АВТОМАТИЗАЦИЯ РАСПАРАЛЛЕЛИВАНИЯ ПРОГРАММ

В. Е. Марлей, В. И. Воробьев, Р. А. Крылов, М. Ю. Петров,
Я. А. Быков

Санкт-Петербургский институт информатики и автоматизации РАН
199178, Санкт-Петербург, 14-я линия, В.О., д.39
<vmarley@iias.spb.su>

УДК 681.3
В. Е. Марлей, В. И. Воробьев, Р. А. Крылов, М. Ю. Петров, Я. А. Быков. **Автоматизация распараллеливания программ** // Труды СПИИРАН, Вып. 2, т. 2. — СПб.: Наука, 2005.
Аннотация. *Рассматриваются принципы построения средств автоматического распараллеливания последовательных программ на основе графа программы и графа информационных потоков. Разработан макет системы автоматизированного распараллеливания программ, написанных на языке С. На исходный текст накладываются ограничения структурного программирования.* — Библ. 5 назв.

UDC 681.3
V. E. Marley, V. I. Vorobiev, R. A. Krylov, M. Y. Petrov, Y. A. Bykov. **Automation of software paralleling** // SPIIRAS Proceedings. Issue 2, vol. 2. — SPb.: Nauka, 2005.
Abstract. *Principles of design of automated paralleling tools for sequential software are considered on the basis of software graph and information streams graph. Template (prototype) of automated paralleling system for software compiled in C is developed, source text being under restrictions of structure programming.* — Bibl. 5 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. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. 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), которые позволяют осуществлять процесс распараллеливания на этапе проектирования с помощью визуальных средств программирования (языки IDEF, UML).

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

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

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.

В данной работе изложены принципы построения средств автоматического распараллеливания последовательных программ, которые легли в основу разработанной системы автоматического распараллеливания программ, написанных на языке С.

2. Постановка задачи — идея подхода

Предлагаемый подход восходит к ярусно-параллельным графам для планирования решения сложных задач в сети ЭВМ [2]. Эта же идея использовалась для планирования вычислений на алгоритмических сетях [3], которая позволила рассмотреть использование алгоритмических сетей для организации параллельных вычислений [4]. При распараллеливании программы не учитываются возможные аппаратные ограничения.

Поскольку построение информационных связей проще выполнять для линейной части программы, необходимо свертывать разветвленные участки программы, циклы, сохраняя информацию об информационных связях между получаемыми блоками. Возможно также выделять каждый участок с ветвлением или вложенным циклом в подпрограммы [4]. Результат - линейная программа с требованием, чтобы каждая переменная алгоритма имела однозначную интерпретацию в терминах предметной области. Однако обычно в программах однозначность интерпретации в терминах предметной области не соблюдается.

Процесс создания программы распадается на несколько этапов. Разбиение на этапы достаточно субъективно, они не всегда наблюдаются в явном виде. На начальном этапе человек составляет некоторый сценарий процесса решения, который потом стремится формализовать. Далее определяются «предметные» переменные, то есть переменные, в которых может быть представлен процесс и действия над ними, соответствующие отдельным этапам формируемого алгоритма. Далее, формируется первоначальный вариант алгоритма, описанный в терминах предметной области, задающий функциональную спецификацию решаемой задачи. Каждая переменная в таком алгоритме будет вычисляться только на одном из шагов, и множества входных и изменяемых переменных на каждом этапе не будут пересекаться. Появление неоднозначности или зацикливания будет означать, что исходный сценарий был сформирован неверно и требует коррекции. На следующем этапе необходимо присвоить переменным имена, соответствующие правилам выбранного языка программирования, привести запись математических выражений в соответствие с его синтаксисом, добавить часть соответствующую эксплуатационной спецификации, отражающую особенности применяемого языка программирования и используемой ЭВМ. На данном этапе в процессе формирования кода программы обычно производится оптимизация исходного алгоритма, включающая, в частности, сокращение используемых идентификаторов переменных, что приведет к потере однозначной интерпретации имен переменных в терминах предметных переменных. Таким образом, если в первоначальном алгоритме всегда можно было однозначно указать входные и расчетные переменные для каждого этапа, в программе этого уже сделать нельзя, так как входная переменная может

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

стать и изменяемой на данном этапе, что усложняет прослеживание информационных связей между блоками.

С другой стороны стиль программирования может оказать существенное влияние на сложность процедуры свертки участков программы. Наиболее просто это делать для программ использующих только конструкции структурного программирования [5]: следование, ветвление, цикл (рис. 1). Каждый блок в любой конструкции может быть любой из перечисленных конструкций. Как показано в [5] множество структурных алгоритмов равномощно множеству всех алгоритмов 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 существующий в настоящее время стиль программирования использует идеи именно данного подхода, а для программ написанным в другом стиле имеются системы осуществляющих преобразование их структурному виду. 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. Однако такая структура будет избыточна. Так как если некоторый оператор выполняется после какого-либо другого, и между ними выполняется путь, включающий ряд других операторов, то информационную связь между ними можно опустить, так как к моменту его выполнения все исходные данные будут уже готовы. Таким образом можно редуцировать все транзитивные замыкания путей в полученной ПГСА. Это позволяет сделать представление более наглядным и упростить формирование параллельных процессов.

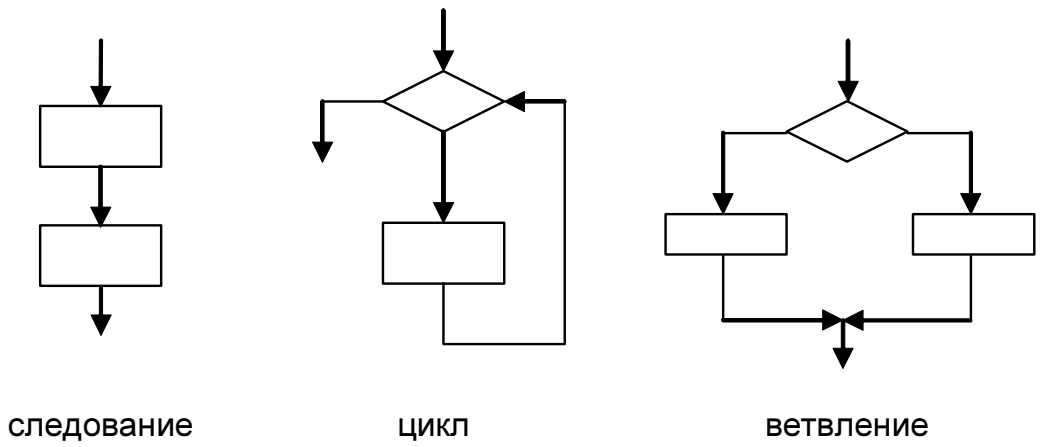

Рис. 1. Конструкции структурного программирования.

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

3. Формализация

Предложенный подход может быть использован для создания программы автоматизированного распараллеливания программ, написанных на языке C. Рассмотрим упрощенную схему операторов. Любое законченное предложение на языке C называется оператором. Оригинальную грамматику, используемую в программе объемом ~30 страниц не представляется возможным привести в данном изложении, поэтому приведем пример ограничений на синтаксис операторов языка C в форме БНФ:

```
<Оператор>::=<Оператор_Выражение>|<Составной_Оператор>|
            <Оператор ветвления>|<Оператор_Цикла>
<Составной_Оператор>::=<Список_Операторов>
<Список_Операторов>::=<Оператор>|
            <Список_Операторов><Оператор>
```

Составные операторы могут встречаться где угодно в программе, в том числе, внутри операторов цикла и внутри выбирающих операторов. Составной оператор 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 языка C, для каждого оператора находятся входные и выходные переменные. Входными переменными назовем все используемые в операторе переменные, то есть все переменные которые должны быть объявлены до выполнения оператора. Выходными же назовем лишь те, которые меняются в процессе выполнения оператора. Таким образом, все выходные переменные являются также и входными, то есть $out(op) \subseteq in(op)$.

Используя множества входных и выходных переменных, формируется бинарное отношение между операторами:

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

Элементы полученной треугольной матрицы P могут рассматриваться 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:

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

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

то есть не существует другого пути из i-й вершины в j-ю, кроме как напрямую. В матрице C отображены лишь основные связи, смысл ее в том, что оператору не требуется дожидаться всех предшественников, а лишь некоторых. Дожидаясь окончания выполнения какого-либо из предшественников, можно гарантировать, что его предшественники в свою очередь уже будут завершены, и соответственно дожидаться их не имеет смысла. Таким образом, упрощая связи предшествования, алгоритм программы не будет нарушен при выполнении.

После того как определены все зависимости выполнения между операторами, требуется априори распределить выполнение по потокам. В рамках данной работы не анализируется информация о времени выполнения каждого из операторов, а учитывается лишь порядок выполнения. Возможно было бы в каждой точке распараллеливания вычислительных потоков создавать новые потоки, но требуется уменьшить число потоков, оставив при этом максимально распараллеленную схему.

Рассмотрим простой пример. Имеется диаграмма зависимостей выполнения операторов и распределение их по потокам (рис. 2):

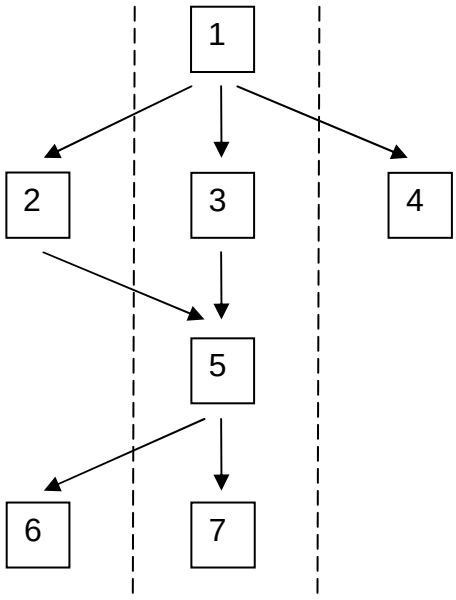

Рис. 2. Диаграмма зависимостей и распределения их по потокам.

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 нельзя 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.

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

- 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.

4. Разработка системы

Рассмотрим конкретные ограничения, накладываемые рассматриваемым языком программирования. Для корректной работы разработанной программы исходный файл на языке C должен удовлетворять некоторым рекомендациям:

- Не поддерживаются операторы безусловного перехода, такие как `goto, continue, break, return`. Это обусловлено начальным предположением о том, что работа ведется только с дейкстровыми конструкциями: циклами, условными операторами и следованиями. Подобные операторы интерпретируются как пустые и становятся независимыми от порядка выполнения, то есть операции с таким кодом могут повлечь изменение алгоритма программы. Другими словами, все действия, выполняемые с кодом, в котором используются операторы безусловного перехода, пользователь выполняет на свой страх и риск.
- Не поддерживается условный оператор `switch`, по причине содержания в нем безусловных операторов `break`.
- Использование указателей несет в себе неопределенности, например, `{*b = 0;}` не меняет переменной `b`, но меняет значение по ее адресу, изменения в памяти не могут быть «предугаданы» на этапе анализа, следовательно, алгоритмические зависимости такого рода могут интерпретироваться неверно. Тоже касается и массивов.
- Необходимо помнить, что код проходит предварительную препроцессорную обработку, все директивы и макросы раскрываются. Следовательно, использование макрокоманд может сильно изменять код после препроцессорной обработки. Это касается в первую очередь таких популярных макросов как `#define min(a,b) ((a)<(b)?(a):(b))`
- Структуры, объединения и перечисления (`structure, union, enumeration`) не поддерживаются в настоящий момент.

Основным предназначением системы являются: исследование программного кода на языке C, автоматизация возможностей распараллеливания кода и исследование параллельных схем базирующихся на стандартных конструкциях.

На первом шаге загружается программный файл на языке C. Загруженный файл проходит препроцессорную обработку, в том числе происходит вставка заголовочных файлов, все макросы также разворачиваются в соответствии с ключами препроцессора. Затем происходит синтаксическая проверка кода. В случае ошибки, выводится сообщение с позицией ошибки. Далее, происходит поиск процедур имеющих тело, все заголовки таких процедур выводятся в отдельное окно.

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

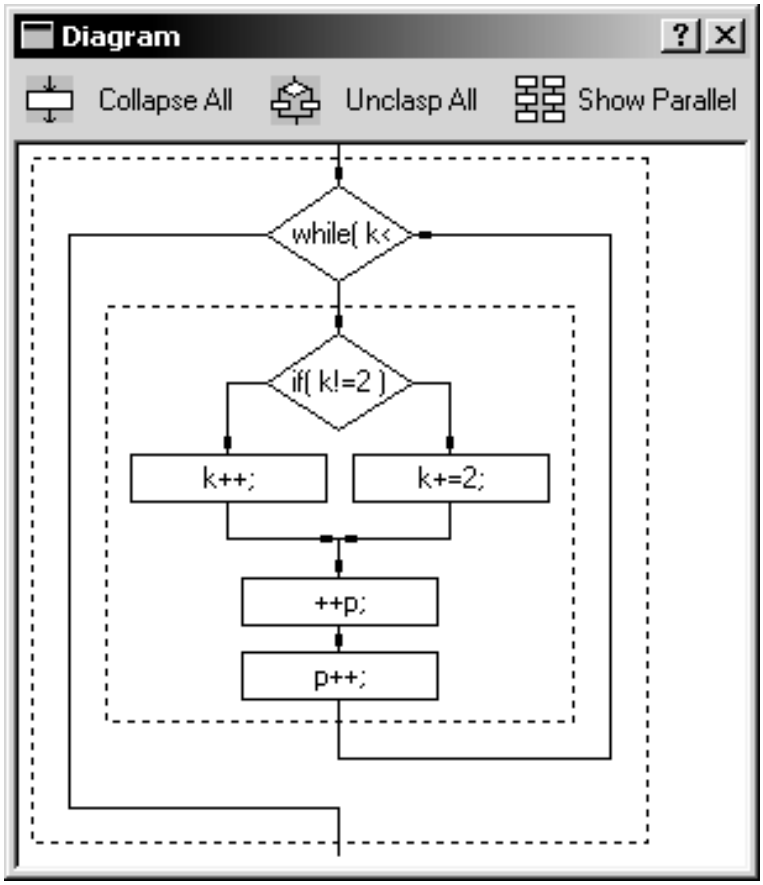

Рис. 3. Структура программы в исходном виде.

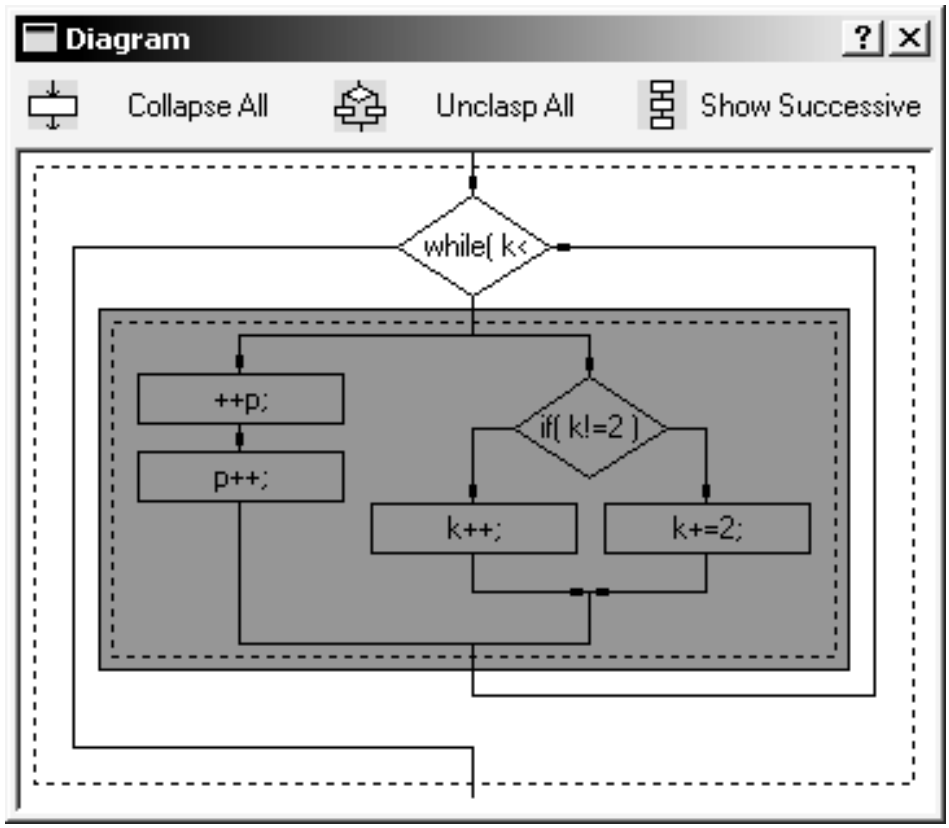

Рис. 4. Распараллеленная структура программного фрагмента.

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Пользователь может выбрать одну из этих процедур, и она предстанет перед ним на экране с использованием синтаксической подсветки в рамках языка.

Пользователь имеет возможность наглядного представления кода в виде блок-схемы, состоящей из простейших конструкций: циклы, условные операторы, составные операторы, выражения. Для каждой цепочки прямого следования, в пределах составного оператора, выделяются связи между операторами, построенные на основе знания о входных и выходных переменных каждого отдельно взятого оператора, которые в свою очередь берутся из деклараций функций, подключаемых заголовочных файлов и синтаксиса самого языка (рис. 3).

После определения данных зависимостей в имеющейся схеме работы программы может быть предложена параллельная схема аналогичной функциональности, в пределах выбранного составного оператора (рис. 4). Также имеется возможность добавлять связи между операторами, в случае наличия логической связи при использовании глобальных переменных, для уменьшения числа параллельных разветвлений и параллельных веток.

Пользователю предоставляется дополнительная возможность добавления связей между операторами. Эта возможность может понадобиться в случае, если связь между операторами существует, но не может быть определена анализатором. Также она используется для уменьшения параллельного ветвления и количества параллельных веток. Многие атомарные операторы не имеет смысла выносить в параллельные ветки алгоритма, в силу того, что практическое время, потраченное на создание параллельного потока во время выполнения программы, может превышать время выполнения самого атомарного оператора.

Имеется также возможность задать связь между операторами, не имеющими прямого предшествования. В этом случае будет выдано соответствующее предупреждение, и, если пользователь все-таки пожелает, будет добавлена и такая связь, в случае если это возможно. Например, в случае косвенного предшествования, невозможно задать прямую связь между операторами.

В дальнейшем планируется произвести собственно перевод исходного текста в новый программный код с учетом произведенных действий по распараллеливанию.

5. Заключение

Данная программная система является системой советчиком позволяющей пользователю отредактировать текст программы для его параллельной реализации. Однако возможно и другое использование рассматриваемого подхода, а именно - декомпозиция программ для их распределенного расчета. В данном случае это подробно не рассматривается, но доказано, что декомпозиция программы по ее параллельным ветвям обеспечивает минимум необходимых информационных взаимодействий между частями программы. Таким образом, задачей дальнейших исследований является не только распространение предложенного подхода на другие языки моделирования и создание системы оптимизации структуры параллельной программы, учитывающей особенности структуры используемых аппаратных средств, но и их применение в сетевых распределенных вычислениях.

Труды СПИИРАН. 2005. Вып. 2. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2005. Issue 2. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

Литература

[1] *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб.: БХВ-Петербург, 2002, 220 с.

[2] *Поспелов Д.А.* Введение в теорию вычислительных систем. М.: Советское радио, 1972, 175 с.

[3] *Иванищев В.В., Марлей В.Е.* Основы теории алгоритмических сетей. СПб.: СПбГТУ, 2000, 300 с.

[4] *Марлей В.Е.* Алгоритмические сети и параллельные вычисления // Труды СПИИРАН. Вып 1,. Том 2, СПб.: СПИИРАН, 2002. – с.114–124.

[5] *Дал У., Дейкстра Э., Хоар К.* Структурное программирование. М.: Мир, 1976.