УДК 681.3

Д. Б. Борзов, С. А. Дюбрюкс, В. С. Титов

МЕТОД ОБЪЕДИНЕНИЯ И РАЗДЕЛЕНИЯ ЦИКЛИЧЕСКИХ УЧАСТКОВ ПОСЛЕДОВАТЕЛЬНЫХ НАСЛЕДУЕМЫХ ПРОГРАММ

Предложен метод объединения/разделения циклических участков последовательных наследуемых программ, позволяющий объединять тела циклов и получать более длинные линейные участки, подлежащие последующему распараллеливанию. На основе данных о внутренней структуре программы проанализированы возможности ее эквивалентного преобразования к виду, позволяющему выполнять параллельно часть задач, назначенных ранее для последовательного выполнения.

Ключевые слова: *распараллеливание, программа, объединение, разделение, цикл, линейка, участок, метод.*

В настоящее время все большее распространение получает использование большого количества процессоров (до нескольких сотен) для решения поставленных задач [1]. В то же время значительная часть задач остается последовательными [2], в связи с этим для повышения производительности необходимо назначать на разные процессоры отдельные части одной задачи. Очевидно, что различные подзадачи при этом должны оставаться независимыми без потери логики исходного алгоритма, в связи с чем необходимо выявлять независимые подалгоритмы внутри последовательных программ. Значительная часть современных алгоритмов состоит из циклических участков [3], на обработку которых используется ощутимая часть ресурсов вычислительной системы.

В настоящей статье предложен метод объединения и разделения циклических участков последовательных исполняемых программ, развиваются идеи, представленные в работах [4, 5].

Предлагаемый метод позволяет объединять циклы с целью получения более длинных линейных участков для их последующего распараллеливания, а также сокращения количества операций приращения счетчиков циклов и проверок условий выхода из них. Метод основан на объединении тел смежных циклов последовательной наследуемой программы, операторы которых имеют одинаковый уровень вложенности.

Введем исходные обозначения.

1. Множество операторов задается массивом $M = \left\| m_{ij} \right\|_{N\times 5}$, где $i = N$ — порядковые номера операторов, N — общее число операторов.

2. $\forall j = 1,\ E_{m_{i1}} = N_{\mathrm{op}_N}$, где N_{op} — порядковые номера элементов массива M, $i = \overline{1,\ N}$.

3. $\forall j = 2,\ E_{m_{i2}} = V_N$, где V — уровень вложенности i-го оператора, $i = \overline{1,\ N}$.

4. $\forall j = 3,\ E_{m_{i3}} = C_{\mathrm{beg}_N}$, где C_{beg} — начальное значение счетчика цикла, в который вложен i-й оператор, $i = \overline{1,\ N}$.

5. $\forall j = 4,\ E_{m_{i4}} = C_{\mathrm{end}_N}$, где C_{end} — конечное значение счетчика цикла, в который вложен i-й оператор, $i = \overline{1,\ N}$.

6. $\forall j = 5,\ E_{m_{i5}} = C_{n_N}$, где C_n — порядковые номера циклов, $i = \overline{1,\ N}$.

На первом этапе при использовании метода осуществляются поиск и выделение на описанном массивом M фрагменте программы участка из циклов, подлежащих объедине-

нию. Операторы, составляющие тела этих циклов, должны иметь одинаковый уровень вложенности.

Рассмотрим более подробно данный этап на примере следующего алгоритма. Счетчик C_t используется для подсчета числа операторов, составляющих выделяемый участок программы, переменная `Level` используется для перебора возможных значений уровня вложенности от первого до максимального для всего фрагмента программы значения `LevMax`, переменная `Copy2` используется для копирования нижней границы искомого участка, i и k являются вспомогательными переменными.

1. `Level=1.`
2. i`=1.`
3. Если $V(i)$=Level, то C_t=1; $A(C_t)=M(i)$; LowSide=i (иначе п. 9).
4. $k=i+1$.
5. Если $V(k)$=Level, то $C_t=C_t+1$; $A(C_t)=M(k)$ (иначе п. 8).
6. $k=k+1$.
7. Если $k=N$, то п. 8 (иначе п. 5).
8. Если $C_t>1$, то Copy2= LowSide; переход на 2 этап (иначе п. 10).
9. $i=i+1$.
10. Если $i=N$, то п. 11 (иначе п. 3).
11. `Level=Level+1.`
12. Если Level>LevMax, то конец (иначе п. 2).

Данные о выделенных участках заносятся во вспомогательный массив $A = \|a_{ij}\|_{N\times 5}$, имеющий одинаковую структуру с массивом M.

Опишем работу приведенного алгоритма на следующем примере. Пусть дан массив M с числом операторов $N = 9$, где максимальное значение уровня вложенности V равно 2 (табл. 1).

Таблица 1

N_{op}	V	C_{beg}	C_{end}	C_n
1	0	1	12	0
2	1	1	15	1
3	1	1	20	2
4	1	1	20	2
5	1	1	12	3
6	0	13	15	0
7	1	13	15	4
8	2	16	20	5
9	0	16	20	0

В п. 1 алгоритма переменной `Level` присваивается значение 1, т.е. выполняются поиск и выделение групп операторов первого уровня вложенности. В п. 2 счетчику i операторов присваивается значение 1. В результате значение первого элемента вектора V меньше значения переменной `Level`, поэтому содержимое счетчика i увеличивается на единицу (i=2). Так как значение второго элемента вектора V равно значению переменной `Level`, то счетчику C_t операторов образуемой группы присваивается значение 1, а нижней границе `LowSide` будущего участка — значение i: `LowSide`=2. Элемент 2 массива M копируется на место элемента 1 вспомогательного массива A. В п. 4 инициализируется вспомогательная переменная k=3. Значение третьего элемента вектора V равно значению переменной `Level`, поэтому содержание счетчика C_t увеличивается на единицу: C_t =2. В то же время элемент 3 массива M перемещается (копируется) на место второго элемента вспомогательного массива A. В п. 6 k=4. Так как k меньше общего числа операторов, осуществляется переход к п. 5. Значение четвертого элемента вектора V равно значению переменной `Level`, следовательно, C_t увеличивается на единицу: C_t =3. При этом элемент 4 массива M копируется на место элемента 3 вспомогательного массива A и k=5. Так как k<9, то осуществляется переход к п. 5. Так как значение пятого элемента вектора V равно значению переменной `Level`, содержимое счетчика C_t увеличивается

на единицу: C_t =4, а элемент 5 массива M копируется на место элемента 4 вспомогательного массива A, при этом и увеличивается на единицу переменная k: k=6. Значение шестого элемента вектора V не равно значению переменной `Level`, следовательно, осуществляется переход к п. 8. Так как C_t>1, то искомый участок выбран, значение его нижней границы (2) сохраняется в переменной `Copy2`, осуществляется переход к следующему этапу.

На втором этапе происходит объединение всех циклов выделенного участка в один с минимальным значением счетчика C_{beg} с последующим разделением оставшихся участков, затем — объединение по новому минимальному значению C_{beg} среди счетчиков конца цикла с дальнейшим разделением и т.д., пока количество циклов объединения/разделения не станет равным числу циклов на участке, уменьшенному на единицу.

Исходными данными этапа 2 являются массив $A = \|a_{ij}\|_{N\times 5}$ (табл. 2), значение C_t счетчика количества операторов выделенного на первом этапе участка, суммарное количество циклов этого участка `Max`N_c, и его нижняя граница `LowSide`.

Таблица 2

N_{op}	V	C_{beg}	C_{end}	C_n
2	1	1	15	1
3	1	1	20	2
4	1	1	20	2
5	1	1	12	3

Этап состоит из многоитерационной процедуры поиска минимума среди элементов C_{end} с последующей модификацией элементов C_{beg} и C_n при объединении циклов участка и копирования информации, характеризующей операторы, при разделении циклов. Процесс представляется следующим алгоритмом, в котором `HighSide` — верхняя граница участка, x — счетчик числа итераций объединения/разделения, `Min` — переменная для поиска минимума среди конечных значений счетчиков циклов, `Copy1` — переменная для копирования значений границ участка, z — вспомогательная переменная:

1. `HighSide=`C_t`; LowSide=1.`
2. x`=1.`
3. Если $x\neq$`1, то HighSide=LowSide+1.`
4. `Min=`C_{end}`(LowSide).`
5. z`= LowSide+1.`
6. Если $C_{\text{end}}(z)$`=Min then Min=` C_{end}`(z).`
7. $z=z$`+1.`
8. `Если z>HighSide, то п. 9 (иначе п. 6).`
9. `Copy1= HighSide.`
10. z`= LowSide.`
11. Если $C_{\text{end}}(z)$`>Min, то` $C_{\text{end}}(z)$`=Min,` $C_{\text{beg}}(z+C_t)$`=Min+1;`
    N_{op}`(z+` C_t`)=` $N_{\text{op}}(z)$`+` C_t`;` $C_n(z)=x$.
12. C_n`(z+` C_t`)=` $C_n(z)$`; Copy1= Copy1+1.`
13. $z=z$`+1.`
14. Если z`>HighSide, то п. 14 (иначе п. 11).`
15. `LowSide= HighSide+1; HighSide=Copy1.`
16. $x=x$`+1.`
17. Если x`=` Max N_c`-1, то переход на этап 3 (иначе п. 4).`

Рассмотрим работу данного алгоритма на примере выходных данных первого этапа, которые состоят из массива A (табл. 2), нижней границы `LowSide`=2, суммарного количества циклов участка `Max`N_c=3 и длины участка, равной значению счетчика C_t =4.

В п. 1 значению начальной верхней границы участка присваивается значение `HighSide`= C_t =4 и нижней границы — `LowSide`=1, т.е. на первой итерации обработка ведется среди элементов $a_{1,1}$—$a_{4,1}$. В п. 2 счетчику x числа итераций присваивается начальное значение 1. На первой итерации модификация границ участка поиска минимума не производится,

за минимум принимается значение $C_{end}(l)=15$, соответствующее первому элементу массива A. Значение нижней границы участка, увеличенное на единицу, присваивается счетчику z перебора участка: $z=2$. Значение $C_{end}(2)$ больше минимума, поэтому z увеличивается на единицу: $z=3$. Так как значение z меньше верхней границы, элемент вектора $C_{end}(3)$ сравнивается с минимумом. Значение $C_{end}(3)$ больше минимума, поэтому содержание счетчика увеличивается и становится равным четырем. Так как значение z равно верхней границе, элемент вектора $C_{end}(4)$ сравнивается с минимумом. $C_{end}(4)$ больше минимума, следовательно, минимуму присваивается значение $C_{end}(4)=12$. Далее $z = 5$ — что больше верхней границы, следовательно, осуществляется переход к п. 9 и копируется значение верхней границы `HighSide` в переменную `Copyl`. В п. 10 счетчику z нижней границы перебора участка присваивается значение 1. Так как $C_{end}(l)$ больше минимума, то в п. 11 минимуму присваивается значение $C_{end}(l)$, а элемент массива $A(1)$ копируется в элемент $A(5)$, при этом происходит модификация значений $C_{end}(l)$=`Min`=12; $C_{beg}(5)$=`Min`+l=13, $C_n(1)=1$ и $C_n(5)=4$, а также увеличивается значение копии верхней границы: `Copy1`=5. В п. 12 осуществляется приращение значения z: $z=2$; Так как $z<4$, то происходит переход к п. 11. Поскольку $C_{end}(2)$ больше минимума, то в п. 11 элементу вектора $C_{end}(2)$ присваивается значение минимума, при этом элемент массива $A(2)$ копируется в элемент $A(6)$, осуществляются модификация значений $C_{end}(l)$=`Min`=12, $C_{beg}(5)$=`Min`+l=13, $C_n(2)=1$, $C_n(6)=5$ и увеличение значения копии нижней границы `Copy1`=`Copy1`+1=6. Далее значение z увеличивается на единицу: $z=3$. Так как $z<4$, то осуществляется переход к п. 11. Элемент вектора $C_{end}(3)$ больше минимума, поэтому в п. 11 ему присваивается значение минимума, а элемент массива $A(3)$ копируется в элемент $A(7)$, при этом происходит модификация следующих значений: $C_{end}(l)$=`Min`=12, $C_{beg}(5)$=`Min`+l=13, $C_n(3)=1$, $C_n(7)=6$. Затем происходит увеличение на единицу значения копии нижней границы и счетчика z: `Copy1`=7, $z=4$. Так как $z=4$, то происходит переход к п. 11. Значение $C_{end}(4)$ равно минимуму, следовательно, осуществляются переход к п. 13 и приращение на единицу значения z: $z=5$. Так как $z>4$, то переходим к п. 14 и модифицируем границы участка поиска минимума: `LowSide`=`HighSide`+l=5; `HighSide`=`Copyl`=7. В п. 15 содержимое счетчика итераций x увеличивается, поиск минимума и копирование элементов массива A, меньших минимума, повторяются. В данном примере необходимо выполнить три итерации, после чего массив A примет вид, представленный в табл. 3. После выполнения последней итерации управление передается третьему этапу.

Таблица 3

N_{op}	V	C_{beg}	C_{end}	C_n
2	1	1	12	1
3	1	1	12	1
4	1	1	12	1
5	1	1	12	1
6	1	13	15	2
7	1	13	15	2
8	1	13	15	2
9	1	16	20	3
10	1	16	20	3

На третьем этапе происходит вставка в исходный массив M вспомогательного массива A с помощью нижеприведенного алгоритма, в котором значения переменных `Copy1` и `Copy2` заимствуются из предыдущих этапов, s и h — вспомогательные переменные для индексации массивов A и M.

1. h=Copy1 – Copy2.
2. $s=N$.
3. $M(s+h)=M(s)$.
4. $s=s$-l.
5. Если s<Copy2, то п. 6 (иначе п. 3).

6. $s=i$.
7. $M(s)=A(s)$.
8. $s=s+1$.
9. Если $s>$Copy2, то $N=N+h$; $i=$Copy1+1; передача управления на этап 1 (иначе п. 7).

В приведенном алгоритме происходит нахождение длины h отрезка массива M, который необходимо освободить для вставки массива A (п. 1 алгоритма), затем перезапись элементов массива M в память на расстояние h (пп. 2—5 алгоритма), а также вставка элементов массива A в массив M (пп. 6—9 алгоритма), что при копировании всех данных, характеризующих операторы, приводит к эквивалентному преобразованию программы. В п. 9 число операторов программы увеличивается на h, счетчик операторов — на Copy1+1, и управление передается п. 10 первого этапа. Аналогичным образом осуществляются обработка оставшихся элементов вектора V и поиск участков, содержащих операторы уровня вложенности 1, после чего содержимое счетчика x уровня вложенности на первом этапе увеличивается на 1. Так как при $x=2$ участков для объединения/разделения в рассматриваемом фрагменте программы нет, массив M примет окончательный вид (табл. 4).

Таблица 4

N_{op}	V	C_{beg}	C_{end}	C_n
1	0	1	12	0
2	1	1	12	1
3	1	1	12	1
4	1	1	12	1
5	1	1	12	1
6	1	13	15	2
7	1	13	15	2
8	1	13	15	2
9	1	16	20	3
10	1	16	20	3
11	0	13	15	0
12	1	13	15	4
13	2	16	20	5
14	0	16	20	0

Предложенный метод позволяет сократить число приращений счетчиков и проверок выполнения условий окончания циклов, а также выполнять некоторые циклы параллельно на определенном пространстве итераций, что ведет к повышению производительности многопроцессорных систем, обрабатывающих алгоритм. Данный метод имеет ряд недостатков, одним из которых является возможность его применения только к циклам, следующим друг за другом. Поэтому возникает проблема предварительной перегруппировки операторов программы, исследование которой планируется в дальнейшем.

СПИСОК ЛИТЕРАТУРЫ

1. *Трахтенгерц Э. А.* Введение в теорию анализа и распараллеливания программ ЭВМ в процессе трансляции. М.: Наука, 1981. С. 184—187.

2. *Воеводин В. В.* Математические модели и методы в параллельных системах. М.: Наука, 1986.

3. *Дюбрюкс С. А., Борзов Д. Б., Титов В. С.* Выявление параллелизма внутри линейных участков последовательных программ со связями по управлению // Сб. тр. XIV Междунар. науч.-технич. конф. „Машиностроение и техносфера XXI века“. Т. 2. Донецк, 2007. С. 26—30.

4. *Борзов Д. Б., Дюбрюкс С. А., Титов В. С.* Метод выявления параллелизма внутри линейных участков последовательных программ и его аппаратная реализация // Изв. вузов. Приборостроение. 2008. Т. 51, № 2. С. 34—38.

5. *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. СПб: БХВ-Петербург, 2002. 608 с.

Сведения об авторах

Дмитрий Борисович Борзов — канд. техн. наук, доцент; Курский государственный технический университет, кафедра вычислительной техники; E-mail: borzovdb@kursknet.ru

Сергей Александрович Дюбрюкс — аспирант; Курский государственный технический университет, кафедра вычислительной техники; E-mail: serhio5551@yandex.ru

Виталий Семенович Титов — д-р техн. наук, профессор; Курский государственный технический университет, кафедра вычислительной техники; зав. кафедрой; E-mail: titov@vt.kstu.kursk.ru

Рекомендована кафедрой вычислительной техники

Поступила в редакцию 12.09.08 г.