**ТАЙЛИНГ**

Разбиение множества операций алгоритма на макрооперации-тайлы; перестановка циклов, распределение циклов. Примеры использования тайлинга. Допустимость тайлинга.

Тайлинг (tiling) – это преобразование алгоритма для получения макроопераций. Получаемые в результате преобразования макрооперации называются зерном вычислений или тайлами. Множество операций алгоритма, составляющих зерно вычислений, выполняется атомарно, как одна единица вычислений; при параллельных вычислениях выполнение операций одного зерна не может прерываться синхронизацией или обменом данными. Разбиение множества операций алгоритма на тайлы позволяет при подходящем задании тайлов организовать параллельные вычислительные процессы и уменьшить накладные расходы на обмен данными при параллельных вычислениях, уменьшить накладные расходы на использование памяти при последовательных вычислениях.

Использование тайлинга приводит, в частности, к известным блочным алгоритмам. Но идея тайлинга не в разбиении на блоки массивов данных, а в разбиении множества операций алгоритма. Вариантов разбиения множества вычислений, вообще говоря, гораздо больше, чем вариантов разбиения массивов данных. Это связано с тем, что размерности итерационных пространств алгоритмов обычно превосходят размерности массивов данных.

При тайлинге каждый цикл разбивается на два цикла: глобальный, параметр которого определяет на данном уровне вложенности порядок вычисления тайлов, и локальный, в котором параметр исходного цикла изменяется в границах одного тайла. Допускается вырожденное разбиение цикла, при котором все итерации относятся к глобальному циклу или все итерации относятся к локальному циклу. Такие циклы будем называть соответственно глобальными не разбиваемыми и локальными не разбиваемыми.

По смыслу тайлинга, для каждого набора операторов локальные циклы должны быть самыми внутренними. Поэтому общие для операторов локальные циклы распределяются между циклами операторов, если только операторы не окружены одинаковым наборов циклов; локальные циклы переставляются с глобальными и становятся самыми внутренними.

Приведем примеры, иллюстрирующие технику тайлинга, и примеры применения тайлинга.

**Пример 1** (разбиение всех циклов; два вида записи алгоритма после тайлинга; перестановка и распределение циклов при тайлинге).Рассмотрим два вложенных цикла:

do *i =* 1, *N*1

do *j =* 1, *N*2

*S*(*i*, *j*)

enddo

enddo

Пусть параметры *r*1 и *r*2 характеризуют размеры тайлов, числа *Q*1 и *Q*2 характеризуют число тайлов; *Q*1, *Q*2 ( обозначает ближайшее «сверху» целое число). Разобьем каждый цикл на глобальный и локальный циклы (пояснения для случая *N*1*=*6, *N*2*=*11 приведены на рисунке, каждой итерации алгоритма поставлена в соответствие точка (*i*,*j*)):

do *igl =* 0, *Q*1–1

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*1)

do *jgl =* 0, *Q*2 –1

do *j =* 1 + *jglr*2, min((*jgl*+1)*r*2, *N*2)

*S*(*i*, *j*)

enddo

enddo

enddo

enddo

Отметим, что порядок вычисления операций после разбиения циклов не изменился.

*r*2 = 3

*r*1= 2

(1, 1)

После перестановки циклов (будем считать это преобразование алгоритма корректным) с параметрами *i* и *jgl* (локальные циклы должны быть самыми внутренними) получим

do *igl =* 0, *Q*1–1

do *jgl =* 0, *Q*2–1

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*1)

do *j =* 1 + *jglr*2, min((*jgl*+1)*r*2, *N*2)

*S*(*i*, *j*)

enddo

enddo

enddo

enddo

Глобальные циклы определяют порядок вычисления тайлов, локальные циклы определяют порядок вычисления итераций исходного алгоритма в границах одного тайла:

do *igl =* 0, *Q*1–1

do *jgl =* 0, *Q*2–1

Tile(*igl, jgl*) \\ макрооперация-тайл включает выполнение *r*1*r*2

итераций (при *igl=Q*1–1 или *jgl=Q*2–1 тайл может

включать меньше итераций

enddo

enddo

Вычисления одного тайла Tile(*igl, jgl*) задаются двумерным циклом

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*1)

do *j =* 1 + *jglr*2, min((*jgl*+1)*r*2, *N*2)

*S*(*i*, *j*)

enddo

enddo

Заметим, что в литературе используется также запись, в которой параметры глобальных циклов изменяются с шагом, определяемым размерами тайла:

do *igl =* 1, *N*1, *r*1

do *jgl =* 1, *N*2, *r*2

do *i = igl*,min(*igl*+*r*1–1, *N*1)

do *j = jgl*, min(*jgl+r*2 –1, *N*2)

*S*(*i*, *j*)

enddo

enddo

enddo

enddo

Рассмотрим случай, когда два оператора окружены не совпадающим набором циклов:

do *i =* 1, *N*

do *j =* 1, *N*

*S*1(*i*, *j*)

enddo

do *j =* 1, *N*

*S*2(*i*, *j*)

enddo

enddo

На рисунке схематично изображены тайлы итераций для первого (ниже) и второго (выше) операторов. Для параметров *i* ось общая, для параметров *j* оси разные.

Разобьем каждый цикл на глобальный и локальный циклы:

do *igl =* 0, *Q*1–1

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*)

do *jgl =* 0, *Q*2 –1

do *j =* 1 + *jglr*2, min((*jgl*+1)*r*2, *N*)

*S*1(*i*, *j*)

enddo

enddo

do *jgl =* 0, *Q*2 –1

do *j =* 1 + *jglr*2, min((*jgl*+1)*r*2, *N*)

*S*2(*i*, *j*)

enddo

enddo

enddo

enddo

Для того чтобы переставить циклы с параметрами *i* и *jgl* (локальные циклы должны быть самыми внутренними) требуется сначала распределить цикл с параметром *i* между разными наборами циклов выполняемых операторов (как и ранее, будем считать все преобразования алгоритма допустимыми):

do *igl =* 0, *Q*1–1

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*)

do *jgl =* 0, *Q*2 –1

do *j =* 1 + *jglr*2, min((*jgl*+1)*r*2, *N*)

*S*1(*i*, *j*)

enddo

enddo

enddo(*i*)

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*)

do *jgl =* 0, *Q*2 –1

do *j =* 1 + *jglr*2, min((*jgl*+1)*r*2, *N*)

*S*2(*i*, *j*)

enddo

enddo

enddo(*i*)

enddo

enddo

После перестановки циклов с параметрами *i* и *jgl* окончательно получим

do *igl =* 0, *Q*1–1

do *jgl =* 0, *Q*2 –1

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*)

do *j =* 1 + *jglr*2, min((*jgl*+1)*r*2, *N*)

*S*1(*i*, *j*)

enddo

enddo(*i*)

enddo(*jgl*)

do *jgl =* 0, *Q*2 –1

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*)

do *j =* 1 + *jglr*2, min((*jgl*+1)*r*2, *N*)

*S*2(*i*, *j*)

enddo

enddo(*i*)

enddo(*jgl*)

enddo

Отметим преобразования циклов, применяемые при тайлинге: каждый цикл заменяется на глобальный и локальный циклы, общие локальные циклы распределяются между операторами (если циклы не являются тесно вложенными), локальные циклы переставляются с глобальными и становятся по отношению к ним внутренними.

**Пример 2** (не разбиваемые глобальные циклы, не разбиваемые локальные циклы).Пусть в рассмотренном первом примере *r*2*=*1; тогда *Q*2*=N*2 (пояснения для случая *N*1*=*7, *N*2*=*11 приведены на рисунке).

*r*2 = 1

*r*1= 3

do *igl =* 0, *Q*1–1

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*1)

do *jgl =* 0, *N*2–1

do *j =* 1 + *jgl* 1, min((*jgl*+1)1, *N*2)

*S*(*i*, *j*)

enddo

enddo

enddo

enddo

После перестановки циклов (если это преобразование алгоритма корректно) получим

do *igl =* 0, *Q*1–1

do *jgl =* 0, *N*2–1

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*1)

*j =* 1 + *jgl*

*S*(*i*, *j*)

enddo

enddo

enddo

Можно поступить проще: считать, что цикл с параметром *j* не разбивается и является глобальным (тогда границы изменения цикла тайлинг оставляет прежними):

do *igl =* 0, *Q*1–1

do *jgl =*1, *N*2

Tile(*igl, jgl*)

enddo

enddo

Вычисления тайла Tile(*igl,jgl*) задаются циклом

do *i =* 1 + *iglr*1,min((+1)*r*1, *N*1)

*j = jgl*

*S*(*i*, *j*)

enddo

Пусть теперь *r*2*=N*2 (рисунок ниже). Тогда (напомним, *Q*2)

do *igl =* 0, *Q*1–1

do *i =* 1 +  *iglr*1,min((*igl*+1) *r*1, *N*1)

do *jgl =* 0, 0

do *j =* 1 + *jglN*2, min((*jgl*+1)*N*2, *N*2)

*S*(*i*, *j*)

enddo

enddo

enddo

enddo

*r*2=*N*2

*r*1=2

*r*2=1

*r*1=*N*1

После упрощения (тайлинг не изменил порядок выполнения операций) имеем

do *igl =* 0, *Q*1–1

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*1)

do *j =* 1, *N*2

*S*(*i*, *j*)

enddo

enddo

enddo

Получили, что цикл с параметром *j* не разбивается и является локальным:

do *igl =* 0, *Q*1–1

Tile(*igl,*0)

enddo

где вычисления тайла Tile(*igl,*0) есть

do *i =* 1 + *iglr*1,min((*igl*+1) *r*1, *N*1)

do *j =* 1, *N*2

*S*(*i*,*j*)

enddo

enddo

Таким образом, если по какой-либо координате размер тайла равен единице, то соответствующий цикл можно не разбивать и считать глобальным, а если по какой-либо координате размер тайла равен числу значений параметра цикла, то соответствующий цикл не разбивается и является локальным.

Заметим, что в случае *r*1*=N*1, *r*2*=*1 тайлинг фактически означает перестановку циклов исходного алгоритма:

do *jgl =* 1, *N*2

Tile(0,*jgl*)

enddo

Вычисления одного тайла Tile(0*,jgl*) есть

do *i =* 1, *N*1

*S*(*i*, *jgl*)

enddo

**Пример 3** ([1], метод окаймления).Рассмотрим двумерный цикл, у которого итерационная область не является прямоугольным параллелепипедом:

do *k =* 1, *N*1

do *i = k+*1, *k+ N*2

*S*(*k*, *i*)

enddo

enddo

После применения тайлинга (пояснения для случая *N*1*=N*2*=*6, *r*1*=r*2*=*2 приведены на рисунке, *Q*1, *Q*2) получим

do *kgl =* 0, *Q*1–1

do *igl =* 0, *Q*2–1

do *k =* 1 + *kglr*1,min((*kgl+*1)*r*1, *N*1)

do *i =* max(1 + *iglr*2, *k+*1), min((*igl*+1)*r*2, *k+ N*2)

*S*(*k*, *i*)

enddo

enddo

enddo

enddo

На рисунке видно, что часть тайлов являются пустыми или неполными; представленный подход, при котором допускаются избыточные области изменения параметров глобальных циклов, называется методом окаймления.

*igl* = 0

*igl* = 1

*igl* = 2

***V***

*igl* = 5

*igl* = 4

*igl* = 3

554

*kgl* = 0

*kgl* = 1

*kgl* = 2

Виды тайлов:

– полный

– неполный

– пустой

**Пример 4** (задание зерна вычислений при организации зернистых параллельных вычислительных процессов). Рассмотрим основную часть алгоритма решения двумерной задачи Дирихле для уравнения Пуассона методом верхней релаксации:

do *m =* 1, *M*

do *i =* 1, *Nx –*1

do *j =* 1,*Ny –*1

*u*(*i*,*j*)*=F*(*u*(*i*–1,*j*),*u*(*i*,*j–*1),*u*(*i*,*j*),*u*(*i*,*j+*1),*u*(*i+*1,*j*)) (1)

enddo

enddo

enddo

Будем считать цикл с параметром *m* глобальным не разбиваемым, а цикл с параметром *j* локальным не разбиваемым. Произведем разбиение цикла уровня вложенности 2: заменим цикл по *i* двумерной циклической конструкцией. Обозначим через *Q*2число итераций в первом новом цикле, а через *r*2 (наибольшее) число значений параметра *i* во втором цикле; параметры *Q*2 и *r*2 связаны соотношениями *Q*2, *r*2. Получим

do *m =* 1, *M*

do *igl =* 0, *Q*2–1

do *i =* 1 + *iglr*2,min((*igl+*1)*r*2, *Nx –*1)

do *j =* 1,*Ny –*1

*u*(*i*,*j*)*=F*(*u*(*i*–1,*j*),*u*(*i*,*j–*1),*u*(*i*,*j*),*u*(*i*,*j+*1),*u*(*i+*1,*j*))

enddo

enddo

enddo

enddo

Пусть *Q*2 число процессов (процессоров, ядер), предназначенных для реализации трехмерного цикла (1). Единый для каждого из *Q*2 процессов псевдокод параллельного алгоритма можно записать следующим образом (Pr процесс с номером *igl*):

Для каждого процесса Pr, 0 *igl**Q*2 –1:

do *m =* 1, *M*

Tile(*m,igl,*0)

*обмен данными (синхронизация)*

enddo

Вычисления тайла Tile(*m,igl,*0) задаются двумерным циклом

do *i =* 1 + *iglr*2,min((*igl+*1)*r*2, *Nx –*1)

do *j =* 1,*Ny –*1

*u*(*i*,*j*) *= F*(*u*(*i*–1,*j*), *u*(*i*,*j–*1), *u*(*i*,*j*), *u*(*i*,*j+*1), *u*(*i+*1,*j*))

enddo

enddo

Каждый процесс *igl* на каждой итерации *m* выполняет операции одноготайла (одного зерна вычислений). Параметры *r*2 и (*Ny –*1) задают размер зерна вычислений: число итераций, принадлежащих одному тайлу (кроме, возможно, граничных тайлов), равно *r*2(*Ny –*1)*.*

**Пример 5** (улучшение локальности при последовательных вычислениях). Рассмотрим один из вариантов алгоритма Гаусса-Жордана решения систем *N* линейных алгебраических уравнений с *N* неизвестными и *P* правыми частями:

do *k=*1*,N*

*l*(1)*=*1*/a*(1*,k*)

do *p=*2*,N*

*l*(*p*)*=* *–a*(*p,k*)

enddo

do *j=k+*1*,N+P*

*u*(*j*)*=a*(1*,j*)*l*(1)

do *i=*2*,N*

*a*(*i–*1*,j*)*=a*(*i,j*)*+l*(*i*)*u*(*j*)

enddo

*a*(*N,j*)*=u*(*j*)

enddo

enddo

Если массивы хранятся в памяти строками, то при изменении параметра цикла *i* данные *a*(*i,j*) извлекаются с большим шагом (при достаточно большом *N*). Тайлинг позволит использовать данные *a*(*i,j*) одного блока кэша, что уменьшит время обращения к памяти.

Заменим цикл по *j* двумерной циклической конструкцией:

do *k=*1*,N*

*l*(1)*=*1*/a*(1*,k*)

do *p =* 2,*N*

*l*(*p*)*=* *–a*(*p,k*)

enddo

do *jgl =* 0, *Q*2–1

do *j =* max(2 *+* *jglr*2 , *k +*1),min(1*+*( *jgl+* 1)*r*2*,* *N + P*)

*u*(*j*)*=a*(1*,j*)*l*(1)

do *i =* 2,*N*

*a*(*i–*1*,j*)*=a*(*i,j*)*+l*(*i*)*u*(*j*)

enddo

*a*(*N,j*)*=u*(*j*)

enddo(*j*)

enddo()

enddo(*k*)

Здесь *r*2 характеризует число значений параметров второго цикла двумерной конструкции, *Q*2. Распределим теперь цикл с параметром *j* между выполняемыми операторами и переставим его с циклом по *i*:

do *k=*1*,N*

*l*(1)*=*1*/a*(1*,k*)

do *p =* 2,*N*

*l*(*p*)*=* *–a*(*p,k*)

enddo

do  *=* 0, *Q*2–1

do *j =* max(2 *+* *jglr*2 , *k +*1),min(1*+*( *jgl+* 1)*r*2*,* *N + P*)

*u*(*j*)*=a*(1*,j*)*l*(1)

enddo

do *i =* 2, *N*

do *j =* max(2*+* *jglr*2 , *k+*1),min(1*+*( *jgl+*1)*r*2*,* *N + P*)

*a*(*i–*1*,j*)*=a*(*i,j*)*+l*(*i*)*u*(*j*)

enddo

enddo

do *j =* max(2 *+* *jglr*2 , *k +*1),min(1*+*( *jgl+* 1)*r*2*,* *N + P*)

*a*(*N,j*)*=u*(*j*)

enddo

enddo()

enddo(*k*)

Теперь при небольших *r*2параметры *j*, *i* каждого из локальных циклов (*i* – параметр не разбиваемого локального цикла) изменяются через небольшое число итераций алгоритма, что позволяет эффективно использовать кэш при хранении элементов массива как строками, так и столбцами.

**Пример 6** (ускорение вычислений на многоядерном процессоре).Основная часть одного из вариантов метода Якоби решения разностной задачи Дирихле (одномерное уравнение) можно задать следующим гнездом циклов:

do *m =* 1, *M*

do *i =* 1, *Nx–*1

*u*(*i*)*=*(*l*(*i–*1)*+l*(*i*)*+l*(*i+*1))/3

enddo

do *i =* 1, *Nx–*1

*l*(*i*)*=u*(*i*)

enddo

enddo

Время выполнения параллельного OpenMP-алгоритма на восьмиядерном процессоре с использованием разбиения на тайлы (при этом требуется предварительное аффинное преобразование циклов) и без использования разбиения отличается почти в 10 раз [2].

**Пример 7** (двухуровневый тайлинг для распараллеливания алгоритмов при программирования общего назначения на графических процессорах [3–5]). Для реализации алгоритма на графическом процессоре множество операций алгоритма должно быть разбито на блоки, а блоки – на потоки (нити) вычислений.

Рассмотрим основную часть алгоритма перемножения матрицы *A* размера *N*1*N*3 и матрицы *B* размера *N*3*N*2:

do *i* = 1, *N*1

do *j* = 1, *N*2

do *k* = 1, *N*3

*c*(*i,j*) *= c*(*i,j*) *+ a*(*i,k*) *b*(*k,j*)

enddo

enddo

enddo

После разбиения всех трех циклов (*r*1, *r*2, *r*3 – параметры, *Q*1, *Q*2, *Q*3) и перестановки циклов получим:

do *igl* = 0, *Q*1–1

do *jgl* = 0, *Q*2–1

do *kgl* = 0, *Q*3–1

do *i* = 1+ *iglr*1, min((*igl*+1)*r*1, *N*1)

do *j =* 1+ *jglr*2, min((*jgl*+1)*r*2, *N*2)

do *k* = 1+ *kglr*3, min((*kgl*+1)*r*3, *N*3)

*c*(*i,j*) *= c*(*i,j*) *+ a*(*i,k*) *b*(*k,j*)

enddo

enddo

enddo

enddo

enddo

enddo

Положим, что один блок вычислений Bl(*igl,jgl,kgl*) составляют вычисления

do *i* = 1+ *iglr*1, min((*igl*+1)*r*1, *N*1)

do *j =* 1+ *jglr*2, min((*jgl*+1)*r*2, *N*2)

do *k* = 1+ *kglr*3, min((*kgl*+1)*r*3, *N*3)

*c*(*i,j*) *= c*(*i,j*) *+ a*(*i,k*) *b*(*k,j*)

enddo

enddo

enddo

Для организации потоков произведем разбиение операций блоков вычислений Bl(*igl,jgl,kgl*) (тайлинг уровня 2).

Обозначим через *Q*1,2, *Q*2,2 число итераций в новых глобальных циклах с параметрами *igl*2*,* *jgl*2 соответственно, а через *r*1,2, *r*2,2 число значений параметров *i,* *j* в локальных циклах второго уровня (*r*1,2, *r*2,2); цикл с параметром *k* зададим локальным не разбиваемым. Вычисления одного блока после разбиения можно записать в виде

do *igl*2= 0, *Q*1,2–1

do *jgl*2= 0, *Q*2,2–1

do *i=*1+*iglr*1+*igl*2*r*1,2,min(*iglr*1+(*igl*2*+*1)*r*1,2, (*igl*+1)*r*1, *N*1)

do *j=*1+*jglr*2+*jgl*2*r*2,2,min(*jglr*2+(*jgl*2*+*1)*r*2,2, (*jgl*+1)*r*2, *N*2)

do *k* = 1+ *kglr*3, min((*kgl*+1)*r*3, *N*3)

*c*(*i,j*) *= c*(*i,j*) *+ a*(*i,k*) *b*(*k,j*)

enddo

enddo

enddo

enddo

enddo

Будем считать, что параметры *Q*1,2, *Q*2,2задают число потоков (*Q*1,2*×Q*2,2) в блоке. Один поток вычислений Thr(*igl*2*,jgl*2) блока Bl(*igl,jgl,kgl*) составляют вычисления

do *i=*1+*iglr*1+*igl*2*r*1,2,min(*iglr*1+(*igl*2*+*1)*r*1,2, (*igl*+1)*r*1, *N*1)

do *j=*1+*jglr*2+*jgl*2*r*2,2,min(*jglr*2+(*jgl*2*+*1)*r*2,2, (*jgl*+1)*r*2, *N*2)

do *k* = 1+ *kglr*3, min((*kgl*+1)*r*3, *N*3)

*c*(*i,j*) *= c*(*i,j*) *+ a*(*i,k*) *b*(*k,j*)

enddo

enddo

enddo

Таким образом, в результате двухуровневого тайлинга получили алгоритм, множество операций которого разбито на блоки и потоки (нити) вычислений. Блоки вычислений могут выполняться независимо для каждого *kgl*:

do *kgl* = 0, *Q*3–1

dopar *jgl* = 0, *Q*2–1

dopar *igl* = 0, *Q*1–1

Bl(*igl,jgl,kgl*)*·*

enddo

enddopar

enddopar

Вычисления одного блока Bl(*igl,jgl,kgl*) есть параллельно выполняемые потоки:

dopar *igl*2= 0, *Q*1,2–1

dopar *jgl*2= 0, *Q*2,2 –1

Thr(*igl*2*,jgl*2)

enddopar

enddopar

**Формальная техника тайлинга** [6–8]. Пусть в гнезде циклов имеется Θ наборов выполняемых операторов таких, что в окружении каждого набора есть хотя бы один разбиваемый цикл. Под набором операторов будем понимать один или несколько операторов, окруженных одним и тем же множеством циклов. Наборы операторов линейно упорядочены расположением их в записи алгоритма; операторы тоже линейно упорядочены расположением в записи алгоритма. Обозначим: *V*θ, 1*≤*θ*≤*Θ, – области изменения параметров циклов, окружающих наборы операторов, *n*θ – размерность области *V*θ, число циклов, окружающих θ-й набор операторов.

Следующие величины и множества используются для формализации тайлинга.

,  – предельные значения изменения параметра цикла уровня вложенности  для -го набора операторов;  и  являются константами, которые могут зависеть от внешних переменных; если два набора операторов имеют общий цикл с параметром , то , ;

 – заданные натуральные числа, определяющие размеры тайла;  обозначает число значений параметра , приходящихся на один тайл θ-го набора операторов;  может принимать фиксированное значение в пределах от 1 до  включительно, где ; если *=*1, то цикл с параметром  является глобальным не разбиваемым; если , то цикл с параметром  является локальным не разбиваемым; если два набора операторов имеют общий цикл с параметром , то ;

, , – число частей, на которые при формировании тайлов разбивается область значений параметра  цикла, окружающего -й набор операторов;

 – области изменения параметров глобальных, т.е. уровня тайлов, циклов;

,, , – области изменения параметров локальных (уровня операций тайлов) циклов при фиксированных значениях параметров глобальных циклов. Если цикл является глобальными не разбиваемыми, то условимся, что границы изменения этого цикла тайлинг оставляет прежними.

Если , то имеют место равенства

, (2)

, (3)

где используются обозначения ближайшего «снизу» целого числа и ближайшего «сверху» целого числа. Действительно, так как , то из системы двух неравенств  и  получим . Так как  является целым числом, то *=*.

Множество операций, выполняемых на итерациях множества , будем также обозначать . Множества  называются тайлами  или тайлами . Тайлы  будем называть тайлами -го типа.

Опишем структуру многомерного цикла после применения к нему преобразования тайлинга. Параметры циклов  изменяются в соответствии с неравенствами , для каждого набора операторов имеется столько локальных циклов, сколько  превосходят единицу. Соответственно этому описанию генерируется код.

Отметим, что некоторые множества  могут быть пустыми, так как применяется метод окаймления.

**Допустимость тайлинга.** Установим необходимые и достаточные условия, при выполнении которых преобразование тайлинга является корректным.

Пусть имеется некоторая зависимость *S*α(*I*)*S*β(*J*). Обозначим через  и  номера наборов операторов, которым принадлежат *S*α и *S*β соответственно, а через *c*α,β множество общих циклов в окружении операторов *S*α и *S*β. Отметим очевидное: если некоторый цикл является общим для операторов, то общим является и любой более внешний, в том числе и самый внешний, цикл. Если общих циклов не существует, то положим *c*α,β=0.

**Теорема 1.** *Пусть существует хотя бы одна такая зависимость S*α(*I*)*S*β(*J*)*,* *,* *, что c*α,β≠0*. Тайлинг является допустимым тогда и только тогда, когда для любой такой зависимости выполняются следующие условия:*

, (4)

*if*  *then* . (5)

*При отсутствии указанных зависимостей тайлинг допустим.*

Действительно, при тайлинге может меняться порядок выполнения операций, зависящих от параметров общих циклов. Рассматриваемый тайлинг не изменяет порядок выполнения операций в пределах любого фиксированного тайла. Поэтому необходимо и достаточно установить корректность тайлинга на глобальном уровне (уровне тайлов). Условие (4) устанавливает сохранение зависимостей между итерациями общих глобальных циклов. Условие (5) является необходимым и достаточным для сохранения зависимостей в случае совпадения параметров общих глобальных циклов. В случае ,  зависимые операции принадлежат одному тайлу, поэтому тайлинг не нарушает зависимостей.

Заметим, что условие (4) может выполняться при одних размерах тайлов, но не выполняться при других размерах.

Ниже приведены некоторые иллюстрации, демонстрирующие необходимость и достаточность условий (4) и (5). Отметим, что один вектор зависимостей на уровне операций алгоритма может порождать несколько глобальных векторов зависимостей на уровне тайлов.

do *i =* 1, *N*

do *j =* 1, *N*

*S*1(*i*, *j*)

enddo

do *j =* 1, *N*

*S*2(*i*, *j*)

enddo

enddo

Рассмотрим непосредственно следующий из теоремы 1 способ проверки корректности тайлинга.

Для каждой пары итераций *I* и *J*, порождающих зависимость между операциями *S*α(*I*)*S*β(*J*):

* определить общие циклы;
* если *c*α,β=0, то зависимость не нарушает корректности тайлинга (ввиду );
* если *c*α,β≠0, то определить первые *c*α,β глобальные координаты итераций *I* и *J* (, );
* если *c*α,β≠0, то проверить справедливость условия ;
* если *c*α,β≠0 и , то проверить справедливость условия  (или, что равносильно, проверить справедливость ).

[При определении глобальных координат: если пользоваться имеющейся на данный момент программой, то считать, чтоне разбиваемый цикл является глобальным, если не существует внешнего по отношению к нему разбиваемого цикла, и является локальным, если существует внешний по отношению к нему разбиваемый цикл.]

Приведем известные удобные для практического использования достаточные условия допустимости тайлинга. Выполнение этих условий гарантирует корректность тайлинга для тайлов любого размера.

**Теорема 2.**  *Пусть существует хотя бы одна такая зависимость* *, что c*α,β≠0*. Тайлинг является допустимым, если для любой такой зависимости выполняются следующие условия:*

, , (6)

. (7)

*При отсутствии указанных зависимостей тайлинг допустим.*

**Доказательство.** Покажем, что справедливость условий теоремы 2 гарантирует справедливость условий теоремы 1.

Отметим сразу, что для  условие (6)  выполняется всегда (следует из определения зависимости операций).

Если выполняются условия (6), то из равенств (2) (или (3)) следует справедливость условия (4)  (для глобальных не разбиваемых циклов , , для локальных не разбиваемых циклов , ). Если выполняется условия (7) , то выполняется условия (5) .

Таким образом, условия (6) и (7) являются достаточными для выполнения условий (4) и (5). 

При тайлинге, с точки зрения преобразования циклов, каждый разбиваемый цикл заменяется на глобальный и локальный циклы, общие локальные циклы распределяются между циклами каждого набора операторов, локальные циклы переставляются с глобальными и становятся по отношению к ним внутренними. Условия (6) являются достаточными для переставляемости указанных циклов, а условия (7) достаточны для распределения циклов.

[Привести примеры (см. рисунки выше), когда условия (6) или (7) не выполняются, но тайлинг корректен.]

[В лекции мы понимаем тайлинг как строго определенное преобразование циклов. Можно рассматривать тайлинг в более широком смысле (см. задачи тайлинга в р. 3.5 Курса лекций).]

[Генерация кода: проблема получения параметрического тайлинга (в PLUTO не параметрический тайлинг, в CLooG - параметрический); GenTile, PrimeTile, DynTile]

**Влияние избыточных вычислений на время реализации зернистых версий алгоритма Гаусса-Жордана.** Еще раз рассмотрим зернистый алгоритм, полученный с помощью разбиения цикла *j*:

do *k=*1*,N*

*l*(1)*=*1*/a*(1*,k*)

do *p =* 2,*N*

*l*(*p*)*=* *–a*(*p,k*)

enddo

do *jgl =* 0, *Q*2–1

do *j =* max(2 *+* *jglr*2 , *k +*1),min(1*+*( *jgl+* 1)*r*2*,* *N + P*)

*u*(*j*)*=a*(1*,j*)*l*(1)

enddo

do *i =* 2, *N*

do *j =* max(2 *+* *jglr*2 , *k +*1),min(1*+*( *jgl+* 1)*r*2*,* *N + P*)

*a*(*i–*1*,j*)*=a*(*i,j*)*+l*(*i*)*u*(*j*)

enddo

enddo

do *j =* max(2 *+* *jglr*2 , *k +*1),min(1*+*( *jgl+* 1)*r*2*,* *N + P*)

*a*(*N,j*)*=u*(*j*)

enddo

enddo(*jgl*)

enddo(*k*)

Метод окаймления в данном примере порождает так называемые пустые тайлы: нижняя граница изменения цикла с параметром *j* при некоторых *jgl* превосходит верхнюю границу. Можно избавиться от пустых тайлов: вместо оператора

do *jgl =* 0, *Q*2–1

где *Q*2, записать оператор

do *jgl =* –1, –1.

Назовем полученный алгоритм вариантом 1 алгоритма Гаусса-Жордана без избыточных вычислений.

Вариант 2 алгоритма Гаусса-Жордана без избыточных вычислений можно получить, если вместо оператора

do *jgl =* 0, –1

и оператора

do *j =* max(2 *+* *jglr*2 , *k +*1),min(1*+*( *jgl+* 1)*r*2*,* *N + P*)

записать оператор

do *jgl =* 0, –1

и оператор

do *j =* 1 + *jglr*2*+k*,min((*jgl+* 1)*r*2*+k,* *N + P*)

Вариант 2:

do *k=*1*,N*

*l*(1)*=*1*/a*(1*,k*)

do *p =* 2,*N*

*l*(*p*)*=* *–a*(*p,k*)

enddo

do *jgl=* 0, –1

do *j =* 1 + *jglr*2*+k*,min((*jgl+* 1)*r*2*+k,* *N + P*)

*u*(*j*)*=a*(1*,j*)*l*(1)

enddo

do *i =* 2, *N*

do *j =* 1 + *jglr*2*+k*,min((*jgl+* 1)*r*2*+k,* *N + P*)

*a*(*i–*1*,j*)*=a*(*i,j*)*+l*(*i*)*u*(*j*)

enddo

enddo

do *j =* 1 + *jglr*2*+k*,min((*jgl+* 1)*r*2*+k,* *N + P*)

*a*(*N,j*)*=u*(*j*)

enddo

enddo(*jgl*)

enddo(*k*)

Приведем результаты вычислительных экспериментов.

*N* = 1000



Red –  реализация с избыточными вычислениями

Yellow – реализация без избыточных вычислений (вариант 1)

Green – реализация без избыточных вычислений (вариант 2)

*N* = 2000 (разбиваются циклы с параметрами *j* и *i*)

|  |  |  |  |
| --- | --- | --- | --- |
| r2\*r3 | С избыточными | Без избыточных (1) | Без избыточных (2) |
| 10\*10 | 43.68 | 42.009 | 41.21 |
| 15\*10 | 44.45 | 43.57 | 43.11 |
| 20\*10 | 46.59 | 45.45 | 45.26 |
| 30\*10 | 38.54 | 37.68 | 37.60 |
| 20\*20 | 40.58 | 40.23 | 39.94 |
| 50\*10 | 30.67 | 30.34 | 30.14 |
| 100\*100 | 43.86 | 43.84 | 43.82 |

**Пример 8** (допустимость тайлинга для алгоритма прямого хода метода Гаусса без выбора ведущего элемента). Пусть *A* – матрица порядка *n*, *b* – *n*-мерный вектор. Введем в рассмотрение расширенную матрицу системы (строка *i* расширенной матрицы есть *i*-я строка матрицы системы, дополненная элементом *bi* вектора правых частей) и запишем алгоритм прямого хода метода Гаусса решения системы *Ax=b*:

do *k=* 1*, n–*1

do *i= k+*1*, n*

do *j= k+*1*, n+*1

*S*1(*k,i,j*): *a*(*i,j*)*=a*(*i,j*) *–*  *a*(*k,j*) (8)

enddo

enddo

enddo

Гнездо циклов содержит один оператор и один массив данных (расширенную матрицу системы). В левой части оператора происходит определение элемента массива. На каждом вхождении в оператор массива *a* в правой части происходит использование ранее вычисленного элемента массива (при *k=*1 используются входные данные): *a*(*i,j*) – использование прежнего значения обновляемого элемента, *a*(*i,k*) – использование столбца, содержащего ведущий элемент, *a*(*k,j*) – использование строки, содержащей ведущий элемент, *a*(*k,k*) – использование ведущего элемента. Таким образом, вхождение массива *a* в левую часть оператора и вхождения в правую часть порождают зависимости. Если понимать детали алгоритма, то формализовать зависимости проще.

Изобразим схематично зависимости операций *k*-го шага, порождаемые ведущим элементом *a*(*k,k*), вычисленном на (*k*–1)-м шаге:

*i*

*j*

***a*(*k,k*)**

Изобразим теперь схематично зависимости операций *k*-го шага, порождаемые строками и столбцами (вычисленными на (*k*–1)-м шаге), содержащими ведущий элемент:

***j***

*i*

***a(k,k)***

***a(k,j)***

***a(i,k)***

Укажем итерации, порождающие зависимости.

*S*1(*k–*1,*i,j*)*S*1(*k,i,j*): данное *a*(*i,j*), вычисленное на итерации (*k–*1,*i,j*), является аргументом *a*(*i,j*) для вычислений на текущей итерации (*k,i,j*).

*S*1(*k–*1,*i,k*)*S*1(*k,i,j*): *a*(*i,k*), вычисленное на итерации (*k–*1,*i,k*), является аргументомдля вычислений на текущей итерации (*k,i,j*);

*S*1(*k–*1,*k,j*)*S*1(*k,i,j*): *a*(*k,j*), вычисленное на итерации (*k–*1,*k,j*), является аргументомдля вычислений на текущей итерации (*k,i,j*);

*S*1(*k–*1,*k,k*)*S*1(*k,i,j*): ведущий элемент *a*(*k,k*), вычисленный на итерации (*k–*1,*k,k*), является аргументомдля вычислений на текущей итерации (*k,i,j*).

Достаточные условия допустимости тайлинга выполняются: условия (6) выполняются, так как *j>k*, *i>k*; условия (7) выполняются, так как имеется только один оператор.

Выделим в алгоритме (8) преобразование строки, содержащей ведущий элемент:

do *k=* 1*, n–*1

do *j= k+*1*, n+*1

*S*0(*k,j*): *u*(*k,j*)*=*

enddo

do *i= k+*1*, n* (9)

do *j= k+*1*, n+*1

*S*1(*k,i,j*): *a*(*i,j*)*=a*(*i,j*) *– a*(*i,k*) *u*(*k,j*)

enddo

enddo

enddo

Зависимости *S*1(*k–*1,*i,j*)*S*1(*k,i,j*) и *S*1(*k–*1,*i,k*)*S*1(*k,i,j*) для алгоритма (9) такие же, как и для алгоритма (8). Появились новые зависимости:

*S*1(*k–*1,*k,j*)*S*0(*k,j*): *a*(*k,j*), вычисленное операцией *S*1(*k–*1,*k,j*), является аргументомдля вычислений операции *S*0(*k,,j*);

*S*1(*k–*1,*k,k*)*S*0(*k,j*): ведущий элемент *a*(*k,k*), вычисленный операцией *S*1(*k–*1,*k,k*), является аргументомдля вычислений операции *S*0(*k,j*).

*S*0(*k,j*)*S*1(*k,i,j*): *u*(*k,j*), вычисленное операцией *S*0(*k,j*), является аргументомдля вычислений операции *S*1(*k,i,j*).

Рассмотрим допустимость тайлинга, для которого *r*1 – параметр, *r*2=*n–*1 (цикл *i* считается локальным не разбиваемым), *r*3 – параметр.

Достаточные условия допустимости тайлинга выполняются для зависимостей *S*1(*k–*1,*i,j*)*S*1(*k,i,j*) и *S*1(*k–*1,*i,k*) *S*1(*k,i,j*) (показано ранее), а также для зависимости *S*0(*k,j*)*S*1(*k,i,j*) (условия (6) выполняются, так как операторы имеют только один общий цикл; условия (7) выполняются, так как , ).

Достаточные условия допустимости тайлинга не выполняются для зависимостей *S*1(*k–*1,*k,j*)*S*0(*k,j*) и *S*1(*k–*1,*k,k*)*S*0(*k,j*). Исследуем, выполняются ли для этих зависимостей необходимые и достаточные условия (4) и (5) теоремы 1.

Имеем: *c*1,0=1 (один общий цикл в окружении операторов *S*1 и *S*0); *I=*(*k–*1,*k,j*) (для второй из рассматриваемых зависимостей *I=*(*k–*1,*k,k*)), *J=*(*k,j*); *=*, где *m*1*=*1, *=* (по формуле (2)). На *r*1*–*1 итерации из каждых *r*1 итераций *k* имеет место *=*. Если *r*1*>*1, то условие (5) требует, чтобы оператор *S*1 встречался в записи алгоритма раньше оператора *S*0, что неверно. Таким образом, если *r*1*>*1, тайлинг не допустим. Если *r*1*=*1, то *>*, в условие (4) выполняется строгое неравенство, тайлинг допустим.

Таким образом, если *r*2=*n–*1, *r*3 – параметр, то преобразование алгоритма (9) посредством тайлинга возможно только в случае *r*1*=*1. В то же время, математически эквивалентный алгоритму (9) алгоритм (8) допускает применение тайлинга с любыми *r*1, *r*2, *r*3.

**Заключение.** Тайлинг – одно из самых часто используемых преобразований алгоритмов при выполнении как последовательных, так и параллельных вычислений. Применяется тайлинг для разбиения множества операций алгоритма на макрооперации, называемые зернами вычислений или тайлами. Зернистые вычисления позволяют уменьшить накладные расходы на использование иерархической памяти при последовательных вычислениях и на организацию обмена данными при параллельных вычислениях.

Мы ознакомились с основами теории тайлинга, с применением тайлинга для организации параллельных вычислительных процессов и потоков при программировании на компьютерах с распределенной памятью, на многоядерных процессорах, на графических процессорах.

**Краткий вопрос для зачета:** Использование тайлинга (перечислить, для чего можно использовать разбиение множества операций алгоритма на макрооперации).

**Ответ:** Разбиение множества операций алгоритма на макрооперации-тайлы можно использовать 1) для улучшения локальности как при последовательных, так и при параллельных вычислениях на компьютерах с общей памятью, 2) для задания зерна вычислений при организации параллельных зернистых вычислительных процессов на компьютерах с распределённой памятью, 3) для задания потоков и блоков вычислений при организации параллельных вычислений на графических процессорах.

**Список использованных источников**

1. Renganarayanan L., Kim D., Rajopadhye S., Strout M.M. Parameterized tiled loops for free// ACM SIGPLAN Conference on Programming Language Design and Implementation.San Diego, California, USA, June 2007. P. 126–138.

2. Hartono A., Baskaran M., Ramanujam J., Sadayappan P. DynTile: Parametric tiled loop generation for parallel execution on multicore processors. // Proceedings of 24th International Parallel and Distributed Processing Symposium, Atlanta, April 2010. P.1–12.

3. Baskaran M., Bondhugula U, Krishnamoorthy S., Ramanujam J., Rountev A., Sadayappan, P. Automatic data movement and computation mapping for multi-level parallel architectures with explicitly managed memories // Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. February 2008.

4. Baskaran M., Ramanujam J., Sadayappan, P. Automatic C-to- CUDA code generation for affine programs // Proceedings of the Compiler Construction, 19th International Conference. Part of the Joint European Conferences on Theory and Practice of Software. Paphos, Cyprus, March 20–28, 2010.

5. Di P, [Ye](http://www.informatik.uni-trier.de/~ley/pers/hd/y/Ye:Ding.html) D., [Su](http://www.informatik.uni-trier.de/~ley/pers/hd/s/Su:Yu.html) Yu, [Sui](http://www.informatik.uni-trier.de/~ley/pers/hd/s/Sui:Yulei.html) Y., [Xue](http://www.informatik.uni-trier.de/~ley/pers/hd/x/Xue:Jingling.html) J. Automatic parallelization of tiled loop nests with enhanced fine-grained parallelism on GPUs // 41st International Conference on Parallel Processing, ICPP 2012. Pittsburgh, PA, USA, September 10-13. IEEE Computer Society, 2012: P. 350-359.

6. Kim D., Rajopadhye S. Parameterized tiling for imperfectly nested loops. // Technical Report CS-09-101, Colorado State University, Department of Computer Science, February 2009. 21 P.

7. Tavarageri S., Hartono A., Baskaran M., Pouchet L.-N., Ramanujam J., Sadayappan P. Parametric tiling of affine loop nests // Proc. 15th Workshop on Compilers for Parallel Computers. Vienna, Austria, July 2010.

8. Лиходед Н.А. Обобщенный тайлинг // Доклады НАН Беларуси. 2011. Т. 55, № 1. С. 16-21.