Преобразования последовательных программ при их распараллеливании с помощью системы САПФОР*

Н.А. Катаев, М.С. Клинов, Н.В. Поддерюгина

Институт прикладной математики имени М.В. Келдыша РАН

Есть такие последовательные программы, которые после преобразования некоторых ее фрагментов эффективно отображаются на современные кластеры с помощью автоматически распараллеливающего компилятора системы САПФОР. Такие преобразования не всегда можно сделать автоматически без участия программиста. Использование автоматически распараллеливающего компилятора для этих программ проще написания программ в модели DVM, потому что при преобразовании последовательных программ у этих подходов много общих действий.

1. Введение

Процесс распараллеливания следует понимать не как преобразование последовательной программы в параллельную, а как преобразование последовательной программы в эффективную параллельную. Поэтому и считается, что автоматическое распараллеливание пока недостижимо [1]. Однако процесс распараллеливания можно представить в виде двух процессов: преобразования в рамках последовательной программы и преобразование последовательной программы в эффективную параллельную. В таком случае к выполнению первого процесса можно привлечь программиста, тем самым, с одной стороны снизить уровень автоматизации процесса, но с другой повысить его качество, а второй процесс выполнять полностью автоматически.

Такой подход используется в системе САПФОР [2-4], которая включает в себя диалоговую оболочку и автоматически распараллеливающий компилятор. Система рассчитана на взаимодействие с пользователем, который достаточно хорошо разбирается в последовательной программе, и является в этом смысле программистом. В процессе диалога с системой программист оценивает возможности автоматически распараллеливающего компилятора по распараллеливанию его текущей версии последовательной программы и определяет пути возможных ее преобразований. Когда преобразования над последовательной программой закончены, через запуск автоматически распараллеливающего компилятора он получает параллельную программу. Случай, когда система не может путем автоматического анализа определить свойства некоторых фрагментов последовательной программы, сводится к добавлению программистом этих свойств к программе. Это может быть реализовано через специальные комментарии, вставленные в текст программы, или через подсказки системе в процессе диалога.

2. Преобразования при распараллеливании на кластер

Рассмотрим в этом разделе такие свойства фрагментов последовательных программ, которые требуют их преобразования для получения эффективных параллельных программ. При этом подразумевается, что все преобразования выполняются в рамках последовательной программы, т.е. после преобразований программа остается последовательной и не содержит никаких свойств параллельной программы, указаний по ее генерации, вариантов распределения данных в ней и т.п.

Некоторые преобразования связаны с тем, что система САПФОР направлена на получение программ для кластерных и распределенных систем. В связи с этим она сталкивается с необходимостью выбора вариантов распределения данных между узлами распределенных систем, которые определяют эффективность распараллеливания программы. Поэтому некоторые преобра-

* Работа поддержана грантами Президента РФ МК-6772.2012.9 и НШ-4307.2012.9, грантами РФФИ № 12-01-33003, 12-07-31205 и 12-07-31260.

зования не требуются при автоматическом распараллеливании, например, на мультипроцессоры. При этом многие из приведенных далее преобразований требуются при автоматическом распараллеливании на кластеры при помощи систем, отличных от системы САПФОР, в том числе и при разработке программ с использованием языков параллельного программирования.

Сформулируем для начала набор свойств, которые требуют преобразований последовательных программ, а затем более подробно их рассмотрим:

- Наличие зависимостей в цикле, которые не являются приватными или редукционными:
  - OUTPUT-зависимости,
  - нерегулярные FLOW-зависимости,
  - регулярные FLOW-зависимости;
- Наличие в цикле операторов ввода-вывода;
- Наличие в цикле нескольких входов или нескольких выходов из него;
- Наличие на одном витке цикла присваиваний в разные элементы одного массива;
- Наличие нескольких циклов с разным соответствием на присваивание элементов разных массивов;
- Отсутствие тесно-вложенных циклов во фрагментах программы, для которых их можно организовать;
- Необходимость коммуникаций (обмены значениями редукционных переменных, теневых граней, буферов с данными разных процессов, синхронизации).

Поясним случай наличия OUTPUT-зависимостей в цикле, которые не являются приватными или редукционными, на примере кода на языке Фортран и приведем пример их устранения. Для этого надо сформулировать несколько определений.

Под OUTPUT-зависимостью понимается информационная зависимость между витками цикла, когда имеет место запись на разных витках в одну ячейку памяти.

Под приватной переменной для цикла (в том числе под приватным массивом) понимается такая переменная, для которой на каждом витке цикла сначала выполняется присваивание в переменную (в элементы массива), затем переменная (элементы массива) может быть использована на этом витке, и за пределами цикла также сначала производится присваивание в переменную (в элементы массива), затем использование. Если в цикле есть приватная переменная, то будем говорить, что в цикле есть приватная зависимость по этой переменной.

Под редукционной переменной и соответственно редукционной зависимостью в цикле понимается неприватная переменная, при которой можно преобразовать цикл (при этом результаты его выполнения останутся прежними с точностью до перестановки слагаемых в сумме или множителей в произведении) таким образом:

- завести на каждом витке свою копию этой переменной,
- присвоить ей начальное значения (0 – для суммы, 1 – для произведение, отрицательный максимум при поиске максимума и т.п.),
- после выполнения всех витков скорректировать значения всех копий по единой схеме (например, сумма локальных сумм, максимум среди локальных максимумов). Для всех элементов редукционного массива (когда каждый элемент массива является редукционной переменной) операция коррекции копии является одинаковой.

В данном случае имеет место цикл с шагом 1 (по умолчанию для Фортрана) и выполняется присваивание на витке в соседние элементы одного массива ro1, аналогично и для массива E1. В результате на соседних витках будет выполняться присваивания в один элемент массива (в одну ячейку памяти). Одним из эффективных способов организации параллельного выполнения цикла является выполнение каждого витка целиком некоторым процессом. Это позволяет перед циклом одной операцией распределить витки (и группы витков) по процессам. В противном случае требуется преобразование цикла к такому виду, который подразумевает выполнение каждого витка целиком на процессе, либо преобразование, которое распределяет вычисления по процессам другими способами, и которое представляется более сложным относительно преобразования цикла к нужному виду. Распределение витков цикла целиком является широко распространенным методом среди высокоуровневых языков, использовать которые для автоматического распараллеливания является предпочтительным для уменьшения объема преобрзований кода программы, необходимых для организации параллельного выполнения.

Наличие OUTPUT-зависимостей в цикле	Пример изменения
…	dimension tmp1(0:nx1,0:ny1)
do j = 1,ny	dimension tmp2(0:nx1,0:ny1)
…	…
do i = 0,nx	do j = 1,ny
…	…
Frox = …	do i = 0,nx
FEx = …	…
ro1(i,j) = ro1(i,j) + Frox	Frox = …
ro1(i+1,j) = ro1(i+1,j) - Frox	FEx = …
E1(i,j) = E1(i,j) + FEx	tmp1(i,j) = Frox
E1(i+1,j) = E1(i+1,j) - FEx	tmp2(i,j) = FEx
enddo	ro1(i,j) = ro1(i,j) + Frox
enddo	E1(i,j) = E1(i,j) + FEx
…	enddo
	enddo
	do j = 1,ny
	do i = 0,nx
	ro1(i+1,j)= ro1(i+1,j) - tmp1(i,j)
	E1(i+1,j)= E1(i+1,j) - tmp2(i,j)
	enddo
	enddo
	…

Рис. 1. Пример устранения OUTPUT-зависимостей в цикле

В случае организации параллельного выполнения цикла в таком виде, что каждый процесс выполняет несколько витков цикла целиком, наличие в цикле OUTPUT-зависимости, которая не является приватной или редукционной, делает невозможным параллельное выполнения цикла. Это требует синхронизации между витками, которая может привести к тому, что некоторый процесс не сможет начать свою работу, пока не закончат свою работу все предыдущие процессы.

В приведенном примере программы выполняется накопление суммы в элементах массива: либо прибавляется некоторая величина, либо вычитается. В этом случае цикл можно разбить на два. В первом сохранять необходимые величины во временный массив с тем, чтобы использовать их для последующих операций над этими ячейками. Полученные циклы не будут содержать OUTPUT-зависимостей и допускают параллельное выполнение в процессе нескольких витков целиком.

Второй случай связан с наличием в цикле нерегулярных FLOW-зависимостей, которые не являются приватными или редукционными.

Под FLOW-зависимостью понимается такая информационная зависимость между витками цикла, когда имеет место запись на более раннем витке (по порядку их выполнения) в ячейку памяти (переменную или элемент массива), а на более позднем витке имеет место чтение из этой ячейки.

Под регулярной зависимостью понимается такая зависимость, при которой зависимые витки в гнезде циклов (несколько циклов, когда телом одного цикла является другой цикл) отстают друг от друга не более чем на константы по каждому из циклов в гнезде. Эти константы называются длинами зависимости. Для гнезда циклов их можно оформить в виде вектора длин. Длина зависимости зависит от шага цикла, например:

- если в цикле шаг равен 2, а один виток (i = 3) зависит от предыдущего (i = 1) – то длина будет 1,
- если шаг был 1, то для витков со значением итерационной переменной 3 и 1 длина равна 2,
- при шаге 3, зависимости может и не быть.

Особое внимание заслуживают регулярные зависимости с малой длиной. Если известны размеры цикла, то всегда можно взять в роли константы, ограничивающей длину зависимости, размер цикла, но она будет большой, и параллельная реализация такого цикла не будет эффек-

тивной. Как правило, если цикл не является циклом с регулярной зависимостью с малой длиной, то для него говорят, что его FLOW-зависимость является нерегулярной.

В случае наличия нерегулярных FLOW-зависимостей, которые не являются приватными или редукционными, для реализации параллельного выполнения цикла требуется рассылка посчитанных данных всем следующим процессам (соответствующим их узлам распределенных систем и областям памяти процессов), а следующих процессов много. Таким образом, имеет место рассылка от одного процессора ко многим, которая не всегда может быть просто организована автоматически, например, в языке Fortran-DVM это невозможно имеющимися в нем средствами, а организация ее другими средствами требует более сложной и низкоуровневой переделки кода программы. К тому же время на подобную рассылку может при значительном количестве процессов превысить выигрыш от распараллеливания этого цикла.

Третье свойство связано с наличием регулярных FLOW-зависимостей, которые не являются приватными или редукционными. Для него требуется посылка посчитанных данных нескольким соседним процессам (это зависит от длины зависимости). Это меньшие затраты по отношению к затратам на нерегулярные FLOW-зависимости. К тому же их можно реализовывать средствами высокоуровневых языков и это достаточно простой метод для автоматического распараллеливания, который связан с минимальным преобразованием текста программы и вставкой директив распараллеливания. Подобные средства и соответствующая поддержка выполнения циклов с регулярными зависимостями реализованы, например, в языке Fortran-DVM. При использовании этого языка, DVM-система автоматически рассчитывает целесообразность и параметры конвейеризации таких циклов. В процессе конвейеризации цикла вычисления разбиваются на кванты, после которых выполняется посылка соседним процессам посчитанных данных, необходимых для их дальнейшей работы. Тем самым, более равномерно загружаются процессы, а цикл выполняется быстрее. Но, конечно, при выполнении циклов с регулярными FLOW-зависимостями все равно есть предел масштабируемости, при котором накладные расходы на организацию конвейера превышают выигрыш от распараллеливания. Это следует учитывать при распараллеливании программ.

Наличие в цикле операторов ввода-вывода препятствует параллельному его выполнению. Дело в том, что для корректной работы порядок операторов ввода-вывода должен соответствовать такому их порядку, который был в исходной последовательной программе. Для обеспечения такого порядка требуются дополнительные синхронизации, которые могут существенно замедлить цикл. В некоторых высокоуровневых языках наличие операторов ввода-вывода в теле параллельного цикла недопустимо из-за того, что в языках не обеспечивается корректная работа с ними с точки зрения алгоритма исходной программы. Можно выносить операторы ввода-вывода из вычислительных циклов и выполнять их в нераспараллеливаемых циклах. Такие преобразования не всегда могут быть полностью автоматизированы, и требуют участие программиста.

Помимо этого некоторые высокоуровневые языки, например Fortran-DVM, накладывают дополнительные ограничения, например, на использование распределенных массивов в операторах ввода-вывода. Это может привести к тому, что без внесения изменений в программу, некоторые массивы не могут быть распределены, что в свою очередь может повлечь за собой требование не распределять и другие массивы и т.д. Требование не распределять другие массивы возникает оттого, что в программе могут быть циклы, которые требуют присваивания на одном витке в элементы массивов, некоторые из которых могут быть распределенными, а некоторые размноженными по процессам. Т.е. с одной стороны виток должен быть выполнен на всех процессах, а с другой не на всех – противоречие, которое можно устранить или преобразованием всех таких циклов, или путем размножения этих массивов.

Следующее свойство связано с наличием в цикле нескольких входов (например, при помощи оператора перехода по метке) или нескольких выходов и приводит к тому, что некоторые витки не надо выполнять. А это сложно обеспечить, потому что условие дополнительного выхода из цикла (а основной выход из цикла – это выход по превышению порогового значения итерационной переменной) можно проверить только в динамике, т.е. в процессе выполнения программы. Автоматически откатить работу нескольких витков непросто, и связано с дополнительными накладными расходами, которые снижают эффективность. Аналогично и для дополнительного входа в цикл.

Наличие на одном витке цикла присваиваний в разные элементы одного массива в случае, если они не находятся в одном процессе из-за соответствующего распределения данных, делает невозможным эффективное выполнение витков целиком в рамках процесса.

Наличие присваиваний в разные элементы одного массива на одном витке цикла	Пример изменения

do i = 1,nx
  ro(i,0) = ro(i,1)
  …
  ro(i,ny1) = ro(i,ny)
  …
enddo
 	
do j = 1,ny
  ro(0,j) = ro(1,j)
  …
  ro(nx1,j) = ro(nx,j)
  …
enddo
	do i = 1,nx
	  ro(i,0) = ro(i,1)
	  …
	enddo
	do i = 1,nx
	  ro(i,ny1) = ro(i,ny)
	  …
	enddo
	
	do j = 1,ny
	  ro(0,j) = ro(1,j)
	  …
	enddo
	do j = 1,ny
	  ro(nx1,j) = ro(nx,j)
	  …
	enddo

Рис. 2. Пример изменения циклов с наличием на одном витке присваиваний в разные элементы одного массива

Такие случаи встречаются достаточно часто в последовательных программах: разработчик прикладной программы не думает о ее распараллеливании или не знает о таких особенностях распараллеливания. Он объединяет похожие операции над массивами и экономит на операциях изменения значений итерационной переменной.

Для приведенных в примере циклов не удается найти такое распределение данных, чтобы элементы массива ro(i,0) и ro(i,ny1) одновременно располагались в одном процессе, и было возможным расположение в одном процессе ro(0,j) и ro(nx1,j). Для получения параллельной программы необходимо разбить хотя бы один из этих циклов на два.

А лучше разбить оба цикла на два, чтобы было возможным использовать двумерное распределение данных.

Наличие нескольких циклов с разным соответствием на присваивание элементов разных массивов	Пример их изменения

do i = 1,nx
  ro(i,0) = …
  …
  E(i+1,0) = …
  …
enddo
 	
do i = 1,nx
  ro(i,1) = …
  …
  E(i,1) = …
  …
enddo
	do i = 1,nx
	  ro(i,0) = ...
	  …
	enddo
	do i = 1,nx
	  E(i+1,0) = …
	  …
	enddo
	
	do i = 1,nx
	  ro(i,1) = …
	  …
	enddo
	do i = 1,nx
	  E(i,1) = …
	  …
	enddo

Рис. 3. Пример изменения циклов с разным соответствием на присваивание элементов разных массивов

Разбиение витка является непростой задачей с точки зрения полной автоматизации этого процесса.

Поясним случай с наличием нескольких циклов с разным соответствием на присваивание элементов разных массивов на примере.

Разное соответствие препятствует способу распределению витков, когда каждый процесс выполняет группу витков целиком. Данные, которые он должен модифицировать, должны быть на нем (он должен иметь их копию). Случай, когда любой процесс должен иметь какой-то определенный элемент и следующий, будет приводить к дублированию элементов между процессами, а значит и к дублированию вычислений, что не очень эффективно.

Следующее свойство связано с тесно-вложенными циклами. Это такие циклы, когда телом одного цикла является другой цикл. Такие многомерные циклы можно эффективно распределять между процессами в отличие от случая, когда циклы не тесно вложены, и требуется повторять в цикле операции распределения внутреннего цикла несколько раз. Тесно-вложенные циклы позволяет снизить накладные расходы на организацию параллельного выполнения.

Наличие нетесно-вложенных циклов	Пример их изменения

`do j = 1,ny`
`  hy4 = …`
`  do i = 0,nx`
`  …`
`  enddo`
`enddo`
	`do j = 1,ny`
	`  do i = 0,nx`
	`  hy4 = …`
	`  …`
	`  enddo`
	`enddo`

Рис. 4. Пример формирования тесно-вложенных циклов

С точки зрения последовательного выполнения преобразованной программы она будет выполняться немного медленнее (крайне незначительно) из-за повтора в цикле одного и того же оператора, однако параллельное ее выполнение значительно эффективнее не преобразованного варианта.

Необходимость разного рода коммуникаций (взаимодействий процессов с целью их синхронизации и обмена данными) замедляет параллельное выполнение программы, что может привести к отсутствию дальнейшего ускорения выполнения программы при увеличении числа вычислительных ядер. Коммуникации являются необходимыми для обеспечения корректной работы параллельных вычислений, их нельзя убрать из программы, но можно оптимизировать: объединять в более крупные обмены, выполнять обмены во время вычислений, разбивать обмены и выполнять некоторые раньше.

Среди операций коммуникаций можно выделить следующие виды:

- обмены значениями редукционных переменных,
- обмены теневыми гранями (теневые грани – это специальные буфера, которые расширяют локальную часть данных процесса за счет элементов с соседних процессов, с целью их своевременной загрузки и удобного к ним доступа),
- пересылка буферов с данными процессов,
- синхронизации.

Алгоритмы автоматической оптимизации коммуникаций зависят от возможностей выходного языка параллельного программирования и в ряде случаев могут потребовать дополнительных модификаций программы, без которых эффективная организация коммуникаций невозможна.

3. Заключение

Для ряда последовательных программ со статическими равномерными прямоугольными сетками удается получать хорошие результаты при помощи системы автоматизированного распараллеливания САПФОР [4]. В процессе распараллеливания программист преобразует по определенной дисциплине последовательную программу. Набор таких преобразований являет-

ся достаточно определенным, что позволяет говорить о накоплении опыта и соответствующих навыков.

Выполнять эти преобразования автоматически без участия программиста не всегда представляется возможным из-за большого разнообразия случаев, хотя некоторая автоматизация этого процесса, безусловно, возможна, и соответствующие методы будут в будущем разработаны и исследованы.

Трудоемкость распараллеливания программ при помощи языков параллельного программирования значительно выше трудоемкости преобразований, которые не выходят за пределы текста последовательной программы, в том числе и из-за того, что наиболее сложные преобразования, такие как устранения зависимостей, одинаковы и при ручном распараллеливании, и при использовании системы САПФОР.

Литература

1. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В. Автоматическое распараллеливание последовательных программ для многоядерных кластеров. // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи”, Новороссийск, сентябрь 2010 – М.: Изд-во МГУ, 2010, С. 12-15.
2. Бахтин В.А., Катаев Н.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н. Автоматическое распараллеливание Фортран-программ на кластер с графическими ускорителями // Сборник трудов Международной научной конференции “Параллельные вычислительные технологии” (ПаВТ’2012), Новосибирск, март 2012, С. 373-379.
3. Бахтин В.А., Бородич И.Г., Катаев Н.А., Клинов М.С., Ковалева Н.В., Крюков В.А., Поддерюгина Н.В. Диалог с программистом в системе автоматизации распараллеливания САПФОР. // Супервычисления и математическое моделирование. Труды XIII Международного семинара / Под ред. Р.М. Шагалиева. – Саров: ФГУП “РФЯЦ-ВНИИЭФ”, 2012. – С. 80-84.
4. Бахтин В.А., Бородич И.Г., Катаев Н.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л. Распараллеливание с помощью DVM-системы некоторых приложений гидродинамики для кластеров с графическими процессорами // Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: поиск новых решений”, Новороссийск, сентябрь 2012 – М.: Изд-во МГУ, 2012. – С. 444-450.