МАТЕМАТИЧЕСКОЕ И АРХИТЕКТУРНОЕ ОБЕСПЕЧЕНИЕ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ
(Вычислительные системы)
1985 год Выпуск 109

УДК 519.665:681.324

ПОСТРОЕНИЕ ОДНОЗНАЧНЫХ И БЕСТУПИКОВЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Н.Н. Миренков

I. Введение. В настоящее время предложено большое число моделей параллельных вычислений, отражающих многообразие целей и возможностей, предполагаемых при той или иной организации вычислительных процессов [2-3,8-9,I8-2I]. Вместе с этим быстрое расширение областей применения параллельного программирования требует построения новых моделей, которые, сохраняя лучшие свойства своих предшественников, давали бы широкие возможности для создания корректных и надежных параллельных программ. Особенность этой проблемы состоит в том, что к традиционной сложности логики последовательного программирования добавляется сложность взаимодействия асинхронных процессов. Кроме того, понятия различных аспектов корректности в значительной степени отличаются при переходе от одной модели к другой (см., например, [I0-I3,I7-20]).

В такого рода ситуациях большую роль играют попытки найти новый обобщающий взгляд на проблему и/или стремление перераспределить трудности между различными этапами проектирования программ. Примерами обобщающего взгляда являются работы [9,I6]. В [9] предложена метамодель системы взаимодействующих процессов, из которой многие известные системы синхронизации получаются путем изменения вида функций, выполняемых над семафорами. Это обобщение, однако, касается только средств синхронизации, оно не учитывает примитивы для передачи сообщений, для порождения и уничтожения процессов. В [I6] средства ввода/вывода, сначала предложенные как языковая модель для интерактивных систем, были развиты в модель для взаимодействующих параллельных процессов.

В этих работах не предполагалось установление непосредственной связи между предлагаемыми обобщениями и достижением корректности программ, хотя такая связь могла оказаться весьма интересной. Говоря о корректности параллельных программ, следует отметить работы [II,I5], в которых предложены строгие описания семантики параллельных процессов. Однако вопрос об использовании этих подходов непосредственно в компиляторах и отладчиках остается открытым. И дело здесь не только в принципиальной возможности выявления различных свойств программ, но и в практической реализуемости выявления. Представляется, что часть трудностей, возникающих при выявлении определенных свойств программ, может быть преодолена путем слежения за состоянием процессов во время выполнения программ, например, как это предложено для выявления дедлоков [5,20]. В общем же случае необходимо перераспределение трудностей между различными этапами создания программ. Наиболее простым примером такого перераспределения является переход к структурированному программированию.

Проблема создания корректных и надежных программ требует привлечения дополнительных знаний о сущности решаемых задач, причем знаний, которые, как правило, неявно могут предоставляться транслирующим системам через стандартизованные языковые конструкции. Стремление к использованию таких конструкций в общем случае требует специального проектирования алгоритмов.

В данной работе описываются модели параллельных вычислений, так называемые параллельные алгоритмы (п-алгоритмы), отражающие опыт автора по созданию языков параллельного программирования для экспериментальных многомашинных систем [4,7,I4]. Эти модели представляют собой "асинхронное" обобщение операторной формы последовательного алгоритма [I]. Причем условия включения операторов конкретизируются через канонизированные наборы схем взаимодействий между процессами. Варьируя составом этих схем, можно переходить от одной модели параллельных вычислений к другой. В работе выделяется два набора таких схем, описывающих соответственно индивидуальные и групповые взаимодействия. Эти схемы выявлены при анализе различных классов задач (см.,например, [2,6,8]) и, как было замечено, имеют близкие аналоги в системах связи между людьми.Так, схемы групповых взаимодействий подобны коммуникациям между людьми при радио- и телепередачах, при работе на синхронном конвейере, а схемы индивидуальных взаимодействий - при телефонных разговорах ,

при работе на асинхронном конвейере и т.п. Для п-алгоритмов с указанными схемами взаимодействий формулируются условия, обеспечивающие однозначность и беступиковость порождаемых процессов. Сущность этих условий связывается со специальным проектированием п-алгоритма, при котором его процессы "размечаются" таким образом, что каждый фрагмент вычислений (подпроцесс) информационно и по управлению независим от взаимодействий, происходящих между процессами во время существования подпроцесса; результаты же этих взаимодействий проявляются лишь на последующих фрагментах.

Предлагаемые для описания п-алгоритмов и их разметки языковые средства представляют собой специальные расширения традиционных структур последовательного программирования. Расширения выполнены таким образом, чтобы, с одной стороны, в рамках асинхронных моделей вычислений в значительной мере сохранить преемственность последовательного программирования, а с другой - предоставить возможность для автоматического выявления ряда свойств п-программ по их тексту. Такими свойствами являются однозначность и беступиковость п-алгоритмов в предположении, что все while-do циклы завершаются за ограниченное число итераций.

2. Понятие параллельного алгоритма. Все операции, производимые в связи с решением какой-либо задачи, рассматриваются как выполнение над общей памятью $D = \{d_i, i \in I\}$ множества операторов:

$$S = \{S_k, k \in K\}, \quad S_k: \bar{D} \to \bar{D}, \quad \bar{D} = \prod_{i \in I} \bar{d}_i,$$

где K, I и $\bar{d}_i$ - конечные множества значений k, i и d_i соответственно. Каждый оператор (назовем его с-оператором, подчеркивая его составную структуру) есть тройка $S_k = ([C_k^1]; T_k; C_k^2)$, где $C_k^1 \in C_1$, $C_k^2 \in C_2$ $(C=(C_1 \cup C_2)$ - конечное множество специальных операторов; операторы из C_1 позволяют задавать синхронизацию вычислительных процессов, обмен данными между ними, а операторы из C_2 - назначать с-операторы - преемники), T_k - функция, для простоты трактуемая как программный сегмент, не содержащий операторов из C и выполняющий ограниченное число действий, в частности, сегмент может быть пустым; квадратные скобки означают, что оператор из C_1 на этом месте может отсутствовать. Условием включения (выполнения) с-оператора является конъюнкция двух подусловий, одно из которых характеризует присутствие/отсутствие с-оператора в списке преемников, другое - соответствует условию включения для $C_k^1 \in C$.

Запись любого последовательного алгоритма в такой трактовке представляет собой последовательность с-операторов:

$$S_1; S_2; \ldots; S_h;$$

где $S_i = (T_i; C_i^2)$, а множество $C \equiv C_2 = \{БП, УП, \emptyset\}$, БП и УП - операторы безусловного и условного переходов соответственно; $\emptyset$ - оператор останова. В каждый момент выполняется только один с-оператор, имя (код) которого находится в регистре команд для данного алгоритма; список преемников содержит только один элемент. Сущность регистра команд алгоритма аналогична сущности регистра команд процессора; здесь регистр команд может трактоваться как переменная, значение которой изменяется автоматически и в любой рассматриваемый момент времени равно имени с-оператора, выполняющегося в этот момент.

При вычислениях, не являющихся последовательными алгоритмами, одновременно может выполняться несколько с-операторов. Поэтому будем предполагать, что в нашей модели в любой момент времени может существовать несколько регистров команд, каждый из которых содержит имя одного из выполняющихся с-операторов, а все вместе - имена всех активных в этот момент с-операторов. С каждым регистром команд связывается его собственный список преемников. Здесь следует дополнительно подчеркнуть, что во введенных обозначениях с-оператор есть сокращение от слов "составной оператор", а не оператор из множества C.

ОПРЕДЕЛЕНИЕ I. П-алгоритмом над памятью D называется четверка: $\langle S, S', H, C \rangle$, где S - конечное множество с-операторов над D; S' - подмножество стартовых операторов ($S' \subseteq S$), начинающих выполнение в заданном конечном промежутке времени (t_1, t_2) *) (имя каждого из них появляется на своем регистре команд из H); H - множество стартовых (исходно заданных) регистров команд, взаимно-однозначно соответствующих элементам из S'; C - конечное множество операторов для межпроцессных взаимодействий, в котором операторы из $C_2 \subseteq C$ могут: I) создавать новые регистры команд или уничтожать существующие и 2) назначать преемников на любом подмножестве регистров команд, существующих в момент своего выпол-

*) Этот промежуток разрешает стартовым с-операторам включаться не одновременно, а с некоторым разбросом во времени, что, в частности, отражает реальное различие временных задержек при распространении сигналов между разными модулями вычислительной системы.

нения*). Оператор останова уничтожает регистр команд, на котором он появляется. П-алгоритм завершает выполнение в момент уничтожения последнего существующего регистра команд.

При появлении нескольких с-операторов в списке преемников какого-либо регистра команд окончательное решение о назначении преемника для регистра осуществляет оператор из C_2, входящий в с-оператор, имя которого представлено в данный момент в рассматриваемом регистре команд. Мы для простоты будем предполагать, что приоритет отдается первому из поступивших в список с-операторов, а остальные аннулируются. Другими словами, список преемников отдельного регистра команд может состоять только из одного элемента, попытка записи второго элемента трактуется как пустая операция. Таким образом, при выполнении п-алгоритма общее число преемников в некоторый момент времени не превышает числа регистров команд, существующих в этот момент.

После выполнения с-оператора его имя в соответствующем регистре команд заменяется на имя преемника. Считается, что с этого момента начинает выполняться новый оператор. Последовательность с-операторов, появляющихся на одном регистре команд, определяет ветвь п-алгоритма.

Поскольку с-оператор является составным оператором, то предполагается, что в каждом регистре команд в каждый момент времени, кроме имени с-оператора, содержится имя выполняющейся в этот момент его составной части, т.е. имя оператора из C или оператора из T_k. Набор всех имен, принадлежащих выполняющимся в рассматриваемый момент времени составным частям (и находящихся на разных регистрах команд), называется вектором текущего состояния п-алгоритма.

ОПРЕДЕЛЕНИЕ 2. П-алгоритм называется беступиковым, если условия включения любого оператора из C, попавшего в регистр команд, удовлетворяются после выполнения конечного числа операторов в других ветвях, а вектор текущего состояния никогда не содержит только операторы из C с неудовлетворенными условиями включения.

Под результатом п-алгоритма будем понимать состояние памяти $\bar{D}^f \in \bar{D}$ после выполнения всех его ветвей, а под реализацией (историей) п-алгоритма - совокупность порожденных им ветвей, последовательность операторов, имевшую место в каждой ветви, а также ре-

*) Примеры конкретных операторов из C_2 см. в разделе 3.

зультаты выполнения каждого оператора. Два выполнения п-алгоритма над одними и теми же исходными данными могут давать разные результаты из-за предполагаемого различия скоростей вычислительных процессов и различных моментов запуска стартовых операторов.

ОПРЕДЕЛЕНИЕ 3. П-алгоритм называется однозначным, если любая его реализация на одних и тех же исходных данных дает один и тот же результат.

Нетрудно видеть, что, изменяя состав элементов множества C, можно получить ту или иную модель параллельных вычислений. Например, если операторы выбора преемников будут всегда назначать в качестве преемников все с-операторы из S, то мы будем находиться в классе максимально асинхронных моделей [3]; если же операторы выбора назначают преемников только на своем регистре команд, а условия включения операторов из C тождественно истинны, то мы будем находиться в классе наиболее "жестких" последовательно-параллельных моделей. Одним из наиболее важных факторов, влияющих на выбор C, является архитектура вычислительной системы, на которой будет реализовываться параллельная обработка данных. В следующих разделах описаны два примера множества C, элементы которого ориентированы на модульные многопроцессорные системы, каждый модуль которых имеет в своем составе процессор и память. Такие системы с точки зрения отдельного процесса обладают общей, но иерархической оперативной памятью, один уровень которой есть память модуля, непосредственно реализующего процесс, другой - память всех остальных модулей.

3. Операторы для задания взаимодействий между ветвями п-алгоритма. Здесь и в следующих разделах будем предполагать, что общая память, над которой выполняется п-алгоритм, разбита на конечное число непересекающихся областей: $D = \bigcup_i D_i$. Над областью D_i выполняется только ветвь с именем i, за исключением случаев, когда с ветвью i вступают во взаимодействие, используя операторы из C, другие ветви. Не нарушая общности, будем считать, что роль имени ветви играет ее номер.

Множество C(I) операторов индивидуальных взаимодействий. Описываемые ниже операторы позволяют задавать индивидуальный доступ ветви j к области памяти D_i, $i \neq j$, и изменять порядок выполнения операторов в других ветвях; эти операторы задают взаимодействие не более чем двух ветвей.

Будем предполагать, что в каждый момент выполнения п-алгоритма существует конечный набор векторов, в общем случае различной размерности, компонентами которых являются семафоры, предназначенные для синхронизации процессов п-алгоритма. Пусть X - один из таких векторов, принадлежность компонентов которого областям из D здесь не уточняется. Пусть j есть имя ветви, в которой используются рассматриваемые ниже операторы из C(I).

Оператор ИЗМЕНИТЬ(X,B) прибавляет значения компонентов целочисленного вектора B ($B \in D_j$) (компоненты могут быть положительными и отрицательными) к значениям соответствующих компонентов X; векторы X и B имеют одинаковую размерность; операция изменения X является неделимой.

Оператор ПРОЧИТАТЬ(a,i,a_1,b,B,X) переписывает (вводит) значение переменной $a \in D_i$ на место $a_1 \in D_j$, B и X - описаны выше. Операция чтения реализуется, если выполнено условие, заданное вектором b ($b \in D_j$), размерность которого совпадает с размерностью X. Компоненты b указывают на компоненты X, которые нужно использовать, и на логическую функцию, которая должна быть над выделенными семафорами вычислена. Например, b = (I,0,I,0) может задавать конъюнкцию над нечетными компонентами X, а b = (0,-I,0,-I) - дизъюнкцию над четными.

Оператор ЗАПИСАТЬ(a,i,a_1,b,B,X) переписывает (выводит) значение переменной $a \in D_j$ на место $a_1 \in D_i$; b,B,X - те же, что и в предыдущем случае.

Оба оператора являются неделимыми в том смысле, что они блокируют доступ других ветвей к вектору X на время проверки предиката и изменения значений семафоров, а также к переменным a и a_1 на время пересылки данных. Опыт использования операторов показал, что объединение синхродействия с доступом к общим данным в одной конструкции является удобным с практической точки зрения.

Каждый из описанных выше операторов есть $C_k^1 \in C_1$.

Оператор ВЫПОЛНИТЬ(S_h,i,b,B,X) назначает с-оператор S_h преемником в ветви i и предварительно задает новый регистр команд, если S_h - первый с-оператор, выполняемый в ветви i. Предполагается, что при порождении новой ветви имя i однозначно определяет область D_i, не пересекающуюся с областями существующих ветвей. При $S_h = \emptyset$ ветвь i уничтожается. Данный оператор есть $C_k^2 \in C_2$.

Описанные операторы с точки зрения анализа моделей взаимодействующих процессов, выполненного в [I6], не требуют взаимного ука-

зания имен процессов и статических границ их числа, позволяют обмениваться сообщениями без механизма буферирования, задают схему обмена "1 → 1" . При реализации [I4] в операторах ПРО и ЗАП (здесь и далее имена операторов из C(I) представляются тремя первыми буквами) был добавлен еще один параметр, значение которого разрешало или запрещало выполнять совмещение обменов с дальнейшими вычислениями.

Множество C(G) операторов групповых взаимодействий. Эти операторы, как правило, задают групповой (одновременный) доступ ветвей к любой подобласти памяти или назначают сразу несколько преемников. Условием включения группового взаимодействия является выход каждой ветви на свой оператор, задающий это же взаимодействие.

Оператор ТО(l,A,L_j) задает (ТО - трансляционный обмен) групповой доступ всех ветвей к значению переменной A, l - имя ветви, которая читает $A \in D_j$ без использования операторов из C и транслирует (как по радио) значение A для всех ветвей; L_j ($L_j \in D_j$) - переменная, в которую копируется значение A и к которой ветвь j имеет доступ без использования операторов из C . Оператор, задающий в ветви i это же взаимодействие, имеет вид: ТО(l,A,L_i).

Оператор КО(T,L_j,M_j) задает (КО - конвейерный обмен) сдвиг значений $L_j \in D_j$ на место $M_{j\pm1} \in D_{j\pm1}$, T - определяет направление сдвига, например, по возрастанию номеров ветвей, по убыванию и т.п.

Каждый из этих операторов есть $C^1_k \in C_1$.

Оператор ОУП(b,F_j,S^1_j,S^2_j) задает обобщенный условный переход во всех ветвях, b - логическая функция, ее аргумент $F = \{F_j, j \in J\}$, где $F_j \in D_j$, J - конечное множество ветвей; с-оператор S^1_j является преемником в ветви j, если b(F) = true , иначе преемником считается S^2_j . Этот оператор может принадлежать как C_1 , так и C_2.

Оператор ОБП(List) задает обобщенный безусловный переход одновременно для всех ветвей на с-операторы, указанные в List ; L_i ($L_i \in$ List) задает преемника для ветви i . Для выполнения обобщенного перехода необходимо и достаточно появление ОБП лишь в одной ветви, т.е. не требуется копия этого же оператора в других ветвях. Предполагается, что ОБП при необходимости задает новые регистры команд п-алгоритма и соответствующие им новые ветви. ОБП есть $C^2_k \in C_2$.

Участие в групповом взаимодействии (за исключением ОБП) требует присутствия в каждой ветви соответствующего оператора, однако взаимного указания имен процессов не требуется. В операторе ТО нужно лишь имя ветви-отправителя, в операторах КО, ОУП - имена совсем не используются. Не требуется и задания статических границ числа порождаемых процессов. Схемы взаимодействия ветвей: в ТО и ОБП - "1 → n", в КО и ОУП - "n → n".

Гибкость использования групповых взаимодействий еще более возрастает при введении для процесса статуса "неучастия", позволяющего ветви игнорировать соответствующие операторы (см. следующий раздел). В реализации [I4], кроме возможности совмещения взаимодействий с вычислениями, расширен состав операторов для задания обменов. Использование расширенного состава позволяет неявно управлять реконфигурацией структуры вычислительной системы и повышать эффективность взаимодействий.

Далее будем считать, что операторы ВЫП из C(I) и ОБП из C(G) используются для назначения преемников только в чужих ветвях, в своей же ветви применяются традиционные УП, БП и Ø.

Отметим еще одну особенность описанных операторов. Их выполнение задерживается, если задаваемое взаимодействие предполагает участие в нем ветвей, процессы которых еще не начались; исключение составляют ВЫП и ОБП, задающие порождение или уничтожение этих ветвей.

4. Условие корректности п-программ. При конструировании п-алгоритмов необходимы правила, следование которым обеспечивает корректность и эффективность п-программ. В данном разделе формулируется ряд теорем, выполнение условий которых гарантирует беступиковость и однозначность п-алгоритмов. С учетом этих условий в следующем разделе предлагаются языки для записи п-алгоритмов.

Начнем с рассмотрения п-алгоритмов, использующих операторы из C(I). Не нарушая общности, будем предполагать, что каждый с-оператор содержит в себе, по крайней мере, один оператор из C, а всякое использование оператора из C сопровождается меткой, идентифицирующей с-оператор, которому он принадлежит. Пусть C^m обозначает произвольный элемент из C.

Событием выполнения п-алгоритма назовем тройку $\hat{c} = \langle t, C^m, M \rangle$, где t - момент завершения C^m, имеющего метку M (t - момент на-

ступления события). Пусть при некоторой реализации x п-алгоритма с исходными данными $\bar{D}^0$ имела место последовательность упорядоченных во времени событий*): $\hat{c}_x(\bar{D}^0) = \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_l$, а при реализации ветви i - подпоследовательность $\hat{c}^i_x(\bar{D}^0) = \hat{c}_{i_1}, \hat{c}_{i_2}, \ldots, \hat{c}_{i_1(i)}$; если несколько событий происходили одновременно, то они были упорядочены по возрастанию номеров ветвей. Две последовательности событий $\hat{c}_x(\bar{D}^0)$ и $\hat{c}_y(\bar{D}^0)$, связанные соответственно с реализациями x и y, назовем совпадающими: $\hat{c}_x(\bar{D}^0) = \hat{c}_y(\bar{D}^0)$, если они имеют равную длину, а элементы с одинаковыми номерами - совпадающие имена операторов и соответствующие операторам метки, а также имена ветвей, в которых операторы выполнялись.

Фрагментом (с номером j) реализации ветви i (обозначим Φ_{i_j}) назовем последовательность операторов, выполненных между событиями $\hat{c}_{i_{j-1}}$ и $\hat{c}_{i_j}$ (или между моментом запуска ветви и $\hat{c}_{i_1}$, или между $\hat{c}_{i_1(i)}$ и моментом завершения работы ветви). В данной работе мы будем рассматривать ограниченные п-алгоритмы, у которых каждый Φ_{i_j} включает конечное число действий и, кроме того, конечным является число порождаемых процессов. Вместе с этим заметим, что для ограниченности п-алгоритма с операторами из C(I) и п-алгоритма с операторами из C(G) достаточно, чтобы соответственно операторы ОБП и операторы ВЫП использовались только для порождения и уничтожения конечного числа процессов.

Фрагмент Φ_{i_j} информационно независим от событий в других ветвях, если операторы ПРО и ЗАП, выполненные между $\hat{c}_{i_{j-1}}$ и $\hat{c}_{i_j}$, информационно не связаны с операторами фрагмента (т.е. если области определения и области значения для операторов фрагмента не зависят от результатов операторов ЗАП и не влияют на результаты операторов ПРО). Фрагмент Φ_{i_j} независим по управлению от событий в других ветвях, если между $\hat{c}_{i_{j-1}}$ и $\hat{c}_{i_j}$ либо не было реализации операторов ВЫП, либо было только одно такое событие, которое инициировало выполнение первого после $c_{i_{j-1}}$ оператора.

ТЕОРЕМА I. Для однозначности бесту-пикового п-алгоритма с операторами

*) В понятие исходных данных входит и их разбиение на совокупность областей D_i, упомянутых в разделе 3.

из C(I) достаточно, чтобы для всякого $\bar{D}^0$ и любых реализаций x и y имело место $\hat{c}_x(\bar{D}^0) = \hat{c}_y(\bar{D}^0)$, а всякий фрагмент φ_{i_j} был информационно и по управлению независим от событий в других ветвях.

ДОКАЗАТЕЛЬСТВО. Из $\hat{c}_x(\bar{D}^0) = \hat{c}_y(\bar{D}^0)$ следует, что для реализаций x и y совпадают подпоследовательности событий, связанных с оператором ВЫП. Поскольку стартовые наборы регистров команд (и соответствующие им ветви) одни и те же, а оператор ВЫП явно называет имя порождаемой или уничтожаемой ветви, то число и имена ветвей в x и y совпадают. Из $\hat{c}_x(\bar{D}^0) = \hat{c}_y(\bar{D}^0)$ также следует, что совпадают и подпоследовательности событий одноименных ветвей. Отсюда, с учетом независимости фрагментов по управлению и однозначной связи C^m с использующим его с-оператором, следует совпадение последовательностей с-операторов, выполненных в одноименных ветвях.

Пусть i - имя ветви, в которой выполняется оператор C^m, соответствующий $\hat{c}_1^x$ при реализации x и $\hat{c}_1^y$ при реализации y, а j - имя ветви, с которой C^m задает взаимодействие. Сравним изменения состояния памяти одноименных ветвей при реализациях x и y до моментов t_x и t_y наступления событий $\hat{c}_1^x$ и $\hat{c}_1^y$. Причем сравнение произведем после выполнения одних и тех же операторов. В ветвях, не участвующих во взаимодействии, совпадение очевидно. Совпадение в ветвях с именем i следует из информационной независимости $\varphi_{j_1}^x$ и $\varphi_{i_1}^y$. Наконец, совпадение в ветвях с именем j имеет место, если раздельно рассматривать две непересекающиеся части памяти D_j, первая из которых изменяется операторами из $\varphi_{j_1}^x$ (или $\varphi_{j_1}^y$), а вторая - оператором C^m. Таким образом, в моменты t_x и t_y наступления событий $\hat{c}_1^x$ и $\hat{c}_1^y$ в ветвях с именем i состояние памяти D_i совпадает, а в других ветвях они могут различаться только в том случае, если выполнилось разное число операторов в соответствующих фрагментах. Выберем теперь в одноименных ветвях наиболее поздние операторы из тех, которые выполнились в обеих реализациях к моменту наступления события $\hat{c}_1$. Следующие за ними операторы объявим "стартовыми", которые в обеих реализациях начнут свое выполнение над совпадающими "исходными" данными. Повторяя аналогичные рассуждения для $\hat{c}_2, \hat{c}_3$ и т.д. при-

ходим для любых x и y к совпадению конечного состояния памяти $(\bar{D}_h)^f$ каждой ветви h.

Перейдем теперь к рассмотрению п-алгоритмов с операторами из $C(G)$. Будем предполагать, что при всяком использовании оператор $C^m \in C(G)$ имеет, кроме метки, идентифицирующей принадлежность с-оператору, метку, идентифицирующую групповое взаимодействие и являющуюся одной и той же в соответствующих операторах из разных ветвей. Промежуточным событием ветви назовем тройку $\langle t, C^m, M-N \rangle$, где t - момент появления в списке преемников оператора C^m, имеющего метки M и N. Это понятие будем использовать для $C^m \neq$ ОБП. Событием ветви назовем тройку $\langle t, C^m, M-N \rangle$, где t - момент завершения C^m с метками M и N.

Пусть для некоторой реализации x в ветви i имела место последовательность событий (промежуточных событий) $\hat{c}^i_x = \hat{c}^i_1, \hat{c}^i_2, \ldots, \hat{c}^i_{1(i)}$, а в ветви j - $\hat{c}^j_x = \hat{c}^j_1, \hat{c}^j_2, \ldots, \hat{c}^j_{1(j)}$. Последовательности $\hat{c}^i_x$ и $\hat{c}^j_x$ называются совпадающими, если $1(i) = 1(j)$, а элементы с одинаковыми номерами имеют одинаковые операторы $C^m \in C(G)$ и соответствующие им вторые метки. Последовательности $\hat{c}^i_x$ и $\hat{c}^i_y$, имевшие место в ветви i при реализациях x и y, называются совпадающими, если $1^x(i) = 1^y(i)$, а элементы с одинаковыми номерами имеют одинаковые $C^m \in C(G)$ и соответствующие им первые и вторые метки.

ТЕОРЕМА 2. *Для бесступикового выполнения п-алгоритма с операторами из $C(G)$ необходимо и достаточно, чтобы последовательности промежуточных событий во всех ветвях совпадали, а среди ветвей, выполняющих оператор ТО$(1, A, L_1)$, была ветвь с именем 1.*

ДОКАЗАТЕЛЬСТВО. Необходимость следует из определения $C^m \in C(G)$, которое требует, чтобы процесс, выполняющий такой оператор, находился в состоянии ожидания, пока все другие ветви не выйдут на свои операторы, задающие то же взаимодействие. Несовпадение промежуточных событий или отсутствие ветви с именем 1 при выполнении ТО приведут к тому, что список преемников будет состоять только из операторов, условия включения которых не выполняются. Достаточность следует из того, что, с одной стороны, выход процесса на ОБП не задерживает его выполнения, а с другой, в силу

свойств ограниченного п-алгоритма и совпадения событий в ветвях задержка на $C^{\infty} \neq$ ОБП, если и возможна, то только на ограниченное число действий.

Пусть в п-алгоритме, по крайней мере, одна ветвь использует оператор ОБП. Тогда имеет место следующий результат.

ТЕОРЕМА 3. Для однозначности беступикового п-алгоритма с операторами из C(G) достаточно, чтобы для всякого $\bar{D}^0$ ветви, использующие оператор ОБП, при любых реализациях сохраняли последовательности событий, а всякий фрагмент был независим по управлению от ОБП.

ДОКАЗАТЕЛЬСТВО. Из сохранения последовательности событий в ветвях, использующих ОБП, следует, что в этих ветвях сохраняются последовательности промежуточных событий, а кроме того, сохраняются число и имена порождаемых ветвей п-алгоритма. Отсюда с учетом беступиковости п-алгоритма следует, что последовательности промежуточных событий совпадают во всех ветвях и сохраняются при любых реализациях. Независимость фрагментов по управлению от ОБП приводит к сохранению последовательности с-операторов в каждой ветви . Синхронное выполнение фрагментов из разных ветвей, обеспечиваемое операторами из C(G)\ОБП, гарантирует для любых реализаций совпадение конечного состояния памяти одноименных ветвей.

Для однозначности беступикового п-алгоритма с операторами из C(G)\ОБП достаточно, чтобы при любых реализациях сохранялась последовательность событий хотя бы в одной ветви.

Сформулированные в теоремах условия требуют,по существу,такой "разметки" вычислительных процессов в ветвях, при которой каждый оператор фрагмента является независимым от событий, которые происходят в других ветвях во время выполнения фрагмента, а зависимым он может быть только от событий, которые произошли во время выполнения предыдущих фрагментов. Эти условия предполагают, что внешние к п-программам события в системе могут изменять скорость процессов в ветвях, но не могут изменять порядок взаимодействия ветвей. Если необходимо, чтобы внешние события влияли на поря - док, то нужно ввести переменные, отражающие эти события, и в зависимости от значений этих переменных изменять порядок.

Хотя существуют обширные классы задач (например, в линейной алгебре и системах дифференциальных уравнений [2,8]), для которых изменение входных данных не влияет на порядок взаимодействий между ветвями, тем не менее возникает ряд вопросов:

- как следовать сформулированным условиям, если мы хотим иметь в п-алгоритме порядок взаимодействий, изменяемый данными;
- как выбрать более эффективную последовательность событий, если мы не знаем скорости вычислительных процессов;
- в каких ситуациях можно разрешить параллельное выполнение нескольких индивидуальных взаимодействий и не требовать строгого порядка в наступлении соответствующих событий;
- как увеличить гибкость операторов групповых взаимодействий;

В рамках поиска ответов на эти вопросы предлагается:

1. Использовать операторы УП и ОУП для изменения порядка наступления и состава событий в зависимости от $\bar{D}^0$ и результатов вычислений.

2. В каждой подобласти D_i выделить двоичную переменную δ_i, значение которой определяет участие ($\delta_i = 1$) или неучастие ($\delta_i = 0$) ветви i в групповых взаимодействиях; в связи с этим условием включения для операторов ТО, КО, ОУП становится выход каждой ветви с $\delta_i = 1$ на свой оператор, задающий то же самое взаимодействие. Если $\delta_i = 0$, то операторы ТО, КО, ОУП и ОБП воспринимаются ветвью i как пустые. Для изменения значения δ_i использовать операторы $\delta_i := 1$ и $\delta_i := 0$, которые могут быть выполнены только в ветви i; ввести эти операторы, в связи с их особой ролью, в C(G).

3. Ввести расширенное понятие фрагмента ветви как последовательности операторов, которые выполнены между двумя операторами из C и которые в своем составе могут иметь другие $C^m \in C$.

4. Использовать не полный, а частичный порядок в наступлении событий п-алгоритма; каждый $C^m \in C$ сопровождать а) дополнительным условием включения, представляющим собой конъюнктивное и/или дизъюнктивное перечисление событий, которые должны или могут произойти до выполнения C^m, и б) условием завершения, представляющим собой перечисление событий, до наступления которых выполнение C^m должно завершиться.

5. Отказаться от использования семафоров в $C^m \in C(I)$, поскольку их роль в значительной мере дублируется дополнительными условиями включения и завершения.

Предположим, что п-алгоритм представлен с развернутыми циклами, т.е. операторы выбора преемников порождают только ациклические графы, вершинами которых являются с-операторы. Тогда под дополнительным условием включения $C^m \in C(I)$ понимается одно из следующих логических выражений:

$$P_1 = A_{i_1} \wedge A_{i_2} \wedge \ldots \wedge A_{i_m};$$
$$P_2 = (A_{j_1} \vee A_{j_2} \vee \ldots \vee A_{j_n}) \wedge P_1, \qquad (1)$$

где индексы при A - имена ветвей, в любом из выражений нет двух одинаковых индексов; $m \geq 1$, $n \geq 2$, $n + m$ не больше максимального числа ветвей, которые могут быть порождены при выполнении п-алгоритма; $A_{\alpha_\beta} ::= P(R_1, \alpha_\beta) \,|\, P(R_1, \alpha_\beta) \vee P(R_2, \alpha_\beta) \vee \ldots \vee P(R_l, \alpha_\beta)$, причем $P(R_h, \alpha_\beta) = true$ $(h = \overline{1,l})$, если оператор C^m, имеющий метку R_h в ветви α_β, выполнился, иначе $P(R_h, \alpha_\beta) = false$; l - любое целое число.

Под условием завершения понимаются те же выражения P_1 и P_2, только $P(R_h, \alpha_\beta) = true$, если C^m с меткой R_h в ветви α_β не выполнился, иначе $P(R_h, \alpha_\beta) = false$.

Каждому условию включения должно соответствовать условие завершения. Опреатор из C(I) может сопровождаться одной или несколькими парами таких условий. Присутствие нескольких пар означает, что для оператора приемлемы условия любой из этих пар, однако между собой пары являются взаимно исключающими, т.е. в процессе выполнения п-алгоритма условие включения всякий раз может выполняться лишь в одной из этих пар.

Предполагается, что в P_1 как в условиях включения, так и в условиях завершения должны присутствовать метки операторов из ветви, с которой C^m задает взаимодействие (за исключением случаев, когда взаимодействие фиктивное (см.раздел 5)). Последовательность операторов, выполнившихся в этой ветви между оператором из условия включения и оператором из условия завершения, называется обобщенным фрагментом, связанным с C^m. Поскольку обобщенные фрагменты могут пересекаться, то и взаимодействовать с каждым из них могут операторы, связанные с разными обобщенными фрагментами. Обобщенный фрагмент называется информационно независимым от взаимодействующих с ним $C^m \in C(I) \setminus ВЫП$, если операторы фрагмента информационно независимы от каждого из этих C^m, которые между собой либо связаны порядком следования (с помощью условий включения), либо

информационно независимы. Определение независимости по управлению для обобщенных фрагментов будем считать совпадающим с определением для фрагмента.

Для $C^m \in C(G)$ под дополнительным условием включения понимается одно из выражений:

$$P_1 = P(R_1) \vee P(R_2) \vee \ldots \vee P(R_l);$$

$$P_2 = P_1 \wedge \delta(i_1, j_1) \wedge \ldots \wedge \delta(i_n, j_n),$$

где $l \geq 1$, $n \geq 1$, $P(R_h)$ = true , если взаимодействие задаваемое операторами $C^m \in C(G)$, имеющими вторую метку R_h, выполнилось, иначе $P(R_h)$ = false ; $\delta(i_\beta, j_\beta)$ = true, если в ветви i_β оператор $\delta_{i_\beta} := 0$ (или $\delta_{i_\beta} := 1$), имеющий метку j_β, выполнился, иначе $\delta(i_\beta, j_\beta)$ = false; $\{i_\beta,\ \beta = \overline{1,n}\}$ - множество имен ветвей, которые должны переопределить свое состояние по участию в групповых взаимодействиях до выполнения $C^m \in C(G)$. Под условием завершения понимается одно из выражений P_1 или P_2, только $P(R_h)$ = true, $\delta(i_\beta, j_\beta)$ = true, если соответствующие действия не произошли. Отметим, что описанные условия включения и завершения должны сопровождать не только операторы из $C(G)$, описанные в разделе 3, но и операторы $\delta_i := 1$ и $\delta_i := 0$; причем наличие оператора для δ_i в одном из путей вычислений ветви требует присутствия оператора для δ_i во всех альтернативных путях.

Пусть п-алгоритм представлен с развернутыми циклами. Тогда можно построить граф взаимодействия ветвей, в котором вершины соответствуют операторам из C, а дуги - связям, описанным в дополнительных условиях включения; причем в вершину, соответствующую C^m, входят дуги из всех вершин, соответствующих операторам, присутствующим в этих условиях. Будем предполагать, что в графе взаимодействий либо есть путь к вершине, оператор которой использует имя некоторой ветви, от вершины, оператор которой задает порождение этой ветви, либо ветвь относится к стартовому подмножеству. Будем также предполагать, что всегда есть путь от вершины, оператор которой использует имя ветви, к вершине, оператор которой уничтожает эту ветвь.

ТЕОРЕМА 4. Для того, чтобы п-алгоритм с операторами из $C(I)$ был бесступиковым, достаточно, чтобы граф взаимодействия ветвей был без петель и контуров.

ТЕОРЕМА 5. Для того, чтобы бесступиковый п-алгоритм с операторами из C(I) был однозначным, достаточно, чтобы для всякого $\bar{D}^0$ и любой реализации каждый обобщенный фрагмент был информационно и по управлению независим от взаимодействующих с ним C^m.

Аналогичным образом можно сформулировать новые условия для п-алгоритмов с $C^m \in C(G)$.

5. Языки для записи п-алгоритмов. Описываются языковые средства PA(I) и PA(G) для записи п-алгоритмов, использующих соответственно операторы из C(I) и C(G). Для простоты эти средства предполагают, что п-алгоритм есть конечное множество ветвей, для каждой из которых должна быть написана "последовательная" программа, хотя при выполнении п-алгоритма не каждая ветвь может быть инициирована. В PA(I) и PA(G) входят традиционные структуры последовательных языков: линейная, if-then-else и while-do , операторы присваивания и описания, ввода-вывода, множество операторов C(I) или C(G), а также согласованные и обобщенные while-do структуры. Каждая ветвь имеет свое описание переменных; переменные, а также метки, принадлежащие различным ветвям, могут иметь одинаковые имена. Каждый оператор из C должен иметь свою метку в ветви, а оператор из C(G), кроме того, - вторую метку, являющуюся, с одной стороны, признаком конкретного группового взаимодействия (операторы из разных ветвей, задающие некоторое групповое взаимодействие должны иметь одинаковую вторую метку, соответствующую этому взаимодействию), а с другой стороны - признаком, выделяющим в тексте отдельной ветви все операторы из C(G), предназначенные для участия ветви в таком взаимодействии, но взаимно исключающие друг друга по логике исполнения программы.

Язык PA(I). В C(I) входят операторы ПРО(a,i,a_1), ЗАП(a, i, a_1), ВЫП(S_h, i). Отличаются они от описанных в разделе 3 тем, что исключены аргументы, определяющие использование семафоров, но зато каждый оператор должен сопровождаться дополнительными условиями включения и завершения в виде логических выражений, описанных в разделе 4. Условия включения и завершения будем помещать на месте комментариев; причем сначала условие включения, а после точки с запятой - условие завершения. Вместо $P(R_1, \alpha_\beta)$ будем писать

(R_1, α_β), вместо $P(R_h) - (R_h)$, вместо $\delta(i_1,j_1) - (i_1,j_1)$, а на события, отражающие запуск и финиш ветви, будем указывать буквами "з" и "ф"; например, запись $(з, \alpha_\beta)$; $(ф, \alpha_\beta)$ означает условия включения и завершения, охватывающие все время существования процесса α_β. Уточнение описаний условий включения и завершения будет дано ниже. Предполагается, что каждый оператор из C(I) может быть без аргументов; тогда он является пустым оператором и используется с помощью условий включения и завершения лишь для синхронизации процессов. Особенность условий включения и завершения для пустого оператора в том, что они могут быть тождественно истинными (отсутствующими) либо иметь тождественно истинным значение P_1 в выражении P_2 из (I).

Применение операторов из C(I) в условных структурах. Пусть сегмент $\hat{s}_1$ программы задает одну последовательность событий в ветви, а $\hat{s}_2$ - другую. Тогда допустима конструкция if P then $\hat{s}_1$ else $\hat{s}_2$, которая в зависимости от данных изменяет последовательность наступления событий в ветви. При этом дополнительные условия включения и завершения любого C^m из $\hat{s}_1$ не должны ссылаться на операторы из $\hat{s}_2$, и наоборот. Операторы других ветвей, зависящие от событий такой конструкции, должны использовать дизъюнктивную форму для A_{α_β} из (I); каждый сегмент должен иметь, по крайней мере, один $C^m \in C(I)$, которым может быть пустой оператор. Эта же конструкция допустима для изменения последовательности наступления событий из двух ветвей. Например:

Ветвь v_1	Ветвь v_2
if P then	
M_0: ЗАП	K_1: ЗАП(a_2, v_1, f_1) | $(M_0, v_1) \vee (M_2, v_1)$; $(ф, v_1)$
. . .	
M_1: ЗАП(a_1, v_2, f_2) | (K_1, v_2); $(ф, v_2)$	
else	
M_2: ЗАП(a_1, v_2, f_2) | $(з, v_2)$; (K_1, v_2)	

Если P = true, то оператор ЗАП(a_1, v_2, f_2) выполнится после оператора с меткой K_1, иначе наоборот.

Использование операторов из C(I) в while-do цикле допускается только в двух случаях. В первом случае предполагается, что

условия включения и завершения операторов, присутствующих в теле цикла, остаются неизменными во время выполнения всех итераций, а сами операторы не присутствуют в условиях включения и завершения C^m из других ветвей. Неизменность условий включения и завершения достигается ссылками на операторы, не содержащиеся в других циклах. Поэтому после выполнения одной итерации рассматриваемого цикла условия включения выполнившихся операторов из C(I) будут истинными для всех последующих итераций. Для ссылки на последнее из событий в таком цикле, число повторений которого неизвестно, предполагается использование дополнительного оператора из C(I) непосредственно после цикла. В цикл могут быть вложены другие циклы, но указанные предположения должны выполняться для самого внешнего цикла. Во втором случае использование операторов из C(I) предполагается в теле <u>согласованного</u> цикла, т.е. цикла, входящего в группу циклов (все из разных ветвей), в которых условия включения и завершения могут зависеть от выполнения операторов на разных итерациях и из разных циклов этой группы. Для указания зависимости от такого оператора нужно написать метку оператора, имя ветви, метку цикла и номер итерации вида $j \pm k$, где k – константа, а j – номер итерации, на которой исполняется зависимый оператор. Зависимость исчезает, если $j + k$ становится больше числа итераций, выполненных в соответствующем завершившемся цикле, или если $j - k \leq 0$. Все ссылки в теле цикла на операторы одной и той же ветви должны иметь одинаковые значения k. Следующий пример демонстрирует использование согласованных циклов для описания асинхронного конвейера, состоящего из n базовых процессов и n буферных процессов (имеющих дробные номера):

Ветвь $v_i (1 < i < n)$

Li: while P_i do

M_1: ПРО$\left(A_{i-\frac{1}{2}}, v_{i-\frac{1}{2}}, B_i\right)$ | $(N_{i-1}, v_{i-1}, L_{i-1}, j) \wedge \left(з, v_{i-\frac{1}{2}}\right)$;

. . . | $(N_{i-1}, v_{i-1}, L_{i-1}, j+1) \wedge \left(ф, v_{i-\frac{1}{2}}\right)$;

. . . . . .

M_2: ЗАП$\left(Z_i, v_{i+\frac{1}{2}}, A_{i+\frac{1}{2}}\right)$ | $(M_{i+1}, v_{i+1}, L_{i+1}, j-1) \wedge \left(з, v_{i+\frac{1}{2}}\right)$;

. . . | $(M_{i+1}, v_{i+1}, L_{i+1}, j) \wedge \left(ф, v_{i+\frac{1}{2}}\right)$;

Конец L_i

Здесь ветвь v_i на каждой итерации цикла L_i сначала выполняет оператор ПРО, т.е. переменной B_i присваивает значение $A_{i-\frac{1}{2}}$ из ветви $v_{i-\frac{1}{2}}$, а затем оператор ЗАП, т.е. переменной $A_{i+\frac{1}{2}}$ в ветви $v_{i+\frac{1}{2}}$ присваивает значение Z_i (вычисленное в v_i). Условием включения оператора ПРО является конъюнкция двух условий: I) выполнения ветвью v_{i-1} оператора ЗАП на j-й итерации цикла L_{i-1} и 2) запуска ветви $v_{i-\frac{1}{2}}$. Условием включения оператора ПРО является также конъюнкция двух условий: I) выполнения ветвью v_{i+1} оператора ПРО на (j-1)-й итерации цикла L_{i+1}, 2) запуска ветви $v_{i+\frac{1}{2}}$.

При выполнении первой итерации j-1=0 и первое условие считается тождественно истинным. Условием завершения операторов ПРО и ЗАП является также конъюнкция двух условий. В частности, оператор ПРО должен выполниться до того, как в ветви v_{i-1} выполнится оператор ЗАП на (j+1)-й итерации цикла L_{i-1}, а также до того, как ветвь $v_{i-\frac{1}{2}}$ завершится. Выполнение группы согласованных циклов в общем случае можно разбить на три фазы: начальную, основную и конечную. Начальная фаза характеризуется тем, что одни ветви выполняют свои согласованные циклы, а другие - либо еще не приступили к их выполнению, либо приступили, но остановились в соответствии с условиями включения и завершения, т.е. в ожидании, когда та или иная ветвь выполнит определенное число итераций своего согласованного цикла.

Во время основной фазы все ветви, имеющие согласованные циклы рассматриваемой группы, выполняют их; фаза завершается, как только завершится какой-либо из этих циклов. После чего наступает конечная фаза. Она финиширует в момент завершения всех согласованных циклов группы. Каждый цикл может быть согласованным только в рамках одной группы. Ссылки между операторами из C(I), находящимися в циклах разных ветвей, разрешаются только для согласованных циклов. Предполагается, по умолчанию, что все взаимодействия между ветвями, задаваемые согласованными циклами, должны в каждой ветви (содержащей такой цикл) начинаться после выполнения последнего перед циклом и заканчиваться до выполнения второго после цикла оператора из C(I).

Согласованные циклы могут иметь вложенные циклы, но условия наступления событий должны включать связи только между операторами, входящими в циклы одинаковой вложенности. Номер итерации в этом случае есть вектор: $(j_1 \pm k_1, j_2 \pm k_2, \cdots, j_n \pm k_n)$, где k_i $(i = \overline{1,n})$ - константа, n - число циклов, которым принадлежит соответствующий оператор. Зависимость от оператора исчезает, если хотя бы один компонент становится больше числа итераций в завершившемся цикле или ≤ 0.

Завершим описание PA(I) правилами использования оператора ВЫП(S_j, i). Они требуют применения этого оператора только для запуска или уничтожения процесса в ветви i, т.е. когда S_j есть первый оператор или оператор Ø, а также, чтобы ВЫП(S_j, i) из ветви n и ВЫП(S_k,1) из ветви h были связаны строгим порядком следования, если имеет место хотя бы одно из равенств: 1 = i, 1 = n, i = h.

Язык PA(G). В C(G) входят операторы ТО(l,A,L_j), КО(T,L_j,M_j), ОУП(b,F_j,S_j^1,S_j^2) и ОБП(List), описанные в разделе 3, а также операторы δ_j:=1 и δ_j:=0. После каждого оператора должны быть представлены дополнительные условия включения и завершения, описанные в разделе 4. Предполагается, что ветвь начинает выполнение своего стартового оператора при $\delta_j = 1$. Для выполнения ТО(l,A,L_j) ветвь l должна иметь $\delta_l = 1$.

Использование операторов из C(G) в условных структурах допускается в следующих случаях. Пусть сегмент $\hat{s}_1$ программы задает одну последовательность событий в ветви, а $\hat{s}_2$ - другую. Тогда выражение if P then $\hat{s}_1$ else $\hat{s}_2$ допустимо, если последовательности можно разбить на одинаковое число отрезков, таких, что подпоследовательности в отрезках с одинаковыми номерами либо совпадают, либо хотя бы одна из двух несовпадающих подпоследовательностей является пассивной, т.е. начинается с оператора δ_j:=0, а кончается - δ_j:=1.

В следующем примере, если P = true, то ветвь v_i выполнит операторы M_1-K_1 и M_2-K_2 ; если P ≠ true , то после оператора M_3 - K_1, на время выполнения взаимодействия K_2, ветвь v_i станет пассивной к групповым взаимодействиям:

Ветвь v_i	Ветвь v_j
if P then	
M_1-K_1: ТО(v_i,B_i,L_i) | (s);(K_2)	M_1-K_1: ТО(v_i,B_i,L_j) | (s);(K_2)

```
. . .
M2-K2:TO(vj,Bj,Lj) | (K1);(ф) | M2-K2:TO(vj,Bj,Lj) | (K1)∧(M4,v1);
                                                    | (M5,v1)
. . .                                               . . .
  else
M3-K1: TO(v1,B1,L1) | (з); (M4)
. . .
M4: δ1:=0           | (K1); (K2)
. . .
M5: δ1:= 1          | (K2); (ф)
```

Условная конструкция может быть вновь использована в $\hat{s}_1$ и $\hat{s}_2$, но указанное правило тогда распространяется на всю совокупность возможных последовательностей, т.е. все они должны допускать разбиение на одинаковое число непересекающихся отрезков, таких, что подпоследовательности для ветвей с $\delta_1 = 1$ совпадают во всех отрезках с одинаковыми номерами. Отметим, что пассивный отрезок может при необходимости трактоваться как последовательность пассивных отрезков.

Для изменения последовательности событий п-алгоритма может быть использован и оператор ОУП. Пусть $\hat{s}^1_j$ для всех j с $\delta_j = 1$ задают одну последовательность операторов из $C(G)$, а $\hat{s}^2_j$ - другую. Тогда в каждой активной ветви допустим оператор ОУП(b,F_j,$\hat{s}^1_j$, $\hat{s}^2_j$). В $\hat{s}^1_j$ и $\hat{s}^2_j$ ОУП может быть применен вновь.

Использование $C^m \in C(G)$ в циклических структурах возможно только в обобщенном while-do цикле, т.е. цикле, входящем в совокупность циклов (все из разных ветвей с $\delta_j = 1$), условием повторения каждого из которых является ОУП(b,F_j,S^1_j, S^2_j), где S^1_j - первый оператор в теле цикла, S^2_j - первый оператор после цикла. Ветвь с $\delta_j = 0$ не должна выполнять оператор $\delta_j = 1$ во время выполнения такого цикла в других ветвях. Обобщенный цикл и while-do цикл, не содержащий $C^m \in C(G)$, могут быть вложены в другой обобщенный цикл. Следующий пример иллюстрирует использование обобщенного цикла при решении системы линейных уравнений $X = f(X)-Z$ методом последовательных приближений;

$$F_1 = \begin{pmatrix} X_1^{(k)} \\ X_1^{(k+1)} \end{pmatrix}, \qquad F_2 = \begin{pmatrix} X_2^{(k)} \\ X_2^{(k+1)} \end{pmatrix},$$

$$b = (|X_1^{(k+1)} - X_1^{(k)}| < \varepsilon) \wedge (|X_2^{(k+1)} - X_2^{(k)}| < \varepsilon),$$

$$X^{(k)} = \begin{pmatrix} X_1^{(k)} \\ X_2^{(k)} \end{pmatrix}, \quad X^{(k+1)} = \begin{pmatrix} X_1^{(k+1)} \\ X_2^{(k+1)} \end{pmatrix}, \quad Z = \begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix}, f = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix};$$

условия включения и завершения, а также первые метки для C^m опущены ввиду очевидности:

Ветвь v_1	Ветвь v_2
M_0: while-ОУП(b, F_1, M_1, M_2)	M_0: while-ОУП(b, F_2, M_1, M_2)
M_1: $X_1^{(k+1)} := f_1(X^{(k)}) - Z_1$	M_1: $X_2^{(k+1)} := f_2(X^{(k)}) - Z_2$
M_2: TO(v_1, $X_1^{(k+1)}$, $X_1^{(k)}$)	M_2: TO(v_1, $X_1^{(k+1)}$, $X_1^{(k)}$)
M_3: TO(v_1, $X_2^{(k+1)}$, $X_2^{(k)}$)	M_3: TO(v_2, $X_2^{(k+1)}$, $X_2^{(k)}$)
Конец M_0	Конец M_0
M_4: . . .	M_4: . . .

Оператор ОБП предназначен для запуска или уничтожения процессов. В условиях его выполнения должны быть указаны операторы из C(G), после и до которых он будет реализован. Операторы ОБП должны быть связаны строгим порядком следования, если подмножество взаимодействующих ветвей при выполнении одного ОБП пересекается с подмножеством взаимодействующих ветвей при выполнении другого.

Фрагмент ветви, связанный с ОБП, уничтожающим процесс в этой ветви, должен быть пустым. Аналогичное правило действует и для оператора ВЫП из C(I).

Теперь можно сформулировать заключительный результат.

ТЕОРЕМА 6. Пусть п-алгоритм, записанный на РА(I) или РА(G), имеет циклы, которые завершаются за конечное число итераций. Тогда бесступиковость и однозначность п-алгоритма установима по тексту п-программ.

Вместо доказательства отметим лишь несколько аспектов, которые имеют к нему отношение. Для определения бесступиковости п-алгоритма нужно построить граф взаимодействия ветвей. Основная пре-

града при этом - циклические структуры, однако здесь достаточно анализа тела отдельного while-do цикла в PA(I), совокупности тел обобщенных циклов в PA(G), а также тел согласованных циклов в PA(I). Причем в последнем случае при анализе используется представление отдельного согласованного цикла в виде последовательности нескольких циклов, имеющих одно и то же тело, но выполняющих разные итерации исходного цикла. Это позволяет выделить итерации, выполняющиеся соответственно во время начальной, основной и конечной фаз, и практически избежать полной развертки анализируемых согласованных циклов. Обнаружение контура в графе означает наличие тупика в п-алгоритме только в том случае, если среди вершин контура нет таких, которые в отдельной ветви принадлежат взаимноисключающим путям вычислительного процесса. При наличии указанных вершин выявление тупиков продолжается путем поочередного исключения из контура альтернативных вершин. Если полагать, что в каждой ветви все пути вычислений допустимы, т.е. для каждого пути существуют исходные данные, при которых этот путь реализуется, то упомянутый анализ графа является необходимым и достаточным для выявления бесступиковости п-алгоритма. Для определения однозначности п-алгоритма нужно выяснить информационную независимость фрагментов, поскольку независимость по управлению задается явно.Здесь существенную помощь оказывает разметка ветвей п-алгоритма, задаваемая условиями включения и завершения, а также тот факт, что операторы, задающие обмены, используют в качестве аргументов имена статически определенных переменных, а не их отдельных компонентов, определяемых динамически вычисляемыми значениями индексных выражений.

6. Заключение. Разработан подход к организации вычислений, при которых каждый параллельный процесс реализуется как последовательность подпроцессов,не зависящих информационно и по управлению от взаимодействий, происходящих с другими процессами во время существования подпроцесса. Выбраны языковые средства и правила их использования, поддерживающие развитый подход. Сформулированы условия, обеспечивающие бесступиковость и однозначность параллельных алгоритмов. Эти условия могут использоваться как основа для установления корректности параллельных программ по их тексту.

Анализ большого количества задач и стратегия крупноблочного распараллеливания [2,8] позволяет надеяться, что данный подход для широкого класса применений (в первую очередь для задач вычис-

лительной математики и физики) позволит создавать не только корректные, но и эффективные параллельные программы. Этот подход, однако, может приводить к некоторой потере эффективности для задач, сущность которых подразумевает коммутативность произвольного порядка выполнения информационно зависимых процедур (вопрос о том, как выявлять по тексту эту коммутативность, является предметом особого рассмотрения). Наконец, следует отметить, что полученные здесь результаты не всегда применимы при решении задач, для которых понятие однозначности п-алгоритма не является адекватной характеристикой его корректности. Для таких задач допустимым является множество различных конечных состояний памяти, в значительной степени связанных со скоростью протекания процессов.

Литература

I. ЕРШОВ А.П. Операторные алгоритмы.I. - В кн.: Проблемы кибернетики, вып.3. М., 1960, с.5-48.

2. ЕВРЕИНОВ Э.В., КОСАРЕВ Ю.Г. Однородные универсальные вычислительные системы высокой производительности.- Новосибирск: Наука, 1966. - 308 с.

3. КОТОВ В.Е. Теория параллельного программирования. Прикладные аспекты. - Кибернетика, 1974, № I, с.3-17.

4. Автоматизация параллельного программирования на основе существующих трансляторов /Головяшкина Л.В., Колосова Ю.И., Косарев Ю.Г., Миренков Н.Н. - В кн.: Вычислительные системы. Вып.30. Новосибирск, 1968, с.63-69.

5. HABERMAN A.N. Prevention of System Deadlocks.- CACM, 1969, v.12, N 7, p.373-377.

6. КОСАРЕВ Ю.Г. О схемах обмена между ветвями параллельных алгоритмов. - В кн.: Вычислительные системы. Вып.52. Новосибирск, 1972, с.70-76.

7. Язык для записи параллельных алгоритмов /Гришаева Н.К., Кербель В.Г., Колосова Ю.И. и др. - В кн.: Вычислительные системы. Вып.57. Новосибирск, 1973, с.33-64.

8. МИРЕНКОВ Н.Н. Параллельные алгоритмы для решения задач на однородных вычислительных системах.- Там же, с.3-32.

9. LIPTON R., SNYDER L., ZALCSTEIN Y. Comparative Study of Models of Parallel Computation.- In: 15th Annual Symposium on Switching and Automata Theory, 1974, p.145-155.

I0. HOWARD B.V. Parallel computation schemata and their hardware implementation.- Digital processes, 1975, v.1, N 3.

II. OWICKI S., GRIES D. An axiomatic proof thechnique for parallel programs.I. - Acta informatica, 1976, v.6, p.319-340.

I2. ROSEN Barry K. Correctness of Papallel Programs: The Church-Rosser Approach.- Theoretical Computer Science, 1976, v. 2, p.183-207.

I3. КОРАБЛИН Ю.П. Проблема корректности граф-схемы параллельных алгоритмов. - Программирование, 1978,№ 5,с.45-52.

I4. Программное обеспечение системы МИНИМАКС / Кербель В.Г., Колосова Ю.И., Корнеев В.Д. и др. - Новосибирск. Б.и.,1979.-43 с. (Препринт/Институт математики СО АН СССР: ОВС-09).

I5. **Semantics of Nondeterminism, Concurrancy, and Communication/ Nissim Francess, C.A.R.Hoare, Daniel J.Lehmann, Willem P.de Poever.-** Journal of Computing and System Sciences, 1979, v.19, p.290-306.

I6. **BOS J.,van den, PLASMEIJER R., STROET J.-** Process Communication Based on Input Specifications. - **ACM Tran's** on Programming Languages and Systems,1981, v.3,N 3,p.224-250.

I7. АНИШЕВ П.А. Один способ анализа корректности граф-схем алгоритмов.- Программирование,1981,№ I,с.20-28.

I8. Алгоритмы, математическое обеспечение и архитектура многопроцессорных систем.- М.:Наука,1982.-336 с.

I9. КУТЕПОВ В.П., ФАЛЬК В.Н. Функциональные системы. Теоретический и практический аспекты.- Кибернетика,1979,№ I,с.46-58.

20. **MISRA J., CHANDY K.M. Termination Detection of Diffusing Computations in Communicating Sequential Processes. - ACM Tran's on Programming Languages and Systems,** 1982,v.4, N 1,p.37-43.

2I. КРИНИЦКИЙ Н.А. Параллельные алгоритмы. - Программирование,1983,№ 4,с.9-I5.

Поступила в ред.-изд.отд.
I7 июля I985 года