительных мощностей, а, прежде всего, для решения конкретной задачи, распараллеливание и распределение вычислительных мощностей является моментом, затрудняющим написание программ. С другой стороны, имеется определенная инерция программистов, привыкших формировать алгоритмы в привычной им последовательной форме, на которую ориентируется большая часть программных средств поддержки процесса разработки программ. Кроме того, построение исходных программ изначально в параллельной форме очень часто привязано к конкретной архитектуре, что объективно тормозит применение параллельных вычислителей. Поэтому зачастую в многопроцессорные ЭВМ и суперкомпьютерные кластеры запускаются программы без распараллеливания, что приводит к неэффективному использованию вычислительных мощностей.

Для ликвидации указанных недостатков, необходимо создание средств автоматического распараллеливания последовательных программ без учета ограничений аппаратной среды. Исходя из этого, будем считать, что организация вычислительного процесса в компьютерной системе, и, в частности, распределение фрагментов программы по процессорам, является делом соответствующей операционной системы и исполнительной среды, а распараллеливание программ должно осуществляться специальной утилитой на этапе создания самой программы. Это позволит не отказываться от существующих и хорошо апробированных технологий разработки последовательных программ, и, вместе с тем, позволит повысить эффективность использования многопроцессорных вычислительных систем.

Известны несколько зарубежных систем автоматического распараллеливания: BERT-77, FORGExplorer, KAP, PIPS, VAST/Parallel и др. К сожалению, подавляющее большинство представленных средств имеют ряд существенных недостатков, в частности, некоторые являются дорогими коммерческими системами, либо трудны в освоении, либо привязаны к языку программирования, не входящему в арсенал современных средств. Поэтому перспективным направлением развития может являться создание систем автоматического распараллеливания, преобразующих исходную последовательную программу в параллельную с использованием интерфейса передачи сообщений. Это даст возможность с одной стороны использовать уже имеющиеся вычислительные установки, оснащенные «параллельными» библиотеками стандарта MPI и MPI-2, а с другой стороны обеспечить распараллеливание алгоритмов путем

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

учета информационных потоков (тонкой информационной структуры программ [1]). Заложенная в стандарте MPI-2 возможность динамического порождения процессов также будет способствовать повышению эффективности параллельных программ.

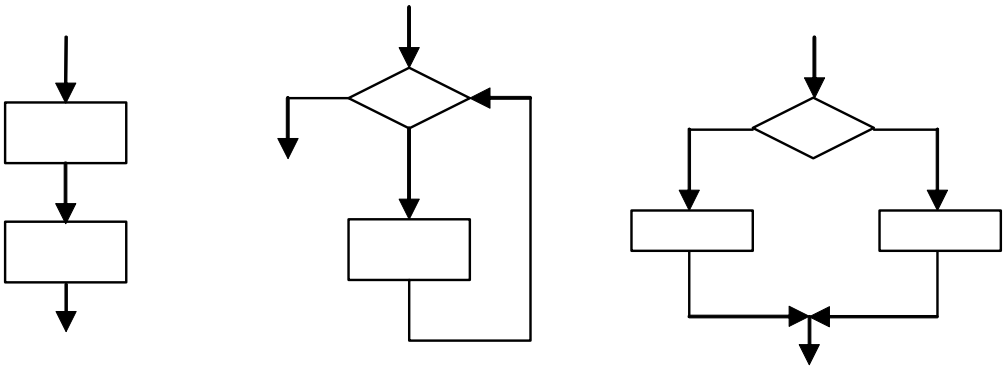
Используя анализ информационных потоков по графовым моделям программ можно выделить в таких программах фрагменты, обладающие естественным параллелизмом [1]. Далее производится распределение выделенных фрагментов по отдельным процессам.

2. Постановка задачи — идея подхода

Предлагаемый подход восходит к ярусно-параллельным графам для планирования решения сложных задач в сети ЭВМ [5]. Эта же идея использовалась для планирования вычислений на алгоритмических сетях [6], которая позволила рассмотреть использование алгоритмических сетей для организации параллельных и распределенных вычислений [7], данная идея использовалась также для автоматизации распараллеливания программ [8]. При формировании распределенной программы учитываются только доступные, заранее известные ресурсы. Данный подход предназначен для процедурно-ориентированных программ, но это не снижает его значимости, так как объектно-ориентированный подход все равно не может обойтись без процедурных программ на нижнем уровне и любую объектно-ориентированную программу всегда можно представить как процедурную задав алгоритм просмотра классов, что опять таки сведется к использованию подходов описанных в [5–8].

Построение информационных связей проще выполнять для линейной части программы, поэтому необходимо свертывать разветвленные участки программы, циклы, сохраняя информацию об информационных связях между получаемыми блоками или возможно также выделять каждый участок с ветвлением или вложенным циклом в подпрограммы [7]. В случае, описанном в [7], результат — линейная программа, в которой каждая переменная алгоритма имеет однозначную интерпретацию в терминах предметной области. Однако обычно в программах однозначность интерпретации в терминах предметной области не соблюдается. Если на начальном этапе составления алгоритма можно однозначно указать входные и расчетные переменные для каждого этапа, в программе этого уже сделать нельзя, так как входная переменная может стать и изменяемой на данном этапе, что усложняет прослеживание информационных связей между блоками. Стиль программирования также оказывает существенное влияние на сложность процедуры свертки участков программы. Наиболее просто это делать для программ использующих только конструкции структурного программирования [9] (рис. 2). Каждый блок в любой конструкции может быть любой из перечисленных конструкций. Множество структурных алгоритмов равномощно множеству всех алгоритмов [9] и имеется процедура позволяющая преобразовать любой произвольный алгоритм в эквивалентный ему по реализуемой функции структурный. Главной особенностью данных конструкций облегчающих свертку является наличие только одной входной и только одной выходной дуги. Данное ограничение на допустимые конструкции оправдано, так как развиваемый используемый в настоящее время стиль программирования ориентирован на использование именно данного подхода, и имеются системы осуществляющие преобразование произвольных программ к структурному виду.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru



а) следование б) цикл в) ветвление

Рис. 2. Конструкции структурного программирования.

Исходя их этого, рассматриваем в дальнейшем только линейные программы.

Считается, что оператору должны предшествовать все те операторы, результаты которых являются для него входными и все операторы, использующие его результаты должны следовать за ним. Далее ранее свернутые операторы могут быть раскрыты, и для их тела процедура может быть повторена и т.д. Отслеживать преобразование программ в параллельный вид легче всего на каком-либо графическом представлении алгоритмов. В качестве графического представления используются параллельные граф-схемы алгоритмов (ПГСА) и процесс распараллеливания последовательной программы визуально представляется как превращение обычной граф-схемы в параллельную. При описываемом подходе структура управляющих связей в ПГСА будет изоморфна структуре информационных связей между операторами.

Для распределенного выполнения требуется осуществить декомпозицию исходной программы. При этом следует учитывать, что наибольшие потери времени происходят при передаче информации от одного фрагмента декомпозированной программы к другому фрагменту. Как показано в [6] число прерываний для передачи информации зависит только от исходной декомпозиции линейной программы. Если декомпозировать программу по выделенным параллельным ветвям, то каждый полученный фрагмент будет выполняться за один просчет без прерываний так как считается, что все данные к моменту начала выполнения параллельной ветви известны. Таким образом, для заданного уровня агрегации, данный вариант декомпозиции обеспечивает минимальное число прерываний. Любой другой вариант декомпозиции может требовать уже большего числа прерываний, так как может нарушить условие готовности информации к началу выполнения фрагмента. Чтобы оценить результат декомпозиции необходимо рассмотреть максимально распараллеленную на данном уровне агрегации программу, тогда будут видны все нарушения готовности данных к началу расчета фрагмента. Таким образом технология декомпозиции программы будет состоять из следующих этапов: агрегация, распараллеливание, собственно декомпозиция, оценка результатов декомпозиции.

Будем считать, что известна структура недекомпозированной максимально распараллеленной программы. Декомпозиция программ для их выполнения в сети может быть управляемая, когда пользователь сам задает фрагменты про-

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

граммы предназначенные для выполнения на разных ЭВМ, и заданная, когда декомпозиция программ не зависит от пользователя, когда рассматривается вопрос организации совместной работы программ принадлежащих разным пользователям. В первом случае необходимо декомпозировать пограмму, во втором оценить качество декомпозиции.

Формальный анализ исходной программы удобно выполнять с использованием языка XML. При этом все операторы языка программирования описываются в виде XML-структур, что позволяет обеспечить представление шаблонов типовых схем распараллеливания, провести исследование и поиск эквивалентных алгоритмов, имеющих программную реализацию. Применяя XSLT процессоры для преобразования исходного алгоритма в соответствующую библиотечную форму, можно свести задачу к выбору шаблонной схемы распараллеливания.

3. Формализация

Рассмотрим основные принципы, которые использовались для формализации задачи автоматизации распараллеливания программ в [10].

Составные операторы могут встречаться где угодно в программе и представляет собой набор последовательно выполняемых операторов. В случае непараллельного кода каждый оператор выполняется сразу после выполнения предыдущего: $op_1 \to op_2 \to \ldots \to op_n$, то есть выполняется обязательное предшествование i-го оператора j-му, при $i < j$. В общем случае отношение полного порядка между операторами не требуется, и оно может быть ослаблено в пользу параллельно выполняемых операторов.

Используя декларации функций и руководствуясь синтаксисом и семантикой, рассматриваемого языка программирования, для каждого оператора находим входные и выходные переменные. Входные переменные — все используемые в операторе переменные, то есть все переменные которые должны быть объявлены до выполнения оператора, выходные все те, которые меняются в процессе выполнения оператора. Таким образом, все выходные переменные являются также и входными, то есть $out(op) \subseteq in(op)$.

Бинарное отношение между операторами, на основе множеств входных и выходных переменных:

$$P_{ij} := i < j \wedge (out(op_i) \cap in(op_j) \neq \varnothing) \vee (in(op_i) \cap out(op_j) \neq \varnothing).$$

Элементы полученной треугольной матрицы P могут рассматриваются как дуги ориентированного графа, в вершинах которого стоят выполняемые операторы. Построив транзитивное замыкание этого графа, получаем матрицу полных связей – отношение частичного порядка на множестве операторов:

$$B_{ij} := \exists k_1,\ldots k_m : P_{ik_1} \wedge P_{k_1k_2} \wedge \ldots \wedge P_{k_{m-1}k_m} \wedge P_{k_mj},$$

то есть существует путь из i-ой вершины в j-ую. B_{ij} означает, что i-ый оператор должен быть выполнен до j-го. Также строится минимальная матрица связей:

$$C_{ij} := B_{ij} \wedge \left\{\forall k \in 1..n\left(\overline{B_{ik} \wedge B_{kj}}\right)\right\},$$

то есть не существует другого пути из i-ой вершины в j-ую, кроме как напрямую. В матрице C отображены лишь основные связи, смысл ее в том, что оператору не требуется дожидаться всех предшественников, а лишь некоторых.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Для любого из предшественников, можно гарантировать, что его предшественники в свою очередь уже завершены, и соответственно их дожидаться не имеет смысла. Таким образом, упрощая связи предшествования удалением всех транзитивных замыканий путей на граф-схеме программы, алгоритм программы не будет нарушен при выполнении.

После того как определены все зависимости выполнения между операторами, требуется априори распределить выполнение по потокам, при этом в данной постановке не учитывается время выполнения каждого из операторов, а лишь порядок выполнения. В каждой точке распараллеливания вычислительных потоков можно создавать новые потоки. Требуется уменьшить число потоков, оставив при этом максимально распараллеленную схему.

Рассмотрим пример. Имеется диаграмма зависимостей выполнения операторов и распределение их по потокам (рис. 3).

Оператор 6 можно смело размещать в первом потоке так, как достоверно известно, что ко времени начала выполнения оператора 6 оператор 2 уже будет завершен, ибо 6 является косвенным предшественником 2. А вот в третий поток мы не можем поставить ни один оператор, так как он не завершается ни одним оператором. Не располагая временем его окончания, не имеем права занимать третий поток вплоть до завершения вычислительного процесса.

Операторы расставляются по потокам за один проход. Изначально подразумеваем, что все потоки свободны, и число их ограничено лишь количеством операторов. Перебирая в порядке очередности все операторы, расставляем их следующим образом:

- Если у очередного оператора нет предшественников, то он ставится в новый поток.
- Если предшественники все-таки есть, то пытаемся поместить в один из потоков, где находится какой-либо из предшественников, при условии что после этого предшественника еще не поставлен оператор.

В противном же случае, пытаемся поставить в каждый поток, начиная с первого, при условии, что в этом потоке нет оператора, ждущего своего.

Теперь рассмотрим декомпозиции программ для распределенных вычислений как декомпозицию максимально-распараллеленной программы.

Рассмотрим в качестве примера схему, ранее приводимую на рис. 3 (рис. 4). Очевидно, что фрагмент, включающий операторы 1, 2, 5 не может быть выполнен без прерываний. Будут выполнены операторы 1, 2 а оператор 5 будет ждать выполнения оператора 3, входящего во фрагмент 3, 4, 7. Фрагмент 3, 4, 7 также не может быть выполнен без прерываний, так как оператор 7 будет ждать выполнения оператора 5.

Таким образом, помещение в один фрагмент операторов относящихся к разным потокам может привести к возникновению необходимости в дополнительных прерываниях. Число этих прерываний соответствуют числу нарушений последовательности выполнения операторов. На рис. 5 приведен пример декомпозиции той же схемы на рис. 3, не приводящей к необходимости дополнительных прерываний.

Однако, хотя пример рассмотренный на рис. 5 не требует дополнительных прерываний, но это не говорит об его большей эффективности при распределенных вычислениях по сравнению с примером на рис. 4, в данном случае исчезает возможность проведения параллельных вычислений, выделенные фрагменты будут выполняться последовательно. Предлагаемый подход в данном случае рассмотрен только как метод оценки варианта декомпозиции с точ-

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

ки зрения числа необходимых прерываний, но может быть применен и для общей оценки эффективности распределенного выполнения программы.

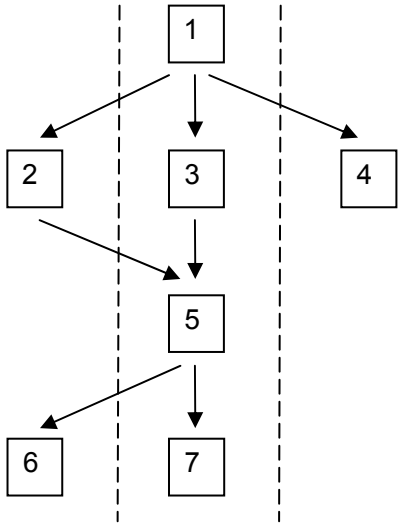


Рис. 3. Диаграмма зависимостей и распределения их по потокам.

То есть при декомпозиции полученные фрагменты не должны, по крайней мере, распадаться на компоненты связности и не должны требовать информации от фрагментов, которые не заканчиваются безусловно раньше начала выполнения данного фрагмента. То есть в треугольной матрице *P* фрагменты должны располагаться компактно.

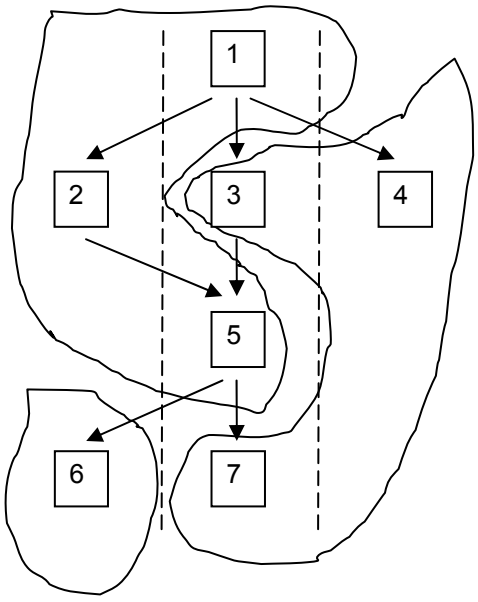


Рис. 4. Пример декомпозиции требующей прерываний вычислений выделенного фрагмента и ожидания дополнительной информации.

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

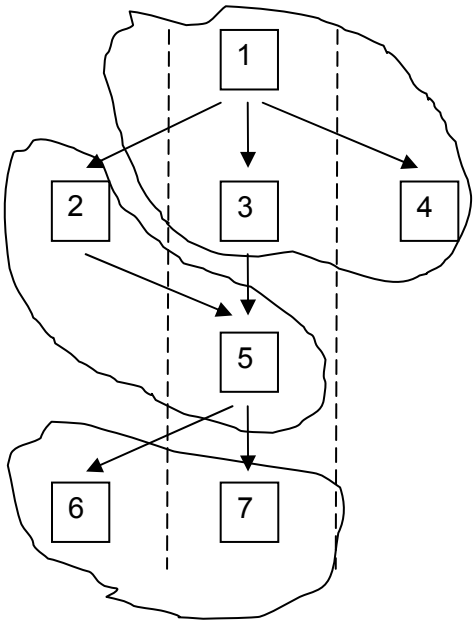


Рис. 5. Пример декомпозиции не требующей прерываний вычислений выделенного фрагмента и ожидания дополнительной информации.

Литература

1. *Шевель А.* Технология Grid // Открытые системы СУБД. 2001. № 2 (58). С. 36–39.
2. *Воеводин В. В., Воеводин Вл.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.
3. *Бараш Л.* Grid Computing — новая парадигма Internet-вычислений // Компьютерное обозрение. 2001. № 32. Также доступно: <http: // www.itc.ua/7249> (по состоянию на 20.05.2005).
4. The Message Passing Interface (MPI) standard [Электронный ресурс] // <http: // www-unix.mcs.anl.gov/mpi/> (по состоянию на 20.05.2005).
5. *Поспелов Д. А.* Введение в теорию вычислительных систем. М.: Советское радио, 1972. 175 с.
6. *Иванищев В. В., Марлей В. Е.* Основы теории алгоритмических сетей. СПб.: СПбГТУ, 2000. 300 с.
7. *Марлей В. Е.* Алгоритмические сети и параллельные вычисления // Труды СПИИРАН. Вып. 1, том 2. СПб.: СПИИРАН, 2002. С. 114–124.
8. *Лабусов А. Н.* Технологии распараллеливания [Электронный ресурс] // <http: // www.spbcas.ru/cfd/techn/Parallel.htm> (по состоянию на 20.05.2005).
9. *Дал У., Дейкстра Э., Хоар К.* Структурное программирование. М.: Мир, 1976. 320 с.
10. *Марлей В. Е., Воробьев В И., Крылов Р. А., Петров М. Ю., Быков Я. А.* Автоматизация распараллеливания программ на основе анализа информационных связей // Программные продукты и системы. 2005. № 1 (69). С. 2–6
11. *Галкин А. А.* Что такое Qt // Русский электронный журнал разработчика [Электронный ресурс] / <http://os2.in.ru/rdm2/articles/qt/> (по состоянию на 20.05.2005).

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru