Современные методы межпроцедурного анализа программ*

А. С. Антонов†

6 мая 1998

Введение

Необходимость выполнения межпроцедурного анализа программ очень часто возникает на практике, в частности при анализе параллельных свойств программ. В общем случае под таким анализом понимается не только определение влияния вызова процедур на контекст вызова [5], но и определение влияния контекста вызова на исполнение вызываемой подпрограммы. Первое помогает определить параллельные свойства вызывающей подпрограммы, а второе — вызываемой. Можно привести множество примеров задач, решаемых с использованием техники межпроцедурного анализа: определение независимости вхождений в тело подпрограммы параметров и глобальных переменных, распараллеливание циклов, содержащих вызовы подпрограмм, определение необходимых пересылок данных для вызова распределяемой подпрограммы при использовании компьютеров с распределенной памятью, поддержка корректности данных в кэш-памяти многопроцессорных систем и многие другие. Если не производить межпроцедурный анализ, то придется предположить, что все фактические параметры и внешние переменные как используются, так и переопределяются в вызываемой подпрограмме, а также что они потенциально могут ссылаться на одну и ту же область памяти. Это приведет к тому, что многие полезные свойства программы не будут использованы.

Одним из первых методов разрешения проблемы межпроцедурного анализа была предложена *подстановка тела подпрограммы* на место вызова (*inline expansion* [25]), но она сильно затруднена, если в графе вызовов есть контуры, что приводит к сильному увеличению размера кода и времени анализа.

В известных обзорах [5, 26] большое внимание уделялось методам межпроцедурного анализа без учета индексных переменных. Решаемые этими методами задачи, такие как *совмещение параметров процедуры (aliasing)*, анализируемое с точностью до имен, *межпроцедурное распространение констант (constant propagation)* и *определение входных и выходных параметров* процедур с точностью до имен, решаются

*Работа поддержана Российским Фондом Фундаментальных Исследований, грант 97–01–00977

† E-mail: asa@vvv.srcc.msu.su

довольно простыми методами и подробно описаны. Кроме того, такой анализ является весьма грубым и недостаточным для реальных программ, поскольку в большинстве случаев необходим анализ существования зависимости между ссылками на отдельные элементы массивов. В работе [24] на основе изучения некоторой выборки программ на языке Паскаль делается даже вывод о бесполезности такого анализа для преобразования реальных программ. Поэтому под межпроцедурным анализом в дальнейшем будем понимать межпроцедурный анализ, производимый с точностью до отдельных элементов массивов.

Методы межпроцедурного анализа с учетом индексных переменных в последнее время получили дальнейшее развитие во многих работах. Анализу и сравнению таких методов посвящена данная статья. Кроме того, рассматривается новый подход, основанный на анализе свойств *графа алгоритма* [1, 2, 3], впервые описанный в работе [4].

Межпроцедурный анализ можно разделить на две части: нахождение входных и выходных данных подпрограммы и описание этих данных в терминах фактических параметров (обратная подстановка). Разделы 2–6 данной работы посвящены существующим подходам к решению первой задачи, а раздел 7 — подходам к решению второй.

1 Основные положения

В большинстве работ, посвященных межпроцедурному анализу, рассматриваются только те области, которые соответствуют вырезкам из массивов, используемым или изменяемым в анализируемой подпрограмме. Следуя [14], будем называть такие области READ и WRITE областями. Однако знания их может оказаться недостаточно для полноценного анализа, так как READ области могут содержать данные, вычисленные ранее в исследуемой подпрограмме, а, следовательно, они не будут представлять входных данных анализируемой процедуры. WRITE области могут содержать данные, которые не потребуются нигде в дальнейшем тексте программы, что также не соответствует выходным данным подпрограммы. Для более точного анализа вводятся IN и OUT области, которые представляют именно входные и выходные данные подпрограммы, то есть данные, необходимые для выполнения подпрограммы, и данные, вычисляемые в исследуемой подпрограмме и используемые где-либо далее.

Все методы описания READ и WRITE областей можно разделить на два класса: неточные и точные методы. Неточные методы проще, но, в отличие от точных, описывают только некоторую аппроксимацию необходимой области. Точные же методы могут потребовать много памяти и могут потребовать много времени для анализа программ. В некоторых случаях могут использоваться комбинированные методы, например, приближенное описание объединения точно описанных областей.

Для описанных любым методом областей наиболее важными операциями явля-

ются объединение и пересечение. Объединение областей позволяет суммировать READ и WRITE области для всех вхождений какой-нибудь переменной, а пересечение необходимо для определения существования зависимостей между различными вхождениями. В случае пересечения областей нас интересует прежде всего наличие или отсутствие зависимости, то есть сам факт целочисленной пустоты или непустоты получаемой области. Две основные операции объединения и пересечения будут рассмотрены для всех методов.

Все описываемые далее методы проиллюстрируем на одном примере — опишем с помощью каждого метода WRITE область для фрагмента программы, производящего вычисления для нижнетреугольной части массива и одной из его строк. Этот фрагмент и суммарная WRITE область приведены на рис. 1. На этом рисунке, как и на всех последующих, изображающих аппроксимации этой области, предполагается, что элемент A(1, 1) находится в левом верхнем углу, а элементы первой размерности массива располагаются сверху вниз.

```
      PARAMETER (N = 13, K = 7)
      DIMENSION A(N+1, N+1)
C обращение к нижнетреугольной
C части массива
      DO I = 1, N
        DO J = 1, I
          A(I, J) = ...
        END DO
      END DO
C обращение к строке с номером K
      DO J = 1, N
        A(K, J) = ...
      END DO
```

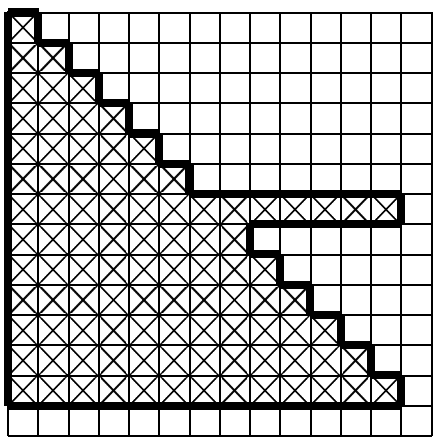

Рис. 1. *Фрагмент программы и вычисляемая область массива*

Следуя [13], опишем отдельное обращение к элементам массива при помощи трех векторов: вектора индексных выражений d-мерного массива $\Phi(A) = (\phi_1, \ldots, \phi_d)$, вектора пар ограничений на управляющие параметры l объемлющих циклов $B(A) = (\beta_1, \ldots, \beta_l)$, где β_j — упорядоченная пара (lb_j, ub_j) ограничений снизу и сверху на параметр j-ого цикла, и вектора шагов циклов $\Sigma(A) = (\sigma_1, \ldots, \sigma_l)$. Тогда описатель доступа к массиву $\Delta(A)$ представляется в виде тройки $< \Phi, B, \Sigma >$.

2 Неточные методы получения READ и WRITE областей

Регулярные секции (regular sections) были предложены в работе [10] как простой способ аппроксимации READ и WRITE областей. Они бывают двух основных ти-

пов — ограниченные регулярные секции (restricted regular sections [9]) и описания с помощью триплетов (bounded regular sections [16]).

Ограниченные регулярные секции представляются с помощью вектора индексных выражений массива $\Phi(A)$, в котором каждая компонента ϕ_k является либо константным выражением (инвариантом подпрограммы), либо выражением вида $\alpha \pm i_j$, где i_j — переменная, сопоставляемая j-ому измерению массива ($k \neq j$), а α — константное выражение (инвариант подпрограммы), либо особым значком $\perp$, который означает всю размерность массива. С помощью такой конструкции можно описать отдельный элемент массива, полные строку, столбец или диагональ, а также их многомерные аналоги.

Объединение двух ограниченных регулярных секций получается следующим образом. Если на k-том месте в одной из секций стоит $\perp$, то на соответствующем месте ϕ_k в объединении также будет $\perp$. Если на k-том месте в обеих секциях стоит одна и та же константа, то в объединении также будет эта константа. Если на соответствующих местах стоят разные константы, то в объединении будет $\perp$. Если на k-том месте хотя бы в одной из секций стоит выражение $\alpha \pm i_j$, то проводится процесс, называемый *нормализацией*[26], который приводит к тому, что в каждой из секций ϕ_k представляется либо значком $\perp$, либо выражением $\alpha \pm i_j$, где $j < k$ и $\phi_j = \perp$. После этого в объединении будет $\alpha \pm i_j$, если эти выражения для обеих секций в точности идентичны, или $\perp$ в противном случае.

Построение пересечения двух секций осуществляется также сравнением векторов индексных выражений. Для существования непустого пересечения для каждого элемента векторов должно выполняться хотя бы одно из четырех условий:

1. В одном из векторов стоит значок $\perp$.
2. В одном из векторов стоит $\alpha \pm i_j$, а в другом выражение, не зависящее от i_j.
3. В обоих векторах стоит $\alpha + i_j$ или $\alpha - i_j$ с одинаковой константой α.
4. В обоих векторах стоит константа α.

Если хотя бы для одной пары элементов векторов ни одно из этих условий не выполняется, то пересечение пусто, иначе непусто.

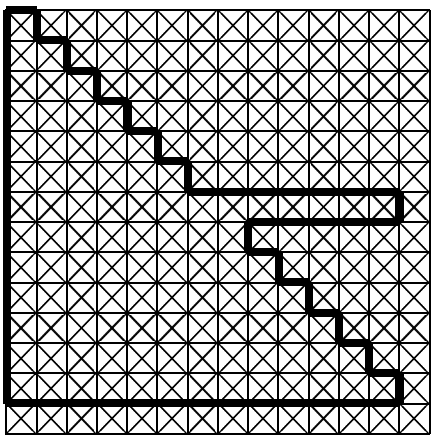

Рис. 2. *Область, полученная методом ограниченных регулярных секций*

Для примера, приведенного на рис. 1, WRITE область, полученная методом ограниченных регулярных секций, описывается следующим образом:

$$\begin{array}{lcl} RRS_1(A) & = & (\perp, \perp), \\ RRS_2(A) & = & (K, \perp), \\ RRS_{1\cup 2}(A) & = & (\perp, \perp). \end{array}$$

Следовательно, получается описание всего массива A (рис. 2).

В отличие от ограниченных регулярных секций, *описание с помощью триплетов (bounded regular sections)* не позволяет задавать диагонали, но включает информацию о границах и шагах циклов. Триплет описывается для каждого измерения массива с использованием элементов векторов $B(A)$ и $\Sigma(A)$ в виде $[lb_i : ub_i : \sigma_i]$, где lb_i, ub_i и σ_i — константы или инвариантные выражения подпрограммы.

Объединение двух описанных с помощью триплетов секций строится следующим образом. В качестве нижней (верхней) границы по каждому измерению берется минимальная (максимальная) из нижних (верхних) границ по данному измерению из описаний объединяемых областей, а в качестве шага — наибольший общий делитель шагов из описаний этих областей.

Для определения пересечения двух секций нужно взять пару триплетов для каждого измерения $[lb_i^1 : ub_i^1 : \sigma_i^1]$ и $[lb_i^2 : ub_i^2 : \sigma_i^2]$ и составить систему равенств и неравенств:

$$\begin{cases} lb_i^1 + \sigma_i^1 k_i^1 = lb_i^2 + \sigma_i^2 k_i^2 \\ 0 \le k_i^1 \le (ub_i^1 - lb_i^1)/\sigma_i^1 \\ 0 \le k_i^2 \le (ub_i^2 - lb_i^2)/\sigma_i^2, \end{cases}$$

где $k_i^1, k_i^2 \in Z$.

Если описываемая такой системой область целочисленно непуста для каждого измерения, то пересечение двух секций непусто. Целочисленную непустоту области можно определять, например, с помощью Омега-теста[23]. В некоторых случаях эту задачу можно решить с помощью простых эвристик.

Для рассматриваемого примера (рис. 1) WRITE область, полученная методом описания с помощью триплетов, описывается следующим образом:

$$\begin{array}{lcl} BRS_1 & = & ([1 : N : 1], [1 : N : 1]), \\ BRS_2 & = & (K, [1 : N : 1]), \\ BRS_{1\cup 2} & = & ([1 : N : 1], [1 : N : 1]). \end{array}$$

Эта область показана на рис. 3.

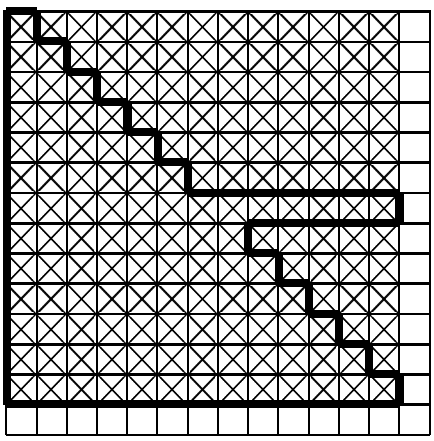

Рис. 3. *Область, полученная методом описания с помощью триплетов*

Простые секции (simple sections), предложенные в работе [6], представляют собой области массивов, которые могут быть описаны при помощи набора ограничений специального вида. Для описания областей предлагается использовать гиперплоскости, которые либо параллельны координатным гиперплоскостям, либо наклонены под углом 45° к любым двум таким гиперплоскостям. При этом получается аппроксимация нужного множества обращений к массиву выпуклой областью простого вида. В объединение двух областей войдут минимальное из ограничений снизу и максимальное из ограничений сверху из описаний рассматриваемых областей для каждой координатной гиперплоскости и для каждой диагонали. В пересечение двух областей войдут максимальное из ограничений снизу и минимальное из ограничений сверху из описаний объединяемых областей для каждой координатной гиперплоскости и для каждой диагонали. Пересечение двух простых секций непусто, если максимальное из ограничений снизу не превосходит минимальное из ограничений сверху для каждой координатной гиперплоскости и для каждой диагонали.

WRITE область, полученная методом простых секций для рассматриваемого примера (рис. 1), описывается следующим образом:

$$
\begin{aligned}
SS_1 \quad &= \{\ 1 \le I \le N,\ 1 \le J [\le N],\ [2 \le] I + J [\le 2N],\\
&\qquad 0 \le I - J [\le N - 1]\ \},\\
SS_2 \quad &= \{\ K \le I \le K,\ 1 \le J \le N,\ [K + 1 \le] I + J [\le K + N],\\
&\qquad [K - N \le] I - J [\le K - 1]\ \},\\
SS_{1\cup 2} \quad &= \{\ 1 \le I \le N,\ 1 \le J \le N,\ [2 \le] I + J [\le 2N],\\
&\qquad K - N \le I - J [\le N - 1]\ \}.
\end{aligned}
$$

В квадратные скобки заключены получившиеся избыточные неравенства. Результирующая область показана на рис. 4.

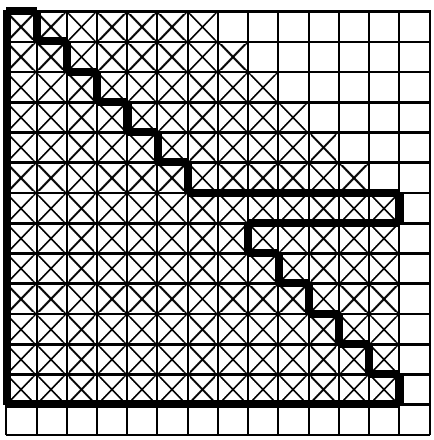

Рис. 4. *Область, полученная методом простых секций*

В работах [14, 28] рассматривается описание множества обращений к массиву в виде минимальной *выпуклой оболочки (convex hull)* этого множества. Выпуклая оболочка описывается с помощью упорядоченного *набора индексных выражений (region)* и *контекста исполнения (execution context)*. Под контекстом исполнения понимается набор ограничений на управляющие параметры циклов. Результирующим описанием READ и WRITE областей является набор линейных равенств и неравенств, описывающих выпуклую область.

Для рассматриваемого примера (рис. 1) WRITE область аппроксимируется при помощи минимальной выпуклой оболочки следующим образом:

$$
\begin{aligned}
CH_1 \quad &= \{\ I \geq 1,\ I \leq N,\ J \geq 1,\ J \leq I\ \},\\
CH_2 \quad &= \{\ I = K,\ J \geq 1,\ J \leq N\ \},\\
CH_{1\cup 2} \quad &= \{\ I \geq 1,\ I \leq N,\ J \geq 1,\ J \leq N,\ I \geq \tfrac{(K-1)J+N-K}{N-1}\ \}.
\end{aligned}
$$

Результирующая область изображена на рис. 5.

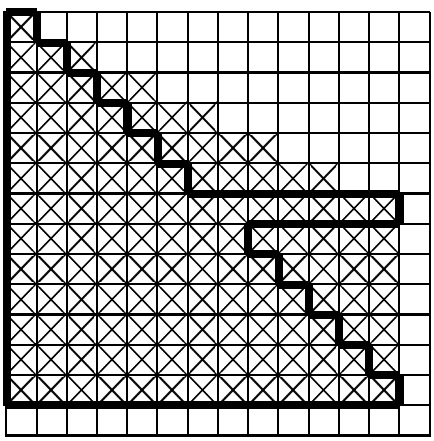

Рис. 5. *Область, заданная минимальной выпуклой оболочкой*

3 Точные методы получения READ и WRITE областей

Подход Бурке-Сайтрона [8] заключается в том, что вся память рассматривается как одномерный массив MEM. Ссылки на элементы конкретного массива преобразуются в ссылки на элементы массива MEM, причем важно только взаимное расположение исследуемых массивов. При использовании этого подхода сразу разрешаются проблемы, связанные с совмещением параметров подпрограммы (aliasing). Для слияния нескольких линейных участков может применяться огрубление результата (предположение, что элементы между верхней границей одного линейного участка массива MEM и нижней границей другого участка принадлежат результирующей области); в этом случае данный метод становится неточным. Объединение нескольких линейных участков, относящихся к данному массиву, может не быть линейным участком, поэтому его далеко не всегда можно точно представить с помощью данного подхода. Для построения пересечения двух областей с целью определения существования зависимости нужно перебрать все сочетания линейных участков из описаний разных областей.

Для нашего примера (рис. 1) получатся следующие описания:

$$\begin{aligned} M_1 &= MEM[(J-1)*(N+1)+I+S_A],\ 1 \le I \le N,\ 1 \le J \le I, \\ M_2 &= MEM[(J-1)*(N+1)+K+S_A],\ 1 \le J \le N, \end{aligned}$$

где S_A — сдвиг начала массива A относительно начала массива MEM. Точное слияние M_1 и M_2 в данном случае невозможно, поэтому нужно либо хранить оба описания, либо огрублять результат, что приведет к описанию такой же области, как и в методе описания с помощью триплетов (рис. 3).

В работе [27] описывается метод, в котором для описания отдельного обращения к массиву используется система линейных равенств и неравенств, а для объединения областей, соответствующих нескольким обращениям, строится *совокупный образ (merged image)*. Совокупный образ получается введением новых целочисленных переменных. Пусть p обращений к массиву описываются с помощью p систем линейных равенств и неравенств $\vec{\phi}_j = F_j\vec{i}_j + \vec{c}_j$, $B_j\vec{i}_j \le \vec{b}_j$, $1 \le j \le p$, где $\vec{i}_j$ — вектор управляющих параметров циклов, B_j и $\vec{b}_j$ — матрица и вектор свободных членов, задающие ограничения на границы изменения параметров циклов, а F_j и $\vec{c}_j$ — матрица и вектор, задающие индексные выражения обращения к массиву. Тогда для формального объединения всех p областей в одну вводится вектор из p целочисленных переменных $\vec{x} = (x_1, \ldots, x_p)$, таких что $x_1 + \ldots + x_p = p-1$, $0 \le x_i \le 1$, $1 \le i \le p$, и результирующая область записывается с помощью неравенств:

$$B_j\vec{i}_j \le \vec{b}_j,\ (\vec{\phi}^{min} - \vec{\phi}^{max})x_j \le \phi - F_j\vec{i}_j - \vec{c}_j \le (\vec{\phi}^{max} - \vec{\phi}^{min})x_j,$$

где $\vec{\phi}^{min}$ и $\vec{\phi}^{max}$ — любые вектора, такие что $\vec{\phi}^{min} \le F_j\vec{i}_j + \vec{c}_j \le \vec{\phi}^{max}$ для любого j, при котором $B_j\vec{i}_j \le \vec{b}_j$.

Для рассматриваемого примера (рис. 1) две WRITE области описываются следующими системами равенств и неравенств:

$$\begin{aligned} T_1 &= \{\ \phi_1 = I,\ \phi_2 = J,\ 1 \le I \le N,\ 1 \le J \le I\ \}, \\ T_2 &= \{\ \phi_1 = K,\ \phi_2 = J,\ 1 \le J \le N\ \}. \end{aligned}$$

В качестве $\vec{\phi}^{min}$ и $\vec{\phi}^{max}$ возьмем вектора из констант, взятых из описания массива A:

$$\vec{\phi}^{min} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}; \quad \vec{\phi}^{max} = \begin{pmatrix} N+1 \\ N+1 \end{pmatrix}.$$

Тогда совокупный образ двух областей T_1 и T_2 будет следующим:

$$\begin{aligned} T_{1\cup 2} = \{\ & x_1 + x_2 = 1,\ 0 \le x_1 \le 1,\ 0 \le x_2 \le 1, \\ & 1 \le I_1 \le N,\ 1 \le J_1 \le I_1,\ 1 \le J_2 \le N, \\ & -Nx_1 \le \phi_1 - I \le Nx_1, \\ & -Nx_1 \le \phi_2 - J \le Nx_1, \\ & -Nx_2 \le \phi_1 - K \le Nx_2, \\ & -Nx_2 \le \phi_2 - J \le Nx_2\ \}. \end{aligned}$$

4 Сравнение методов получения READ и WRITE областей

Результаты сравнения описанных методов основаны на анализе, проведенном в работе [13].

По требуемому объему памяти для описания области. Для описания ограниченной регулярной секции требуется указать d компонент вектора индексных выражений массива $\Phi(A)$, для чего необходим объем памяти, пропорциональный d. Для задания регулярной секции с использованием триплетной нотации также требуется объем памяти, пропорциональный d, но с бóльшим коэффициентом, так как для каждого измерения массива требуется хранить три поля. Для описания простой секции требуется $2d$ ограничений, параллельных координатным плоскостям, и $2d^2 - 2d$ ограничений, задающих все возможные диагонали. В итоге получается, что объем требуемой памяти пропорционален d^2. Описание в виде выпуклой оболочки трудно оценить по затратам памяти, поскольку многое зависит от конкретной ситуации. Описание отдельного обращения к массиву при помощи системы линейных неравенств требует задания неравенств на индуктивные переменные циклов и равенств для всех измерений массива. Требуемый для этого объем памяти пропорционален $l(l+1)/2 + d\,l$ (l — количество объемлющих циклов). Требования к объему памяти в подходе Бурке-Сайтрона зависят от способа хранения коэффициентов, но по порядку это величина, пропорциональная $l(d+l)$.

По сложности операции объединения ссылок на один и тот же массив. Объединение ограниченных регулярных секций требует проведения нормализации,

что требует $O(d^2)$ шагов по времени. Объединение регулярных секций, заданных с помощью триплетов, требует только линейного просмотра ссылок на массив, то есть $O(d)$ шагов по времени. Объединение двух простых секций требует просмотра всех ограничений, то есть затраты по времени составят величину порядка $O(d^2)$. Время на получение минимальной выпуклой оболочки двух областей не может быть выражено через d и l. Согласно [28] для этого обычно требуется $O(I^2V)$ шагов по времени, где I — число неравенств в описаниях областей, а V — число переменных в этих неравенствах. Построение объединения двух областей в подходе Бурке-Сайтрона может пониматься либо как просто набор линейных участков, либо аппроксимироваться, либо в определенных случаях возможно слияние, что, однако, может потребовать значительных затрат времени. Построение совокупного образа требует линейного просмотра ссылок на массив для нахождения векторов $\vec{\phi}^{min}$ и $\vec{\phi}^{max}$, то есть затраты по времени составят величину порядка $O(d)$.

По сложности операции пересечения (анализ на существование зависимости). Для неточных методов не имеет смысла применять точные тесты на наличие зависимостей, то есть полностью реализовывать операцию пересечения. Поэтому время на построение пересечения зависит от применяемого теста на наличие зависимости. Построение пересечения для простых секций требует $O(d^2)$ шагов по времени. Для определения пересечения двух областей, задаваемых при помощи метода Бурке-Сайтрона, нужно проанализировать возможности пересечения всех возможных пар линейных участков. Если одна из областей состоит из m_1 линейных участков, а другая — из m_2, то для построения пересечения необходимо $O(m_1m_2)$ раз произвести операцию элементарного пересечения. При построении совокупного образа этого удается избежать, так как все ссылки на массив объединены в одном описателе.

5 Получение IN и OUT областей из READ и WRITE областей

Методы получения IN и OUT областей из READ и WRITE областей подробно описаны в [14, 15]. Под IN областью массива, используемого в анализируемой подпрограмме, понимается множество элементов данного массива, необходимых для выполнения подпрограммы и не вычисляемых ранее в ее теле, то есть полученных извне данной подпрограммы. Под OUT областью массива понимается множество элементов, вычисляемых в теле подпрограммы и используемых где-либо после выхода из нее. IN и OUT области точно описывают влияние подпрограммы на контекст ее вызова, то есть решают одну из задач межпроцедурного анализа.

Процесс вычисления IN областей идет “снизу вверх” — от вызываемых подпрограмм к вызывающим. Внутри одной подпрограммы вычисление IN области идет в порядке, обратном направлению дуг графа управления. Сначала в качестве IN области принимается READ область, соответствующая последнему оператору, вы-

числяющему элементы анализируемого массива. Затем при движении по операторам к началу подпрограммы из текущей области вычитаются WRITE области, соответствующие данному массиву, и добавляются соответствующие READ области. Подобный же алгоритм существует и для получения IN области DO-цикла, суммирующей IN области со всех его итераций.

Для получения OUT области подпрограммы недостаточно анализа ее тела, необходимо знание использования элементов данной области в последующем тексте. Поэтому процесс вычисления OUT областей идет "сверху вниз" — от вызывающих подпрограмм к вызываемым. Сначала для анализируемого массива получаются IN области для операторов вызывающей подпрограммы и других вызовов после вызова данной подпрограммы. Объединение полученных областей пересчитывается в терминах формальных параметров анализируемой подпрограммы, после чего и начинается собственно процесс построения OUT области для данной подпрограммы. Для всех операторов подпрограммы, перебираемых в порядке, задаваемом дугами графа управления, берется пересечение WRITE областей, соответствующих анализируемому массиву с переданной в подпрограмму областью, после чего берется объединение таких пересечений.

6 Получение IN и OUT областей с помощью графа алгоритма

Использование графа алгоритма, введенного в [1, 2, 3], позволяет получить точное описание входных и выходных данных фрагмента программы.

Основная идея этого метода заключается в том, чтобы получить описание нужного множества в пространстве элементов массива средствами анализа пространства итераций. Если из всей области срабатывания оператора Ω_J [3] вычесть все области Δ_k из описания графа алгоритма фрагмента по входу A_i (точки областей Δ_k являются конечными точками дуг графа алгоритма данного фрагмента), то полученная область Ω_{inp} будет описывать множество точек пространства итераций, в которых A_i потребляет входные для исследуемого фрагмента данные.

Теперь нужно получить описание области пространства итераций Ω_{inp} в пространстве элементов массива A. Рассмотрим задачу для входа $A_i(P(J))$ массива A, где $P(J)$ — это векторная функция, определяющая индексные выражения данного входа. Будем предполагать, что для входа A_i найдена подобласть пространства итераций Ω^i_{inp}, в каждой точке которой аргументом для входа A_i являются начальные данные. По определению данной области, для любой точки $J \in \Omega^i_{inp}$ элемент массива $A_i(P(J))$ нигде в данном фрагменте не вычисляется, а берется из входных данных, т.е. является элементом искомого множества.

Сконструируем вспомогательный фрагмент, содержащий вход A_0 по переменной

A:

```
DO  I_1 = l_1, u_1
  ...
  DO  I_d = l_d, u_d
    ... = ... A_0(I_1, I_2, ..., I_d) ...
  END DO
  ...
END DO  ,
```

где d — это размерность переменной A, а l_k, u_k — нижняя и верхняя границы k-го измерения массива A соответственно, $k = \overline{1,d}$. Будем считать, что данный фрагмент достижим из каждой точки программы и всегда срабатывает последним — этого всегда можно добиться эквивалентным преобразованием фрагмента.

Возьмем любую точку J области Ω^i_{inp}. Ясно, что элемент массива $A_i(P(J)) = A_i(p_1(J), \ldots, p_d(J))$ принадлежит искомому множеству и надо найти описание всех подобных точек $P(J)$ в пространстве элементов массива A.

Предположим, что в каждой точке J области Ω^i_{inp} происходит не использование, а определение элемента $A_i(P(J))$, т.е. вход A_i будет играть роль выхода. Будем считать областью срабатывания оператора, содержащего A_i, не область Ω_J, а область Ω^i_{inp}. Область срабатывания входа A_0 определяется только границами циклов вспомогательного фрагмента, так как он безусловно достижим из каждой точки программы.

В таких предположениях решим стандартную задачу построения элементарного графа алгоритма $A_i \to A_0$ и найдем область Δ_{A_0}, на которой определены дуги графа алгоритма. Особенность множества Δ_{A_0} заключается в том, что, являясь многогранником в пространстве итераций, он одновременно является и описанием множества входных данных в пространстве элементов массива A для входа A_i.

Аналогичным образом данная задача решается для всех входов, а искомое подмножество входных элементов массива A является объединением областей, полученных при решении данной задачи для каждого отдельного входа.

Использование такого метода позволяет получить точное описание IN и OUT областей подпрограммы. Существование эффективных алгоритмов построения графа алгоритма обеспечивает возможность использования этого метода при анализе реальных программ.

7 Описание входных и выходных данных подпрограммы в терминах фактических параметров

Перейдем теперь к решению второй задачи межпроцедурного анализа — описанию входных и выходных данных подпрограммы в терминах фактических параметров.

Описываемый здесь метод опирается на [14] и требует, чтобы входные и выходные данные подпрограммы уже были описаны в виде системы линейных неравенств.

Пусть в программе есть две подпрограммы P и Q, такие, что:

SUBROUTINE P(...)
DIMENSION $A_p(lp_1 : up_1, \ldots, lp_p : up_p)$
...
CALL Q(..., $A_p(op_1, \ldots, op_p)$, ...)
...
END

SUBROUTINE Q(...,A_q,...)
DIMENSION $A_q(lq_1 : uq_1, \ldots, lq_q : uq_q)$
...
END

Пусть элементы массива A_q, представляющие часть входных и выходных данные подпрограммы Q, представлены в виде области Ω_q, описанной с помощью набора линейных равенств и неравенств. Требуется пересчитать эту область в терминах вырезки из соответствующего фактического параметра-массива A_p.

Запишем условие Γ_{pq} того, что два элемента массивов $A_p(\psi_1, \ldots, \psi_p)$ и $A_q(\phi_1, \ldots, \phi_q)$ ссылаются на одну и ту же область памяти:

$$\Gamma_{pq}: \ (\psi_1 - lp_1) + \sum_{i=2}^{p} \left[(\psi_i - lp_i) \prod_{j=1}^{i-1} (up_j - lp_j + 1) \right] - offset =$$

$$= (\phi_1 - lq_1) + \sum_{i=2}^{q} \left[(\phi_i - lq_i) \prod_{j=1}^{i-1} (uq_j - lq_j + 1) \right],$$

где

$$offset = (op_1 - lp_1) + \sum_{i=2}^{p} \left[(op_i - lp_i) \prod_{j=1}^{i-1} (up_j - lp_j + 1) \right].$$

Тогда пересечение трех областей $\Omega = \Omega_q \cap \Gamma_{pq} \cap \{lp_i \le \psi_i \le up_i \mid i = 1, p\}$ неявно задает область Ω_p массива A_p, соответствующую области Ω_q массива A_q. Для получения явного описания Ω_p необходимо получить проекцию $(p+q)$-мерной области Ω на p-мерное подпространство, соответствующее массиву A_p. Это можно сделать с помощью исключений Фурье-Моцкина [21, 23], если равенство Γ_{pq} линейно. Определение условий его линейности рассматривается в дальнейшем изложении.

Если равенство Γ_{pq} нелинейно, то при некоторых условиях можно получить более простое условие. Если массивы A_p и A_q имеют одинаковое число элементов в первых $(d-1)$ размерностях (то есть $up_i - lp_i = uq_i - lq_i$, $1 \le i \le d-1$), и $op_i = lp_i$, $1 \le i \le d-1$,

то добавим в описание области Ω равенства $\psi_i - lp_i = \phi_i - lq_i, 1 \le i \le d-1$ и составим частичное уравнение ранга d:

$$\Gamma^d_{pq}: \ (\psi_d - lp_d) + \sum_{i=d+1}^{p} \left[(\psi_i - lp_i) \prod_{j=d}^{i-1} (up_j - lp_j + 1) \right] - offset =$$

$$= (\phi_d - lq_d) + \sum_{i=d+1}^{q} \left[(\phi_i - lq_i) \prod_{j=d}^{i-1} (uq_j - lq_j + 1) \right],$$

где

$$offset = (op_d - lp_d) + \sum_{i=d+1}^{p} \left[(op_i - lp_i) \prod_{j=d}^{i-1} (up_j - lp_j + 1) \right].$$

Это уравнение проще, чем Γ_{pq}, и в реальных случаях может оказаться линейным, в то время как полное уравнение таковым не является. Если количество размерностей с одинаковым числом элементов равно q, но меньше p, то в описание области Ω вместо условия Γ^d_{pq} нужно добавить равенства $\{\psi_i = op_i, \mid i = q+1, p\}$.

Практически такой же метод описан и в [27]. Там рассмотрение ограничивается случаем, когда число измерений формального параметра-массива не превосходит числа измерений соответствующего фактического параметра, размеры этих массивов совпадают в первых $(d-1)$ размерностях и могут отличаться только в d-ой размерности. Этих условий достаточно, чтобы равенство Γ^d_{pq} было линейным, то есть для проведения полного анализа и предыдущим методом. Сам же описанный здесь метод отличается от предыдущего практически только тем, что формулы не выписываются явно, а получаются через последовательность промежуточных шагов.

Для того, чтобы получить проекцию $(p+q)$-мерной области Ω на p-мерное пространство, необходимо исключить переменные, введенные для описания Ω_q, из всех равенств и неравенств. Это можно сделать методом Фурье-Моцкина ([21] [23]), если все ограничения, содержащие эти переменные, линейны. Так как в описание Ω_q входят только линейные ограничения, нелинейность по данным переменным может возникнуть только в равенстве $\Gamma_{pq}(\Gamma^d_{pq})$. Кроме того, для проведения дальнейшего анализа программы важно, чтобы все получающиеся ограничения также были линейными. Поэтому нужно выделить условия на внешние переменные, при которых $\Gamma_{pq}(\Gamma^d_{pq})$ будет линейным по всем переменным.

Для этого берем равенство $\Gamma_{pq}(\Gamma^d_{pq})$, приводим в нем подобные, после чего приравниваем к нулю все нелинейные части. Если из получившихся равенств можно выразить переменные, введенные для описания Ω_q, через внешние переменные, и подстановка этих выражений в ограничения вместо переменных не входит в противоречие с заданными ограничениями на внешние переменные, то добавим к равенству $\Gamma_{pq}(\Gamma^d_{pq})$ с зануленными нелинейными частями ограничения $lp_i \le \psi_i \le up_i, 1 \le i \le p$ и $lq_i \le \phi_i \le uq_i, 1 \le i \le q$. После этого исключаем из получившихся линейных ограничений все переменные, кроме внешних, и получаем искомые ограничения на внешние переменные.

Рассмотрим пример:

SUBROUTINE P(...)
DIMENSION $A_p(N,N,N)$
...
CALL Q(..., $A_p(1,s,m)$, ...)
...
END

SUBROUTINE Q(...,A_q,...)
DIMENSION $A_q(N,t)$
...
END

Первая размерность массивов описана одинаково, поэтому записываем равенство $\psi_1 - 1 = \phi_1 - 1$, то есть $\psi_1 = \phi_1$. Для оставшейся части записываем частичное уравнение ранга 2:

$$\Gamma^2_{pq} : (\psi_2 - 1) + (\psi_3 - 1)N - (s - 1) - (m - 1)N = \phi_2 - 1.$$

Приведя подобные, получаем $\phi_2 = \psi_2 - s + 1 + (\psi_3 - m)N$. Это равенство будет линейным при $\psi_3 = m$. Так как $1 \le \psi_3 \le N$, то это возможно при $1 \le m \le N$. К получившемуся линейному равенству $\phi_2 = \psi_2 - s + 1$ добавляем ограничения $1 \le \phi_2 \le t$ и $1 \le \psi_2 \le N$. После исключения переменных ϕ_2 и ψ_2 получаем неравенства $s \ge 1$ и $t \le N - s + 1$. Итак, равенство Γ^2_{pq} будет линейным при выполнении условий:

$$\begin{cases} 1 \le m \le N \\ 1 \le t \le N - s + 1 \\ 1 \le s \end{cases}$$

Таким образом накладываются дополнительные условия на внешние переменные подпрограммы, выполнение которых гарантирует линейность условия $\Gamma_{pq}(\Gamma^d_{pq})$, а значит, и возможность проведения межпроцедурного анализа.

Нами была собрана статистика по многим реальным программам с целью определить применимость методов межпроцедурного анализа и, в частности, возможность описания входных и выходных данных подпрограммы в терминах фактических параметров. Статистика собиралась по многим параметрам. Некоторые полученные данные для нескольких проектов приведены в таблице на рис. 6. Представлены проекты из самых различных областей, например, FLO52 и MHD взяты из известного пакета Perfect Club, GCMA — программа моделирования долговременных изменений климата, LIU_FTC — программа моделирования устойчивости плазмы в установках управляемого термоядерного синтеза. Данные анализировались для всех вызовов подпрограмм и функций по всем аргументам и всем данным из COMMON-блоков.

	GCMA	ELVEC	K_SPA	MHD	LIU_FTC	FLO52
Число строк	15288	1187	42732	2577	26765	1990
Число подпрограмм	161	11	344	71	404	28
Число вызовов	366	50	3079	147	4890	86
Скаляры	1410	309	12351	6075	617349	803
Массивы	1198	95	5474	3668	126734	170
Совпадение	918	57	4976	3668	126561	79
Анализируемо	120	38	249	0	152	35
Другие случаи	160	0	249	0	21	56
Процент	77%	100%	94%	100%	99%	76%

Рис. 6. *Результаты сбора статистики*

Некоторые пояснения к таблице. В ее столбцах приведены данные для 6 достаточно больших и разнородных проектов. Строки таблицы обозначают следующее: "Число строк" — число строк исходного текста программы; "Число подпрограмм" — количество программных единиц в проекте; "Число вызовов" — общее число проанализированных вызовов в тексте; "Скаляры" — количество скалярных аргументов в вызовах; "Массивы" — количество аргументов-массивов в вызовах; "Совпадение" — количество случаев, когда описание формального параметра-массива в точности совпадает с описанием фактического параметра; "Анализируемо" — количество вхождений аргументов-массивов, не попавших в предыдущий пункт, но не составляющих проблемы при межпроцедурном анализе; "Другие случаи" — вхождения аргументов-массивов, не попавшие в предыдущие два пункта (это не обязательно значит, что это случаи составляют трудность для межпроцедурного анализа, потому что статистика собиралась с использованием относительно грубых эвристик); "Процент" — процент вызовов, все аргументы которых не попали в предыдущий случай (то есть вызовов, заведомо не препятствующих межпроцедурному анализу).

Из таблицы видно, что описываемая техника применима к подавляющему большинству вызовов в реальных программах. Причем дополнительное исследование наиболее сложной для межпроцедурного анализа программы FLO52 показало, что в основной вычислительной части, на которую приходится более 90 % времени исполнения программы, нет ни одного неанализируемого аргумента.

8 Заключение

В данной работе приведен обзор существующих методов межпроцедурного анализа программ, а также предложены некоторые пути их дальнейшего усовершенствования. Приведенная статистика по реальным программам подтверждает разрешимость задач межпроцедурного анализа для большинства приложений. Описанная техника межпроцедурного анализа используется во многих реальных системах, таких как: PIPS [17], FIAT [11], ParaScope [12], Polaris [7], Parafrase-2 [22].

Хочется выразить большую благодарность П.А. Церетели за помощь в сборе статистики, а также Вл.В. Воеводину за внимательное прочтение статьи, многочисленные замечания и предложения.

Литература

[1] В.В. Воеводин *Математические основы параллельных вычислений*// М., 1991.

[2] Вл.В. Воеводин *Статический анализ и вопросы эффективной реализации программ*// Вычислительные процессы и системы (Под. ред. Г.И. Марчука). М., 1992. №9. С. 249-301.

[3] Вл.В. Воеводин *Теория и практика исследования параллелизма последовательных программ*// Программирование. 1992. №3. С. 38–53.

[4] Вл.В. Воеводин *Описание входных и выходных данных фрагментов программ*// Вестник Московского университета. Серия 15. 1997. №1. С. 41–44.

[5] В.А. Евстигнеев, В.А. Серебряков *Методы межпроцедурного анализа (обзор)*// Программирование. 1992. №3. С. 4–15.

[6] V. Balasundaram, K. Kennedy *A Technique for Summarizing Data Access and Its Use in Parallelism Enhancing Transformation*// In Proceedings of the 1989 ACM SIGPLAN Conference on Programming Language Design and Implementation. Vol. 24. №7. pp. 41–53. Portland, Orgen. June 1989.

[7] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger, P. Tu *Parallel programming with Polaris*// Computer. Vol. 29. №12. pp. 78–82. December 1996.

[8] M. Burke, R. Cytron *Interprocedural Dependence Analysis and Parallelisation*// ACM SIGPLAN'86 Symposium on Compiler Construction. Vol. 21. №7. pp. 162–175. June 1986.

[9] D. Callahan, K. Kennedy *Analysis of Interprocedural Side Effects in a Parallel Programming Environment*// Journal of Parallel and Distributed Computing. Vol. 5. pp. 517–550. Oktober 1988.

[10] D. Callahan *A Global Approach to Detection of Parallelism*// PhD thesis. Rice University. Comp. Sci. Tech. Rep. TR87-50. March 1987.

[11] A. Carle, M. W. Hall, J. Mellor-Crummey, R. Rodriguez *FIAT: A Framework for Interprocedural Analysis and Transformation*// Rice University. CRPC Tech. Rep. TR95522-S. March 1995.

[12] D. Callahan, K. D. Cooper, R. T. Hood, K. Kennedy, L. Torczon *ParaScope: A Parallel Programming Environment*// The International Journal Supercomputer. Vol. 2. Nº 4. pp. 84–99. 1988.

[13] S.-E. Choi *An Overview of Compiler Techniques for Interprocedural Array Section Analysis*// University of Washington. Tech. Rep. 95-11-06. November 1995.

[14] B. Creusillet, F. Irigoin *Interprocedural Array Region Analyses*// Eighth International Workshop on Languages and Compilers for Parallel Computing. pp.4-1 to 4-15. Colombus (Ohio), USA. August 1995.

[15] B. Creusillet *IN and OUT Array Region Analyses*// Fifth Workshop on Compilers for Parallel Computers. Malaga, Spain. June 1995.

[16] P. Havlak, K. Kennedy *An Implementation of Interprocedural Bounded Regular Section Analysis*// IEEE Transactions on Parallel and Distributed Systems. Vol. 2. Nº 3. pp. 350–360. July 1991.

[17] F. Irigoin, P. Jouvelot, R. Triolet *Semantical Interprocedural Parallelisation: An Overview of the PIPS Project*// ACM International Conference on Supercomputing, ICS'91. Cologne, Allemagne. June 1991.

[18] Z. Li, P.-C. Yew *Efficient Interprocedural Analysis for Program Parallelisation and Restructuring*// In Proceedings of ACM SIGPLAN Symposium on Parallel Programming: Experience with Applications, Languages and Systems. Vol. 23. Nº 9. pp. 85–99. July 1988.

[19] Z. Li, P.-C. Yew *Interprocedural Analysis and Program Restructuring for Parallel Programs*// CSRD Tech. Rep. 720. January 1988.

[20] Z. Li, P.-C. Yew *Interprocedural Analysis for Parallel Computing*// Proceedings of the 1988 International Conference on Parallel Processing. Vol. 2. pp. 221–228. August 1988.

[21] D. E. Maydan, J. L. Hennessy, M. S. Lam. *Efficient and Exact Data Dependence Analysis*// Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation.

[22] C. D. Polychronopoulos, M. B. Gircar, M. R. Haghighat, C. L. Lee, B. P. Leung, D. A. Shouten *The Structure of Parafrase-2: Advanced Parallelizing Compiler for C and Fortran*// Languages and Compilers for Parallel Computing. Pitman Publ. London. 1990.

[23] W. Pugh. *A Practical Algorithm for Exact Array Dependence Analysis*// Communications of the ACM. Vol. 35, Nº 8. pp. 102-114. August 1992.

[24] S. Richardson, M. Ganapathi *Interprocedural Analysis Useless for Code Optimization*// Stanford University. Tech. Rep. CSL-TR-87-342. November 1987.

[25] R. W. Scheifler *An Analysis of Inline Substitution for a Structured Programming Language*// Communications of the ACM. Vol. 20. Nº 9. September 1977.

[26] D. A. Schouten *An Overview of Interprocedural Analyses Techniques for High Performance Parallelizing Compilers*// Univ. of Illinois at Urbana-Champaign. CSRD Tech. Rep. 1005. May 1990.

[27] P. Tang *Exect Side Effects for Interprocedural Dependence Analysis*// Australian National University. Tech. Rep. TR-CS-92-15. November 1992.

[28] R. Triolet, F. Irigoin, P. Feautrier *Direct Parallelism of Call Statements*// In Proceedings of the ACM SIGPLAN'86 Symposium on Compiler Construction. SIGPLAN Notices. Vol. 21. Nº 7. pp. 176–185. June 1986.