УДК 004.432.4 **DOI: 10.14529/cmse220201**

РЕСУРСОНЕЗАВИСИМОЕ ОПИСАНИЕ ИНФОРМАЦИОННЫХ ГРАФОВ С ДИСТРИБУТИВНЫМИ ОПЕРАЦИЯМИ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ SET@L

© 2022 И.И. Левин[1], И.В. Писаренко[2], Д.В. Михайлов[2], А.К. Мельников[3], А.И. Дордопуло[2]

[1]*Южный федеральный университет (347928 Таганрог, пер. Некрасовский, д. 44),*
[2]*«НИЦ супер-ЭВМ и нейрокомпьютеров» (347922 Таганрог, пер. Итальянский, д. 106),*
[3]*АО «Вычислительные решения» (117587 Москва, Варшавское ш., д. 125с17)*
E-mail: iilevin@sfedu.ru, pisarenko@superevm.ru, mixailow.den@gmail.com, ak@comp-sol.ru, dordopulo@superevm.ru
Поступила в редакцию: 05.05.2022

В данной работе предлагается преобразовать стандартную последовательную топологию информационного графа с дистрибутивными операциями к комбинированному варианту с последовательными и параллельными фрагментами, что позволяет эффективно описать реализацию вычислений в ресурсонезависимой форме. Конечная топология зависит от доступного вычислительного ресурса реконфигурируемой системы и обеспечивает повышение удельной производительности в сравнении с исходным вариантом. Разработанный алгоритм преобразования линейной структуры в различные комбинированные топологии в зависимости от конфигурации вычислительной системы описан на языке программирования Set@l.

Ключевые слова: дистрибутивные операции, ресурсонезависимое программирование, реконфигурируемые вычислительные системы, редукция производительности, Set@l, признаки «разбиение пополам» и «голова/хвост».

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Левин И.И., Писаренко И.В., Михайлов Д.В., Мельников А.К., Дордопуло А.И. Ресурсонезависимое описание информационных графов с дистрибутивными операциями на языке программирования Set@l // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2022. Т. 11, № 2. С. 5–17. DOI: 10.14529/cmse220201.

Введение

Информационные графы многих прикладных задач имеют последовательную (линейную) топологию, поэтому реализация их распараллеливания (ускорения) в общем виде затруднительна. Однако, если операции графа обладают определенными свойствами, такими как дистрибутивность и ассоциативность, то целесообразно преобразовать их к виду, для которого допустимо параллельное (ускоренное) выполнение. Например, в работе [1] показано, что за счет перестановки вершин в информационном графе с ассоциативными операциями и дальнейшей оптимизации вычислительной структуры время решения задачи может быть сокращено в число раз, соответствующее латентности операционной вершины. В данной статье представлен другой пример оптимизации последовательного информационного графа с дистрибутивными операциями для реализации на реконфигурируемых ВС с различными конфигурациями. Подобные графы широко применяются для решения многих задач цифровой обработки сигналов [2, 3]. Для практического применения предложенных преобразований необходимы эффективные методы и средства представления параллельных программ в ресурсонезависимом виде. В работах [4–6] предложен язык архитектурно-

независимого параллельного программирования Set@l (Set Aspect-Oriented Language), основанный на парадигме аспектно-ориентированного программирования (АОП) и теоретико-множественном представлении исходного кода программы. Описав базовую топологию графа с дистрибутивными операциями и принципы ее преобразования в виде специальных признаков метода обработки на языке Set@l, возможно синтезировать множество вариантов топологий и переходить между ними путем изменения типов и разбиений совокупностей. Приведение графа к оптимальному виду при заданном объеме вычислительного ресурса реконфигурируемой ВС может осуществляться без изменения исходного кода программы.

1. Преобразование информационных графов с дистрибутивными операциями

Рассмотрим последовательный информационный граф, топология которого изображена на рис. 1. Представленный граф состоит из чередующихся операционных вершин α и β, причем обе базовые операции обладают свойством ассоциативности, а операция β является дистрибутивной по отношению к операции α. Граф на рис. 1 позволяет вычислить последовательность коэффициентов $W_1, W_2, \dots W_n$, определяемых следующей рекуррентной формулой:

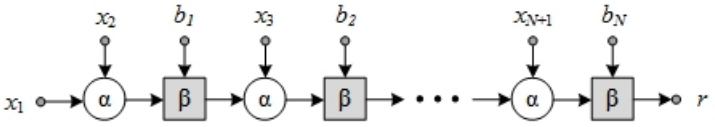


Рис. 1. Последовательный информационный граф с дистрибутивными операциями α и β

Граф на рис. 1 позволяет вычислить последовательность коэффициентов $W_1, W_2, \dots W_n$, определяемых следующей рекуррентной формулой:

$$W_0 = x_1; \quad W_i = (W_{i-1}\alpha\chi_{i+1})\beta b_i, \quad i = 1, 2, \dots, N, \tag{1}$$

где N — мощность множества, включающего все коэффициенты β_i. Время выполнения всех операций графа составляет

$$T_G = \tau \cdot N \cdot (l_\alpha + l_\beta), \tag{2}$$

где τ — длительность такта в секундах; $l_{\alpha,}$, l_β — латентности операционных вершин α и β в тактах.

Предположим, что объем доступного вычислительного ресурса R реконфигурируемой ВС не ограничен. Тогда для сокращения времени решения задачи T_G возможно модифицировать последовательную топологию на рис. 1, используя свойства дистрибутивности и ассоциативности операций α и β. Введем элементарные преобразования пар вершин, показанные на рис. 2, а и б. Преобразование на рис. 2, а перегруппирует пару последовательно соединенных вершин α и β в соответствии с законом дистрибутивности:

$$(\alpha(a, b, i), \beta(i, c, r)) = (\beta(a, c, i), \beta(b, c, ii), \alpha(i, ii, r)), \tag{3}$$

где a, b, c — входные вершины; r — выходная вершина; i, ii — дуги, соответствующие промежуточным результатам вычислений. Преобразование на рис. 2, б представляет со-

бой слияние двух ассоциативных вершин β в одну вершину β, на вход которой подается множество исходных операндов:

$$(\beta(a,b,i),\alpha(i,c,r)) = \beta(a,\{b,c\},r). \tag{4}$$

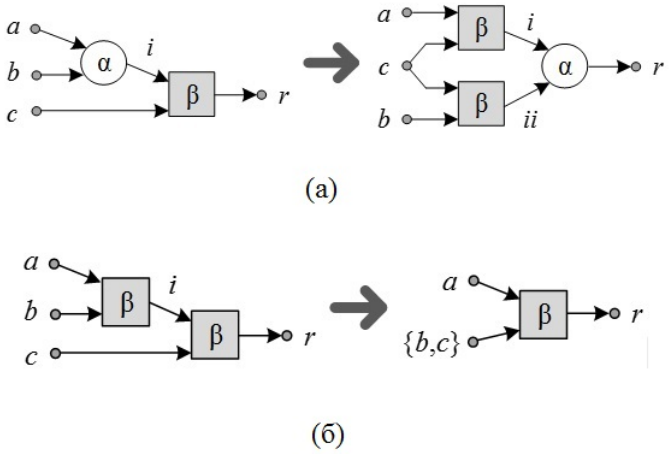


Рис. 2. Преобразование пары дистрибутивных вершин α и β (а) и слияние пары ассоциативных вершин (б), показанное на примере операции β

Последовательно применяя преобразования на рис. 2, а и б к исходному линейному графу (см. рис. 1), возможно получить модифицированную топологию с множеством подграфов-ветвей, одна из которых состоит только из вершин α, а остальные – только из вершин β, как показано на рис. 3, а.

Теперь, когда граф разбит на фрагменты, содержащие ассоциативные операционные вершины только одного типа α или β (см. рис. 3, а), возможно распараллелить вычисления, преобразовав каждый из последовательных подграфов к пирамидальному виду [7]. В результате будет получена топология, изображенная на рис. 3, б и включающая нижний пирамидальный подграф из вершин α, входы которого соединены с выходами множества верхних пирамидальных подграфов, составленных из вершин β. Размер пирамид из вершин β будет меняться от N вершин (для двух крайних левых подграфов на рис. 3, б) до одной вершины (крайне правая пирамида на рис. 3, б).

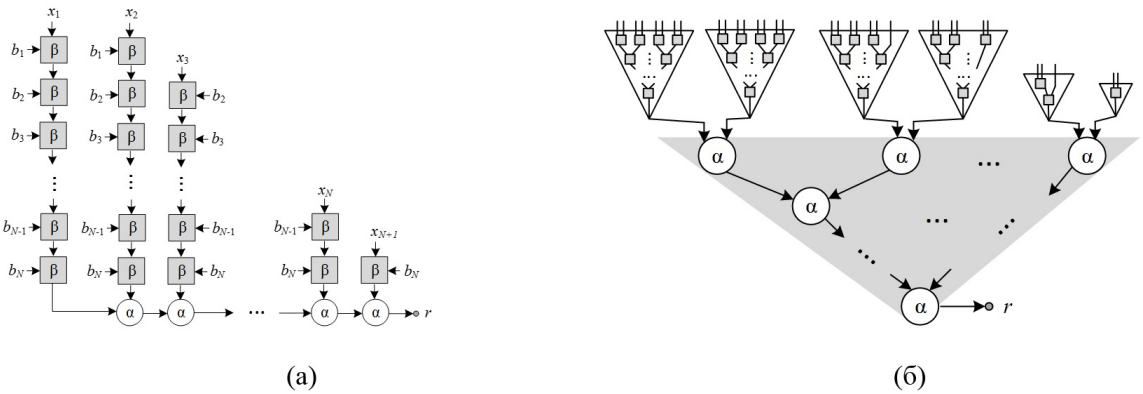


Рис. 3. Модифицированные топологии информационного графа с дистрибутивными операциями α и β с последовательными (а) и параллельными (б) подграфами-ветвями

Время выполнения всех операций информационного графа на рис. 3, б является минимальным и составляет

$$T_G = (l_\alpha + l_\beta) \cdot \lceil \log_2 N \rceil \cdot \tau, \tag{5}$$

где $\lceil \quad \rceil$ — округление до целого в большую сторону. В формуле (5) округление множителя $\log_2 N$ учитывает случаи, когда размерность пирамидальных подграфов необходимо дополнить до степени 2. В отличие от исходной топологии (см. рис. 1) информационные графы на рис. 3 характеризуются существенно большим объемом аппаратного ресурса, занимаемого соответствующими им вычислительными структурами: количество вершин β возрастает до $(N^2 + N)/2$, тогда как число вершин α остается равным N. Например, если в исходном графе была 1000 вершин β, то после преобразования их количество возрастет более чем в 500 раз.

На практике при решении задач даже небольшой размерности может возникнуть ситуация, когда объем аппаратного ресурса реконфигурируемой ВС недостаточен для реализации топологий, изображенных на рис. 3. Анализируя информационный граф на рис. 3, а, выделим следующие особенности:

- каждая ветвь из вершин β использует только одну из переменных x_i;
- на две левые ветви из вершин β подаются все коэффициенты $\{b_1 \dots b_N\}$, а на каждую последующую ветвь — на один коэффициент меньше, то есть m-я ветвь оперирует коэффициентами $\{b_{m-1} \dots b_N\}$.

Поскольку операция β ассоциативна, перегруппируем вершины в ветвях таким образом, чтобы блоки с входами $\{x_1 \dots x_{N+1}\}$ располагались вплотную к подграфу из вершин α (см. рис. 4, а). В результате такого преобразования каждая ветвь, кроме крайней правой, будет содержать вершину β с входами b_{N-1} и b_N (выделены на рис. 4, а). Оставим одну вершину $\beta(b_{N-1}, b_N, i)$ в крайней левой ветви и удалим повторяющиеся вершины, подав на входы остальных ветвей данные с выхода i (см. рис. 4, б). В полученном графе возникнет новое множество идентичных вершин $\beta(i, b_{N-2}, ii)$, которое выделено на рис. 4, б и может быть заменено одной вершиной аналогичным образом. Каждый шаг преобразования уменьшает количество вершин β и формирует промежуточную топологию, реализуемую на ВС с определенным объемом вычислительного ресурса. В предельном случае, показанном на рис. 4, в, процедура удаления одинаковых вершин повторялась до тех пор, пока во всех ветвях, кроме крайней левой, не осталось по одной вершине β. Количество вершин β в такой топологии равно $2 * N - 1$ вместо $(N^2 + N)/2$ в исходном графе (см. рис. 4, а). Линейку вершин α в графе на рис. 4 в целесообразно преобразовать в пирамиду (см. рис. 4, г), обеспечив снижение времени выполнения всех операций до

$$T_G = (l_\alpha \cdot \lceil \log_2 N \rceil + l_\beta \cdot N) \cdot \tau. \tag{6}$$

Дальнейшее уменьшение аппаратных затрат возможно при разбиении исходного графа (см. рис. 1) на линейные изоморфные подграфы, содержащие максимальное реализуемое число вершин, однако при таком подходе время решения задачи T_G будет не оптимальным. Более эффективный подход предполагает преобразование каждого подграфа к виду, аналогичному рис. 4, г.

Если количество операционных вершин в графе с дистрибутивными операциями после модификаций, представленных на рис. 3 и 5, все еще велико для реализации соответствующей вычислительной структуры на имеющемся аппаратном ресурсе, то для формирования плотного потока данных необходимо выполнить следующие преобразования. Вернемся к по-

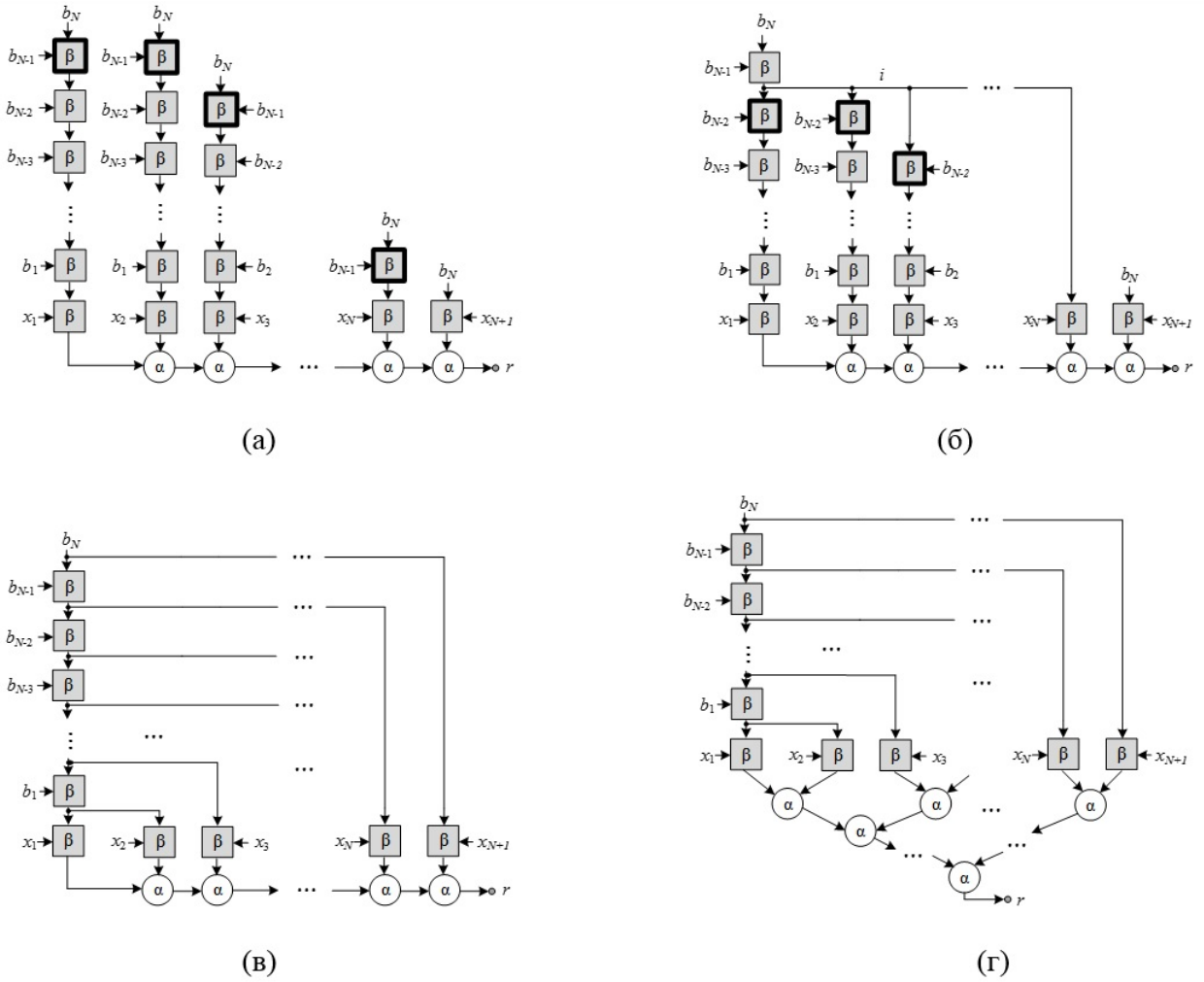


Рис. 4. Модификация информационного графа с дистрибутивными операциями: перестановка вершин β в ветвях (а), однократное удаление повторяющихся вершин β (б), граф с одной полной ветвью (в) и перегруппировка вершин α в пирамидальную структуру (г)

следовательной топологии (см. рис. 1) и разобьем исходный граф на изоморфные подграфы, содержащие по L линейно соединенных пар вершин α и β. Далее, используя предложенную выше методику, приведем каждый выделенный подграф к виду «полная линейка β — пирамида α», показанному на рис. 5, а. В соответствующей полученному графу вычислительной структуре обратная связь будет охватывать полную линейку операций β и пирамиду операций α, поэтому интервал обработки данных будет достаточно высоким. Для его снижения перегруппируем вершины таким образом, чтобы обратная связь охватывала только пару вершин α и β. Поскольку операция α ассоциативна, полную пирамиду размерности N можно заменить на пирамиду меньшей размерности $(N-1)$ и одиночную операционную вершину α, как показано на рис. 5, б. Вместе с вершиной α из линейки операций β будет вынесена одна ближайшая к пирамиде вершина β, поэтому в конечной вычислительной структуре на рис. 5, б обратной связью будет охвачено всего две вершины, по одной каждого типа. Чтобы обеспечить минимальный интервал обработки данных, размерность выделенных подграфов должна соотноситься с латентностями операционных вершин: $L = (l_\alpha + l_\beta)$.

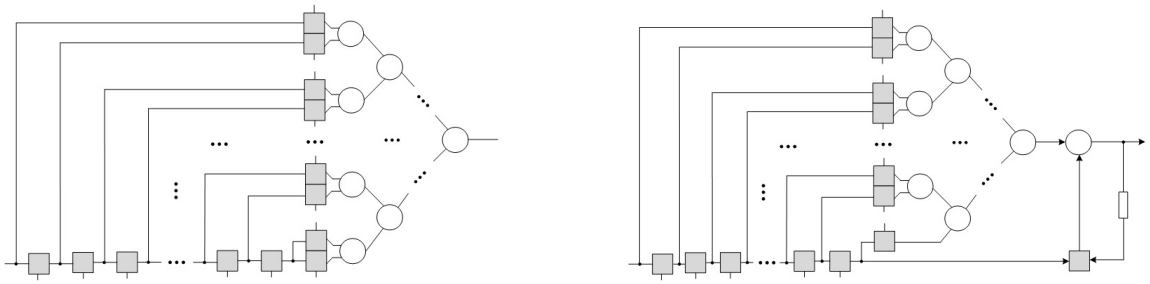
После описанного преобразования время решения задачи составит:

$$T_G = \left(L + \left\lceil \frac{N}{l_\alpha + l_\beta} \right\rceil \right) \cdot \tau, \tag{7}$$

где $L = (l_\alpha + l_\beta) \cdot l_\beta + \lceil \log_2(l_\alpha + l_\beta) \rceil \cdot l_\alpha)$ — длина критического пути в тактах для вычислительной структуры на рис. 5, б.

Если аппаратный ресурс реконфигурируемой ВС недостаточен даже для реализации вычислительной структуры на рис. 5, б, то целесообразно использовать минимальную структуру с парой вершин α и β. В таком случае время выполнения всех операций графа составит

$$T_G = (l_\alpha + l_\beta) \cdot \tau \cdot N. \tag{8}$$



(а) (б)

Рис. 5. Преобразованный фрагмент исходного информационного графа (а) и вычислительная структура (б), получаемая при вынесении пары вершин α и β

Таким образом, в зависимости от соотношения объемов вычислительного ресурса R_α и R_β, доступных для реализации вершин соответствующих типов, их латентностей l_α и l_β, а также размерности обрабатываемых массивов данных N возможно выделить несколько способов построения вычислительной структуры, обеспечивающих разные скорости решения задачи. Ниже приведено составное выражение, позволяющее рассчитать время выполнения всех операций графа для каждого из этих случаев:

$$T_G = \begin{cases} (l_\alpha + l_\beta) \cdot \lceil \log_2 N \rceil \cdot \tau, \ \text{ если } \ R_\alpha \geq N, R_\beta \geq \frac{N^2+N}{2}; \\ (l_\alpha \cdot \lceil \log_2 N \rceil + l_\beta \cdot N) \cdot \tau, \ \text{ если } \ R_\alpha \geq N, \ 2N \leq R_\beta < \frac{N^2+N}{2}; \\ \left\{ (l_\alpha + l_\beta - 1) \cdot l_\beta + \lceil \log_2(l_\alpha + l_\beta) \rceil \cdot l_\alpha + \lceil \frac{N}{l_\alpha + l_\beta} \rceil \right\} \cdot \tau \cdot (l_\alpha + l_\beta), \\ \text{если } \ l_\alpha + l_\beta \leq R_\alpha < N, \ 2 \cdot (l_\alpha + l_\beta) \leq R_\beta < 2N; \\ \left\{ (k-1) \cdot l_\beta + \lceil \log_2 k \rceil \cdot l_\alpha + \frac{N}{k} \right\} \cdot \tau \cdot (l_\alpha + l_\beta), \\ \text{если } \ R_\alpha < l_\alpha + l_\beta, \ R_\beta < 2 \cdot (l_\alpha + l_\beta), \end{cases} \tag{9}$$

где $(k-1)$ — число автоподстановок [8].

Первый вариант описывает предельный случай, когда аппаратного ресурса достаточно для реализации вычислительной структуры, соответствующей полному графу с пирамидальными подграфами на рис. 3, б, что позволяет выполнить все операции за наименьшее время. Второй вариант описывает ситуацию, когда аппаратного ресурса R_β не хватает на построение множества пирамидальных структур, но хватает на построение линейки, использование которой позволяет удалить повторяющиеся вершины (см. топологию на

рис. 4, г). Третий вариант описывает ситуацию, когда доступного ресурса не хватает на реализацию всего графа целиком, но достаточно для реализации подсистемы, охваченной обратной связью, включающей в себя $l_\alpha + l_\beta$ блоков α и $2 \cdot (l_\alpha + l_\beta)$ блоков β (см. вычислительную структуру на рис. 5, б). Наконец, последний вариант описывает случай, когда оборудования не хватает для того, чтобы реализовать даже описанную выше подсистему. В таком случае автоподстановка реализуется столько раз, сколько позволяет объем свободного аппаратного ресурса. В предельном случае $k = l_\alpha + l_\beta$ рассматриваемое выражение переходит в третье, а при $k = 1$ — в формулу (8) для минимальной вычислительной структуры с парой вершин α и β, охваченных обратной связью.

Если исходный последовательный граф состоит из чередующихся вершин, содержащих ассоциативные операции двух типов, одна из которых является дистрибутивной по отношению к другой, мы можем значительно ускорить выполнение всех операций графа. Степень этого ускорения зависит от того, сколько вычислительных блоков возможно реализовать на доступном нам оборудовании и как именно будет преобразован исходный граф. При этом в случае, если мы реализуем вычислительную систему с обратной связью, глубина которой и размер системы определяется суммарной латентностью блоков α и β, то мы получим лучший результат, чем при реализации всего графа целиком, при условии, что у нас не хватает ресурса для представления всех блоков β в виде пирамидальной структуры.

2. Ресурсонезависимая программа на языке Set@l

Традиционные методы параллельного программирования, как правило, оперируют информационным графом с фиксированной структурой, поэтому их использование для описания преобразований топологии графа с дистрибутивными операциями (см. рис. 1–5) достаточно трудоемко и малоэффективно. Код ресурсонезависимой программы, которая реализует рассмотренные преобразования, будет состоять из множества подпрограмм, связанных условными операторами и задающих отдельные варианты топологий. В отличие от классических языков параллельного программирования, возможности языка архитектурнонезависимого программирования Set@l позволяют описать принципы построения и модификации графов с дистрибутивными операциями в виде множеств типов «голова/хвост» и «разбиение пополам» и правил их декомпозиции. В таком случае исходный код программы описывает не отдельные реализации, а множество возможных топологий графа для заданной размерности задачи. Аспекты выбирают определенный вариант топологии, исходя из конкретных значений параметров конфигурации. Для изменения структуры информационного графа достаточно изменить тип и разбиение множеств, тогда как исходный код программы, описывающий математическую сущность алгоритма, остается неизменным.

Исходный код программы на языке Set@l, приведенный на рис. 6, описывает базовую последовательную топологию графа с дистрибутивными операциями (см. рис. 1). Отметим, что вершины α (`alpha`) и β (`beta`) могут соответствовать как простым операциям над числами с фиксированной запятой, так и более сложным операциям сложения и умножения комплексных чисел, векторов и матриц.

Полный граф описывается рекурсивно (`Rec`, см. строку (1) на рис. 6) как отношение `F` между множествами переменных `X` и коэффициентов `B` и результирующим элементом `r`. Поскольку граф строится по последовательному принципу «голова/хвост», в условии завершения рекурсии (`break`, строки (2)–(4)) и правиле раскрытия итераций (`main`, строки (5), (6)) используются соответствующие признаки `Head` («голова»: выделение первого

```
(1)     Rec[F(X,B,r)|set(X,B) && element(r)]:
(2)         break[cap(B)=1:
(3)             F(X,B,r)=conc{alpha(Head(Tail(X)),Head(X),i),
(4)                                 beta(i,Head(B),r)|element(i)}; ];
(5)         main[F(X,B,r)=conc{F(Tail(X),Tail(B),e),alpha(e,Head(X),i),
(6)                                 beta(i,Head(B),r)|element(e,i)};];
(7)     end(F);
```

Рис. 6. Исходный код программы на языке Set@l, описывающий последовательную топологию информационного графа с дистрибутивными операциями (см. рис. 1)

элемента множества) и `Tail` («хвост»: выделение подмножества, содержащего все элементы исходного множества, кроме первого). Альтернативный принцип построения графов, рассматриваемый в данной работе, — «разбиение пополам» — предполагает формирование параллельной пирамидальной структуры.

На каждой итерации (см. строки (5), (6) на рис. 6) линейку дистрибутивных операций `F` над множествами `X` и `B` можно представить как аналогичную линейку операций над «хвостами» этих множеств и пару связанных через элемент `i` вершин α и β, на входы которых подаются «головы» множеств `X` и `B` и промежуточный результат `e` последовательного выполнения ассоциативных операций `F` над `Tail(X)`, `Tail(B)`, а выход `r` является конечным или промежуточным результатом вычислений. Так как исходный последовательный граф не привязан к какой-либо вычислительной архитектуре, все его элементы связаны между собой по параллельно-зависимому принципу `conc` [6]. При выполнении условия завершения рекурсии (строки (2)–(4)) последняя линейка операций `F` преобразуется в особую пару вершин, на входы которой подаются оставшиеся элементы множеств `X` и `B`.

На рис. 7 представлен код на языке программирования Set@l, который описывает базовое преобразование пары дистрибутивных вершин α и β (признак `distr`, строки (1)–(4), см. рис. 2, а) и слияние пары ассоциативных вершин любого типа (признак `mrg`, строки (5)–(8), см. рис. 2, б), на основе которых осуществляется дальнейшая модификация последовательного информационного графа. Преобразование `mrg` позволяет передать тип `H/T` («голова/хвост») или `DIV2` («разбиение пополам») формируемому множеству `k`, подаваемому на вход блока `f`.

```
(1)     attribute(distr(a1,a2)|a1=alpha(a,b,i) && a2=beta(i,c,r)):
(2)         conc(a1,a2)=conc(beta(a,c,i),beta(b,c,ii),
(3)                                         alpha(i,ii,r)|element(ii));
(4)     end(distr);
(5)     attribute(mrg(a1,a2,z)|a1=f(a,b,c) && a2=f(c,d,e) &&
(6)
    z=type('H/T'||'DIV2')):
(7)         conc(a1,a2)=[f(a,k,e)|k=union(b,d) && type(k)=z];
(8)     end(mrg);
```

Рис. 7. Код базовых признаков, описывающих преобразование пары дистрибутивных вершин (`distr`) и слияние пары ассоциативных вершин (`mrg`), на языке программирования Set@l

Фрагмент программы на языке Set@l изображенный на рис. 8, определяет процедуру преобразования `Tr` линейного информационного графа с дистрибутивными операциями

(см. топологию на рис. 1 и код на рис. 6) к топологии с подграфами-ветвями (см. рис. 3, а). Движение по итерациям начинается от пары вершин α и β, соединенной с выходом `r`.

На каждой итерации (`main`, см. строки (6)–(8) на рис. 8) последовательно (`seq`) реализуются одно преобразование пары дистрибутивных вершин (`distr`, строка (6)) и два отдельных слияния (`mrg`, строки (6)–(8)) групп ассоциативных вершин α и β. В итоге граф представляется как объединение этих операций с аналогичным подграфом, вершины которого входят в множество `Tail`^{2}`(G)=Tail(Tail(G))`. Преобразования `mrg` осуществляются над группами вершин, связанных общими элементами `c`, а характер параллелизма объединяемых вершин определяется передаваемыми типами `talpha` и `tbeta` (`H/T` или `DIV2`). Строки (10)–(14) описывают параллелизм ветвей из вершин β, свойственный для графа с топологией на рис. 3, а.

Переход от топологии с последовательными подграфами-ветвями (см. рис. 3, а) к его параллельной реализации (см. рис. 3, б) может осуществляться за счет передачи типов `talpha='DIV2'` и `tbeta='DIV2'` при вызове процедуры `Tr`.

```
(1)  Rec[Tr(G,talpha,tbeta)|set(G),type(talpha,tbeta)]:
(2)      break[cap(G)=2:
(3)         Tr(G)=seq[distr(Head(G),Head(Tail(G))),
(4)         mrg(k,l,m|k,l in G && k=alpha(a,b,c) && l=alpha(c,d,e))]];
(6)      main[Tr(G)=seq[Tr(Tail^2(G)),distr(Head(G),Head(Tail(G))),
(7)          mrg(e,f,m|e,f in G && e=alpha(a,b,c) && f=alpha(c,d,e)),
(8)          mrg(x,y,z|x,y in G && y=beta(c,d,e) && z=tbeta)]];
(9)  end(Tr);
(10) attribute(Tpar(G)|set(G)):
(11)    set(K);
(12)    K=(a|a in G && a=beta(c,d,e));
(13)    type(K)='par';
(14) end(Tpar);
```

Рис. 8. Код преобразования исходного информационного графа (см. рис. 1) в структуру с последовательными подграфами-ветвями (см. рис. 3, а) на языке Set@l

На следующем этапе топология с подграфами-ветвями (см. рис. 3, а) оптимизируется путем удаления повторяющихся вершин β (см. рис. 8). Соответствующая процедура `Gcut` задается фрагментом кода на языке программирования Set@l, представленном на рис. 9. Признак `vcut` (см. строки (1)–(3)) выделяет пару вершин β с одинаковыми входами `c`, `f` и разными выходами `e`, `ee` и заменяет их на одну вершину с объединенным выходом `k`, в котором обе связи реализуются параллельно (`par`). Процедура `scut` (см. строки (4)–(7)) последовательно выделяет все повторяющиеся вершины в текущем графе `B` и применяет к ним преобразование `vcut`. Процесс продолжается до тех пор, пока из исходной группы не останется одна уникальная вершина (см. рис. 4, б). Процедура `Gcut` многократно повторяет преобразование `scut` и приводит граф к виду, показанному на рис. 4, в: все ветви из вершин β, кроме первой, содержат по одной вершине, а повторяющиеся вершины полностью удалены.

После оптимизации топологии графа (см. код на рис. 9) необходимо синтезировать вычислительную структуру `CS`, показанную на рис. 5, б. Для этого множества входных данных `XD` и `BD` типа `pipe` (конвейерная обработка) разбиваются на подмножества типа `conc` (параллельно-зависимая обработка), размерность которых определяется латентностями операционных вершин l_α (`la`) и l_β (`lb`), а затем преобразование `FG`, описанное на рис. 6–9, выполняется над каждым подмножеством (`sub`):

```
(1)     attribute(vcut(a1,a2)|a1=beta(c,f,e) && a2=beta(c,f,ee)):
(2)         par(a1,a2)=[beta(c,f,k)|k=par(e,ee) && e=ee];
(3)     end(vcut);
(4)     Rec[scut(B)|set(B)]:
(5)         break[cap(B)=1];
(6)         main[scut(B)=seq{vcut(Head(B),Head(Tail(B))),scut(Tail(B))}];
(7)     end(scut);
(9)     Rec[Gcut(G)|set(G)]:
(10)        B=(a,b,...|a,b in G && a=beta(c,f,e) && b=beta(c,f,ee));
(11)        break[cap(B)=1];
(12)        main[Gcut(G)=seq(Gcut(G),scut(B))];
(13)    end(Gcut);
```

Рис. 9. Фрагмента кода программы на языке Set@l, реализующий удаление повторяющихся вершин β (см. рис. 4)

```
XD=pipe(union[conc(x(a) ...x(b))|(x(a),x(b)) in X &&
                                a in set(1,la+lb+1, ...) &&
                                b in set(la+lb,2*(la+lb), ...)]);
BD=pipe(union[conc(b(c) ...b(d))|(b(c),b(d)) in B &&
                                c in set(1,la+lb+1, ...) &&
                                d in set(la+lb,2*(la+lb), ...)]]);
CS=FG(x,b,r| forall sub(x) in XD && forall sub(b) in BD);
```

В итоге будет получена вычислительная структура, изображенная на рис. 5, а. Чтобы перейти к конечной вычислительной структуре, изображенной на рис. 5, б, необходимо выделить множество входов A пирамиды из вершин α и модифицировать его следующим образом:

```
A=(a|beta(c,d,a) && alpha(a,b,e));
A*=(H/T(DIV2(a1...ak-1), ak)|k=cap(A));
```

Представленный в данном разделе код на языке программирования Set@l учитывает все случаи, выделенные в формуле (9), и позволяет описать реализацию информационных графов с дистрибутивными операциями на реконфигурируемых ВС в компактной ресурсонезависимой форме.

Заключение

Таким образом, в данной статье показан метод, который за счет перестановки вершин в информационном графе с дистрибутивными операциями и дальнейшей оптимизации вычислительной структуры позволяет ускорить выполнение всех операций графа, причем степень этого ускорения зависит от того, сколько вычислительных блоков возможно реализовать на имеющемся аппаратном ресурсе. Язык архитектурно-независимого программирования Set@l обеспечивает описание преобразований в компактной ресурсонезависимой форме. В отличие от традиционных языков параллельного программирования, в которых изменение топологии информационного графа требует модификации исходного кода программы, Set@l задает множество вариантов реализации в одной программе, а синтез конкретной вычислительной структуры осуществляется автоматически в соответствии с заданными пользователем параметрами конфигурации — объемом доступного вычислительного ресурса и латентностью базовых операций.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 20-07-00545.

Литература

1. Levin I.I., Dordopulo A.I., Pisarenko I., *et al.* Resource-Independent Description of Information Graphs with Associative Operations in Set@l Programming Language // Parallel Computing Technologies, 16th International Conference, PaCT 2021, Kaliningrad, Russia, September 13–18, 2021. Proceedings. Vol. 12942 / ed. by V. Malyshkin. Springer, 2021. Lecture Notes in Computer Science. P. 74–87. DOI: 10.1007/978-3-030-86359-3_6.
2. Tan L., Jiang J. Digital Signal Processing: Fundamentals and Applications. 2nd ed. Oxford: Elsevier Science, 2013. 896 p.
3. Winser A., Cranos M.W. Digital Signal Processing: Principles, Algorithms and System Design. London: Elsevier, 2017. 634 p.
4. Levin I.I., Dordopulo A.I., Pisarenko I.V., Melnikov A.K. Aspect-Oriented Set@l Language for Architecture-Independent Programming of High-Performance Computer Systems // Supercomputing. RuSCDays 2019. Vol. 1129 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2019. P. 517–528. Communications in Computer and Information Science. DOI: 10.1007/978-3-030-36592-9_42.
5. Levin I.I., Dordopulo A.I., Pisarenko I.V., Melnikov A.K. Objects of Alternative Set Theory in Set@l Programming Language // Parallel Computing Technologies, 15th International Conference, PaCT 2019, Almaty, Kazakhstan, August 19–23, 2019. Proceedings. Vol. 11657 / ed. by V. Malyshkin. Springer, 2019. P. 18–31. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-25636-4_3.
6. Левин И.И., Дордопуло А.И., Писаренко И.В., Мельников А.К. Язык архитектурно-независимого программирования вычислительных систем Set@l // Вестник компьютерных и информационных технологий. 2019. № 3. С. 48–56. DOI: 10.14489/vkit.2019.03.pp.048-056.
7. Карепова Е.Д. Основы многопоточного и параллельного программирования. Красноярск: Сиб. федер. ун-т, 2016. 356 с.
8. Levin I.I., Dudko S.A. Equivalent Transformations of Some Kinds of Computing Structures of Non-linear Recurrent Expressions for Reconfigurable Computing Systems // Parallel Computational Technologies. Vol. 1437 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2021. P. 3–17. Communications in Computer and Information Science. DOI: 10.1007/978-3-030-81691-9_1.

Левин Илья Израилевич, д.т.н., профессор, кафедра интеллектуальных и многопроцессорных систем, Южный федеральный университет (Таганрог, Российская Федерация)

Писаренко Иван Вадимович, «НИЦ супер-ЭВМ и нейрокомпьютеров» (Таганрог, Российская Федерация)

Михайлов Денис Васильевич, «НИЦ супер-ЭВМ и нейрокомпьютеров» (Таганрог, Российская Федерация)

Мельников Андрей Кимович, к.т.н., доцент, АО «Вычислительные решения» (Москва, Российская Федерация)

Дордопуло Алексей Игоревич, к.т.н., «НИЦ супер-ЭВМ и нейрокомпьютеров» (Таганрог, Российская Федерация)

DOI: 10.14529/cmse220201

RESOURCE-INDEPENDENT DESCRIPTION OF INFORMATION GRAPHS WITH DISTRIBUTIVE OPERATIONS IN THE SET@L PROGRAMMING LANGUAGE

© 2022 I.I. Levin[1], I.V. Pisarenko[2], D.V. Mikhailov[2], A.K. Melnikov[3], A.I. Dordopulo[2]

[1]*Southern Federal University (Nekrasovsky lane 44, Taganrog, 347928 Russia),*
[2]*Supercomputers and Neurocomputers Research Center (Italyansky lane 106, Taganrog, 347922 Russia),*
[3]*"Computational Solutions" JSC (Varshavskoe highway 125c17, Moscow, 119991 Russia)*
E-mail: iilevin@sfedu.ru, pisarenko@superevm.ru, mixailow.den@gmail.com, ak@comp-sol.ru, dordopulo@superevm.ru

Received: 05.05.2022

In the paper, we suggest to transform a standard sequential topology of an information graph with distributive operations into its hybrid version with sequential and parallel fragments. Such transformation allows to provide efficient description of calculations in the resource-independent form. The result topology depends on available hardware resource of a reconfigurable system and provides an increase in the special performance in comparison to the initial topology. We have developed an algorithm of a linear structure transformation into various hybrid topologies according to the configuration of the computing system. The algorithm is described in the Set@l programming language.

Keywords: distributive operations, resource-independent programming, reconfigurable computing systems, performance reduction, Set@l, "half-splitting" and "head/tail" attributes.

FOR CITATION

Levin I.I., Pisarenko I.V., Mikhailov D.V., Melnikov A.K., Dordopulo A.I. Resource-independent Description of Information Graphs with Distributive Operations in the Set@l Programming Language. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2022. Vol. 11, no. 2. P. 5–17. (in Russian) DOI: 10.14529/cmse220201.

References

1. Levin I.I., Dordopulo A.I., Pisarenko I., *et al.* Resource-Independent Description of Information Graphs with Associative Operations in Set@l Programming Language. Parallel Computing Technologies, 16th International Conference, PaCT 2021, Kaliningrad, Russia, September 13–18, 2021. Proceedings. Vol. 12942 / ed. by V. Malyshkin. Springer, 2021. Lecture Notes in Computer Science. P. 74–87. DOI: 10.1007/978-3-030-86359-3_6.

2. Tan L., Jiang J. Digital Signal Processing: Fundamentals and Applications. 2nd ed. Oxford: Elsevier Science, 2013. 896 p.

3. Winser A., Cranos M.W. Digital Signal Processing: Principles, Algorithms and System

Design. London: Elsevier, 2017. 634 p.

4. Levin I.I., Dordopulo A.I., Pisarenko I.V., Melnikov A.K. Aspect-Oriented Set@l Language for Architecture-Independent Programming of High-Performance Computer Systems. Supercomputing. RuSCDays 2019. Vol. 1129 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2019. P. 517–528. Communications in Computer and Information Science. DOI: 10.1007/978-3-030-36592-9_42.

5. Levin I.I., Dordopulo A.I., Pisarenko I.V., Melnikov A.K. Objects of Alternative Set Theory in Set@l Programming Language. Parallel Computing Technologies, 15th International Conference, PaCT 2019, Almaty, Kazakhstan, August 19–23, 2019. Proceedings. Vol. 11657 / ed. by V. Malyshkin. Springer, 2019. P. 18–31. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-25636-4_3.

6. Levin I.I., Dordopulo A.I., Pisarenko I.V., Melnikov A.K. Architecture-independent Set@l programming language for computer systems. Vestnik komp'iuternykh i informatsionnykh tekhnologii. 2019. No. 3. P. 48–56. DOI: 10.14489/vkit.2019.03.pp.048-056. (in Russian)

7. Karepova E.D. Fundamentals of Multithreaded and Parallel Programming. Krasnoyarsk: Publishing of the Siberian Federal University, 2016. 356 p. (in Russian)

8. Levin I.I., Dudko S.A. Equivalent Transformations of Some Kinds of Computing Structures of Non-linear Recurrent Expressions for Reconfigurable Computing Systems. Parallel Computational Technologies. Vol. 1437 / ed. by L. Sokolinsky, M. Zymbler. Cham: Springer, 2021. P. 3–17. Communications in Computer and Information Science. DOI: 10.1007/978-3-030-81691-9_1.