БЛОЧНЫЕ МАТРИЦЫ. БЛОЧНОЕ ПЕРЕМНОЖЕНИЕ МАТРИЦ. ПАРАЛЛЕЛЬНЫЕ ЦИКЛЫ. СКОШЕННЫЙ ПАРАЛЛЕЛИЗМ

Блочные алгоритмы используются для построения эффективных параллельных алгоритмов и для уменьшения накладных расходов на использование памяти при последовательных вычислениях. В этом материале рассмотрены последовательный и параллельный блочные алгоритмы перемножение матриц, указаны некоторые их преимущества по сравнению с обычными, точечными алгоритмами. Кратко рассматривается так называемый скошенный (косой) параллелизм.

Блочные матрицы. Операции над блочными матрицами.

Рассмотрим некоторую матрицу A. Разделим матрицу горизонтальными и вертикальными прямыми линиями. Получим подматрицы, которые называются блоками матрицы A. Например, если использовать одну вертикальную и одну горизонтальную прямые, то получим разбиение матрицы на четыре блока:

$$A=\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix} = \left(\begin{array}{|cc|cc} \hline a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \\ \hline \end{array} \right) = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}.$$

Очевидно, что умножение блочной матрицы на число, вычитание и сложение матриц с блоками согласованных размеров совершаются по тем же правилам, что и операции над обычными матрицами.

Проиллюстрируем, что и перемножение блочных матриц выполняется таким же образом, как и перемножение обычных матриц. Пусть матрицы A и B имеют две блочные строки, два блочных столбца и блоки произвольного размера $r\times r$. В этом случае порядок матриц равен $2r$. Рассмотрим результат перемножения матриц A и B: $C=A\cdot B$, $c_{i,j}=\sum_{k=1}^{2r} a_{i,k}b_{k,j}$, $1\le i\le 2r$, $1\le j\le 2r$. Покажем, что

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \begin{pmatrix} A_{1,1}B_{1,1}+A_{1,2}B_{2,1} & A_{1,1}B_{1,2}+A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1}+A_{2,2}B_{2,1} & A_{2,1}B_{1,2}+A_{2,2}B_{2,2} \end{pmatrix}.$$

Действительно, левая часть есть

$$A\cdot B=\left(\sum_{k=1}^{2r} a_{i,k}b_{k,j}\right)_{i=1,j=1}^{2r,2r}.$$

В правой части имеем:

$$\begin{pmatrix} \left(\sum_{k=1}^{r} a_{i,k}b_{k,j}\right)_{i=1,\,j=1}^{r,r} + \left(\sum_{k=r+1}^{2r} a_{i,k}b_{k,j}\right)_{i=1,\,j=1}^{r,r} & \left(\sum_{k=1}^{r} a_{i,k}b_{k,j}\right)_{i=1,\,j=r+1}^{r,2r} + \left(\sum_{k=r+1}^{2r} a_{i,k}b_{k,j}\right)_{i=1,\,j=r+1}^{r,2r} \\ \left(\sum_{k=1}^{r} a_{i,k}b_{k,j}\right)_{i=r+1,\,j=1}^{2r,r} + \left(\sum_{k=r+1}^{2r} a_{i,k}b_{k,j}\right)_{i=r+1,\,j=1}^{2r,r} & \left(\sum_{k=1}^{r} a_{i,k}b_{k,j}\right)_{i=r+1,\,j=r+1}^{2r,2r} + \left(\sum_{k=r+1}^{2r} a_{i,k}b_{k,j}\right)_{i=r+1,\,j=r+1}^{2r,2r} \end{pmatrix} =$$

$$\begin{pmatrix} \left(\sum_{k=1}^{2r} a_{i,k}b_{k,j}\right)_{i=1,\,j=1}^{r,r} & \left(\sum_{k=1}^{2r} a_{i,k}b_{k,j}\right)_{i=1,\,j=r+1}^{r,2r} \\ \left(\sum_{k=1}^{2r} a_{i,k}b_{k,j}\right)_{i=r+1,\,j=1}^{2r,r} & \left(\sum_{k=1}^{2r} a_{i,k}b_{k,j}\right)_{i=r+1,\,j=r+1}^{2r,2r} \end{pmatrix} = \left(\sum_{k=1}^{2r} a_{i,k}b_{k,j}\right)_{i=1,\,j=1}^{2r\;\;2r}.$$

Отметим, что при получении блоков результирующей матрицы суммы имеют вид $A_{i^{gl},1}\,B_{1,j^{gl}} + A_{i^{gl},2}\,B_{2,j^{gl}}$ (i^{gl}, j^{gl} – индексы глобальной, т.е. уровня блоков, нумерации). Поэтому произведение матриц в первом слагаемом порождает суммы с пределами суммирования от 1 до r, во втором слагаемом – от $r+1$ до $2r$, пределы изменения i и j в блочных слагаемых совпадают.

Алгоритм перемножение блочных матриц в рассматриваемом случае представим в виде

do $i^{gl} = 0, 1$
  do $j^{gl} = 0, 1$
    do $k^{gl} = 0, 1$
      $C_{i^{gl}+1,\,j^{gl}+1} = C_{i^{gl}+1,\,j^{gl}+1} + A_{i^{gl}+1,k^{gl}+1} \cdot B_{k^{gl}+1,\,j^{gl}+1}$
    enddo
  enddo
enddo

Операцию $C_{i^{gl}+1,\,j^{gl}+1} = C_{i^{gl}+1,\,j^{gl}+1} + A_{i^{gl}+1,k^{gl}+1} \cdot B_{k^{gl}+1,\,j^{gl}+1}$ перемножения блоков матриц A и B с суммированием промежуточных результатов (перед началом вычислений массив C обнуляется) можно представить в следующем виде:

do $i = 1+ i^{gl}\,r, (i^{gl}+1)r$
  do $j = 1+ j^{gl}\,r, (j^{gl}+1)r$
    do $k = 1+ k^{gl}\,r, (k^{gl}+1)r$
      $c(i,j) = c(i,j) + a(i,k)\; b(k,j)$
    enddo
  enddo
enddo

Таким образом, на примере матриц с двумя блочными строками и двумя блочными столбцами показано, что операция перемножения блочных матриц выполняется по тем же правилам, что и операция перемножения обычных матриц.

При транспонировании матрицы, разбитой на блоки, транспонированию подлежат как блоки матрицы, так и элементы блоков. Например:

$$A^{\mathrm{T}} = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} A_{1,1}^{\mathrm{T}} & A_{2,1}^{\mathrm{T}} \\ A_{1,2}^{\mathrm{T}} & A_{2,2}^{\mathrm{T}} \end{pmatrix}.$$

Действительно, для блоков размера 2×2:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix},$$

$$\begin{pmatrix} A_{1,1}^{\mathrm{T}} & A_{2,1}^{\mathrm{T}} \\ A_{1,2}^{\mathrm{T}} & A_{2,2}^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}^{\mathrm{T}} & \begin{bmatrix} a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix}^{\mathrm{T}} \\ \begin{bmatrix} a_{1,3} & a_{1,4} \\ a_{2,3} & a_{2,4} \end{bmatrix}^{\mathrm{T}} & \begin{bmatrix} a_{3,3} & a_{3,4} \\ a_{4,3} & a_{4,4} \end{bmatrix}^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} \begin{bmatrix} a_{1,1} & a_{2,1} \\ a_{1,2} & a_{2,2} \end{bmatrix} & \begin{bmatrix} a_{3,1} & a_{4,1} \\ a_{3,2} & a_{4,2} \end{bmatrix} \\ \begin{bmatrix} a_{1,3} & a_{2,3} \\ a_{1,4} & a_{2,4} \end{bmatrix} & \begin{bmatrix} a_{3,3} & a_{4,3} \\ a_{3,4} & a_{4,4} \end{bmatrix} \end{pmatrix} = A^{\mathrm{T}}.$$

Использование блочных алгоритмов в последовательных вычислениях

Время решения задачи на современном компьютере определяется не столько скоростью выполнения вычислительных операций, сколько скоростью доступа к памяти. Скорость доступа существенно зависит от места в иерархической памяти (диск, оперативная память, кэши, регистры), где хранятся данные. Поэтому для быстрой реализации алгоритма, заданного последовательной программой, задача быстрого (до исчезновения из памяти с быстрым доступом) переиспользования данных является одной из важнейших.

Загрузка значений в кэш осуществляется блоками (строками кэша) из последовательно расположенных в памяти элементов массивов (размер строки кэша может быть равен, например, 128 байтам). Кэш будет эффективно использоваться, если в выполняемых друг за другом вычислениях происходит обращение к одним и тем же элементам оперативной памяти или к элементам памяти с последовательно возрастающими адресами. Блок матрицы содержит близко расположенные в памяти элементы массива по строкам (при использовании языка программирования C) или по столбцам (при использовании Фортрана). Поэтому при выполнении блочных вариантов алгоритмов кэш используется эффективно.

Рассмотрим алгоритм перемножения матрицы A размера $N_1\times N_3$ и матрицы B размера $N_3\times N_2$:

```
do i = 1, N1
  do j = 1, N2
    c(i,j) = 0
    do k = 1, N3
      c(i,j) = c(i,j) + a(i,k) b(k,j)                    (1)
    enddo
  enddo
enddo
```

Пусть хранение элементов массивов осуществляется по строкам. При выполнении алгоритма используются N_3 раза элементы массива $b(k,j)$, расположенные в памяти на расстоянии друг от друга N_2 ячеек. Если N_3 и N_2 являются достаточно большими числами, то обращение к элементу массива $b(k,j)$, перенесенному ранее в кэш в составе строки кэша (т.е. в составе последовательно расположенных в памяти элементов массива), произойдет уже после того, как он будет «вытеснен» из кэша другими данными. Таким образом, при больших значениях N_3 и N_2 обращение к близко расположенным в памяти элементам массива происходит через значительное число итераций и неизбежны кэш-промахи.

Блок матрицы имеет сравнительно небольшие размеры, внутренние циклы блочных алгоритмов выполняют сравнительно (с размером задачи) небольшое число итераций. Поэтому при выполнении блочных вариантов алгоритмов кэш используется эффективно: гораздо чаще происходит обращение к близко расположенным в памяти элементам массива (причем независимо от того, по строкам или по столбцам они расположены в памяти).

Обозначим $Q_1 = \frac{N_1}{r}$, $Q_2 = \frac{N_2}{r}$, $Q_3 = \frac{N_3}{r}$. Будем предполагать для простоты записи пределов изменения числа блоков, что Q_1, Q_2, Q_3 являются целыми; иначе потребовалось бы использовать обозначение ближайшего «сверху» целого числа: $\left\lceil \frac{N_1}{r} \right\rceil$, $\left\lceil \frac{N_2}{r} \right\rceil$, $\left\lceil \frac{N_3}{r} \right\rceil$. Числа Q_1, Q_2, Q_3 задают количество блочных строк и столбцов матриц *A*, *B* и *C*. Например, Q_1 задает количество блочных строк матриц *A* и *C*.

Рассмотрим алгоритм перемножения блочных матриц с блоками размера $r \times r$. Перед началом вычислений массив *C* обнуляется. Операции, как и в рассмотренном ранее частном случае, объединены в макрооперации. Алгоритм уровня макроопераций имеет следующий вид:

```
do i^gl = 0, Q1 – 1
    do j^gl = 0, Q2 – 1
        do k^gl = 0, Q3 – 1
            C_{i^gl+1, j^gl+1} = C_{i^gl+1, j^gl+1} + A_{i^gl+1, k^gl+1} · B_{k^gl+1, j^gl+1}
```

```
        enddo
      enddo
  enddo
```

Блочные алгоритмы являются двухуровневыми: на первом (глобальном) уровне выполняются блочные операции, на втором (локальном) уровне выполняются обычные операции. Одной макрооперацией $C_{i^{gl}+1,\,j^{gl}+1} = C_{i^{gl}+1,\,j^{gl}+1} + A_{i^{gl}+1,k^{gl}+1} \cdot B_{k^{gl}+1,\,j^{gl}+1}$ является перемножение блоков матриц A и B с суммированием промежуточных результатов:

```
do i = 1+ i^gl r, (i^gl+1)r
    do j = 1+ j^gl r, (j^gl+1)r
        do k = 1+ k^gl r, (k^gl+1)r
            c(i,j) = c(i,j) + a(i,k) b(k,j)
        enddo
    enddo
enddo
```

Фактически реализуется следующий алгоритм:

```
do i = 1, N_1
  do j = 1, N_2
    c(i,j) = 0
  enddo
enddo
do i^gl = 0, Q_1 –1
  do j^gl = 0, Q_2 –1
    do k^gl = 0, Q_3 –1
      do i = 1+ i^gl r, (i^gl+1)r
        do j = 1+ j^gl r, (j^gl+1)r
          do k = 1+ k^gl r, (k^gl+1)r
            c(i,j) = c(i,j) + a(i,k) b(k,j)                    (2)
          enddo
        enddo
      enddo
    enddo(k^gl)
  enddo(j^gl)
enddo(i^gl)
```

При проведении экспериментов вычисления надо оптимизировать. Например: $ai=i^{gl}r$, $aileft=i^{gl}r+1$, $airight=i^{gl}r+r$ вычислить один раз (сразу после do $i^{gl}=0,Q_1-1$), а не вычислять многократно 1+ $i^{gl}\,r$, $(i^{gl}+1)r$.

Параллельные блочные алгоритмы для реализации на многоядерном CPU

В рассмотренном алгоритме перемножения матриц вычисление всех элементов массива $c(i,j)$ можно проводить параллельно, независимо друг от друга. Фактически независимо вычисляются суммы $c_{i,j}=\sum_{k=1}^{N_3} a_{i,k}b_{k,j}$, $1\le i\le N_1$, $1\le j\le N_2$ (циклы, итерации которых заведомо можно выполнять независимо, запишем как dopar):

```
dopar i = 1, N1 // Запись в виде dopar означает возможность, но не
                   обязательность, независимого выполнения итераций цикла
  dopar j = 1, N2
    c(i,j) = 0
    do k = 1, N3
      c(i,j) = c(i,j) + a(i,k)·b(k,j)
    enddo
  enddopar
enddopar
```

Этот алгоритм можно использовать для OpenMP-реализаций на многоядерных компьютерах. Одновременное выполнение операций потоками вычислений позволяет добиться ускорения вычислений на многоядерном процессоре.

Можно также использовать блочный алгоритм, параллельность внешних циклов сохраняется.

Использование для организации параллельных потоков циклов с параметром i^{gl}:

```
do i = 1, N1
  do j = 1, N2
    c(i,j) = 0
  enddo
enddo
// Итерации следующего цикла распределяются между потоками
//          #pragma omp parallel for num_threads(4)
dopar i^gl = 0, Q1 –1
  do j^gl = 0, Q2 –1
    do k^gl = 0, Q3 –1
      do i = 1+ i^gl r, (i^gl+1)r
        do j = 1+ j^gl r, (j^gl+1)r
          do k = 1+ k^gl r, (k^gl+1)r
            c(i,j) = c(i,j) + a(i,k) b(k,j)                    (3)
          enddo
```

```
          enddo
        enddo
      enddo(k^gl)
    enddo(j^gl)
  enddopar(i^gl)
```

Использование для организации параллельных потоков циклов с параметром j^{gl}:

```
do i = 1, N_1
  do j = 1, N_2
    c(i,j) = 0
  enddo
enddo
do i^gl = 0, Q_1 –1
  // Итерации следующего цикла распределяются между потоками
  //              #pragma omp parallel for num_threads(4)
  dopar j^gl = 0, Q_2 –1
    do k^gl = 0, Q_3 –1
      do i = 1+ i^gl r, (i^gl+1)r
        do j = 1+ j^gl r, (j^gl+1)r
          do k = 1+ k^gl r, (k^gl+1)r
            c(i,j) = c(i,j) + a(i,k) b(k,j)                    (4)
          enddo
        enddo
      enddo
    enddo(k^gl)
  enddopar(j^gl)
enddo(i^gl)
```

Использование для организации параллельных потоков циклов с параметром i^{gl} и с параметром j^{gl}:

```
do i = 1, N_1
  do j = 1, N_2
    c(i,j) = 0
  enddo
enddo
// Итерации следующих двух циклов распределяются между потоками
//              #pragma omp parallel for num_threads(4) collapse(2)
dopar i^gl = 0, Q_1 –1
  dopar j^gl = 0, Q_2 –1
    do k^gl = 0, Q_3 –1
      do i = 1+ i^gl r, (i^gl+1)r
        do j = 1+ j^gl r, (j^gl+1)r
```

```
            do k = 1+ k^gl r, (k^gl+1)r
               c(i,j) = c(i,j) + a(i,k) b(k,j)                    (5)
            enddo
          enddo
        enddo
      enddo(k^gl)
    enddopar (j^gl)
  enddopar(i^gl)
```

Как и в случае последовательных алгоритмов, при выполнении блочных вариантов эффективно используется кэш. С ростом числа используемых ядер преимущество параллельного алгоритма по сравнению с последовательным алгоритмом увеличивается, так как помимо общей для всех ядер кэш-памяти каждое ядро имеет свою кэш-память. Суммарно размер кэша становится больше.

Скошенный параллелизм

Сделаем еще замечание о так называемом скошенном (косом) параллелизме.

Пусть в алгоритме вида

```
do  i^gl = 0, Q_1 –1
    do  j^gl = 0, Q_2 –1
        Bl(i^gl,j^gl)
    enddo
enddo
```

циклы не являются параллельными. Тогда часто на практике можно одновременно и независимо выполнять итерации (i^{gl},j^{gl}) такие, что $i^{gl}+j^{gl}=t$, где t является фиксированным числом:

```
do  t=0, Q_1 + Q_2 –2
    dopar  i^gl = 0, Q_1 –1
        dopar  j^gl = 0, Q_2 –1
            if  i^gl+j^gl=t  then
            Bl(i^gl,j^gl)
        enddopar
    enddopar
enddo
```

Алгоритм с явно заданным параллелизмом можно представить еще в следующем виде:

```
do  t=0, Q_1 + Q_2–2
    dopar  j^gl = max(0, t– Q_1 +1), min(t, Q_2–1)
```

$i^{gl}=t-j^{gl}$
Bl(i^{gl},j^{gl})
enddopar
enddo

Такая запись является следствием аффинного преобразования $t=i^{gl}+j^{gl}$, $j^{gl}new=j^{gl}old$ исходного алгоритма

do i^{gl}= 0, Q_1–1
do j^{gl}= 0, Q_2–1
Bl(i^{gl},j^{gl})
enddo
enddo

Обоснуем указанный конструктивный способ выделения параллелизма.

Определение. *Итерация (i_{in},j_{in}) зависит от итерации (i_{out},j_{out}), если некоторое данное, вычисленное на итерации (i_{out},j_{out}), используется на итерации (i_{in},j_{in}).*

Утверждение 1. *Пусть имеется функция $t_1(i,j)$ такая, что для любой пары зависимых итераций (i_{out},j_{out}) и (i_{in},j_{in}) справедливо $t_1(i_{in},j_{in})>t_1(i_{out},j_{out})$. Тогда можно одновременно и независимо выполнять итерации (i,j), для которых функция $t_1(i,j)$ принимает одно и то же значение.*

Действительно, если выполняется условие утверждения 1, то среди итераций с одинаковым значением функции $t_1(i,j)=t$ не может быть зависимых: для любых двух зависимых итераций (i_1,j_1) и (i_2,j_2) имеет место либо $t_1(i_1,j_1)>t_1(i_2,j_2)$, либо $t_1(i_2,j_2)>t_1(i_1,j_1)$.

Утверждение 2. *Если для любой пары зависимых итераций (i_{out},j_{out}) и (i_{in},j_{in}) справедливо $i_{in}\geq i_{out}$, $j_{in}\geq j_{out}$, то можно одновременно и независимо выполнять итерации (i,j) такие, что i+j=t, где t есть фиксированное число.*

Утверждение 2 является следствием утверждения 1, если рассмотреть $t_1(i,j)=i+j$: так как $i_{in}\geq i_{out}$, $j_{in}\geq j_{out}$, $(i_{out},j_{out})\neq(i_{in},j_{in})$, то $t_1(i_{in},j_{in})=i_{in}+j_{in}>i_{out}+j_{out}=t_1(i_{out},j_{out})$.

Замечание. В действительности следует учитывать еще так называемые ложные зависимости: изменение порядка выполнения операций должно сохранять корректное использование ячеек памяти.