УЛУЧШЕНИЕ ЛОКАЛЬНОСТИ ДЛЯ ПОСЛЕДОВАТЕЛЬНЫХ ВЫЧИСЛЕНИЙ
(Улучшение локальности вычислений на компьютерах с общей памятью)

Локальность алгоритма – это вычислительное свойство алгоритма, отражающее степень использования памяти с быстрым доступом. Для компьютеров с распределенной памятью быстрым считается доступ к локальной памяти процессора, для компьютеров с общей памятью – обращение к кэшам. При вычислениях на GPU быстрым является процесс обращения к регистрам, разделяемой памяти мультипроцессора и кэшам, но не обращение к глобальной и локальной видам памяти GPU.

Реорганизация вычислений алгоритма таким образом, чтобы добиться более частого использования памяти с быстрым доступом, называется улучшением локальности данных (или просто улучшением локальности). Улучшение локальности данных называют еще локализацией данных.

Пространственная локальность. Временная локальность. Время решения задачи на современном компьютере определяется не столько скоростью выполнения вычислительных операций, сколько скоростью доступа к памяти («стена памяти» – примерно 100 тактов при обращении процессора к оперативной памяти). Скорость доступа существенно зависит от места в иерархической памяти (Рисунок 1 [1]), 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. Иметь латентность для всей оперативной памяти, как у кэш-памяти, технологически сложно и очень дорого, поэтому возникает иерархия памяти.

Рисунок 1 – Иерархия памяти процессора

В некоторых многоядерных процессорах L3-кэш является общим для

всех ядер, в то время как каждое ядро имеет свою кэш-память меньших уровней. Когда требуется какое-либо данное, то производится проверка на наличие этого данного в кэше процессора. Если данное находится в кэш-памяти, то обращение к оперативной памяти не происходит. Такая ситуация называется попаданием в кэш-память (cache hit). В противном случае происходит кэш-промах (cache miss) – считывание данного из оперативной памяти.

Отметим еще, что энергозатраты на выполнение одной арифметической операции составляют 100 пДж (1 пДж равен 10^{-12} Дж), в то время как энергозатраты на перемещение данных в пределах вычислительного узла – 4800 пДж.

Так как время считывания данных из оперативной памяти многократно (например, в 15 раз) превосходит время считывания из кэш-памяти, то возникает задача минимизации кэш-промахов.

Будем считать, что алгоритм задан последовательной программой, основную вычислительную часть которой составляет многомерный цикл произвольной структуры вложенности, границы изменения параметров циклов задаются неоднородными (в общем случае) формами, линейными по совокупности параметров циклов и внешних переменных. Пусть в гнезде циклов имеется K выполняемых операторов S_β и используется L массивов a_l. Область изменения параметров гнезда циклов для оператора S_β будем называть итерационной областью и обозначать V_β. Обозначим n_β – число циклов, окружающих оператор S_β, ν_l – размерность l-го массива.

Вхождением (l,β,q) будем называть q-е вхождение массива a_l в оператор S_β; наряду с обозначением (l,β,q) будем также использовать более наглядное обозначение (a_l,S_β,q). Индексы элементов массива, относящегося к вхождению (l,β,q), будем выражать аффинной функцией

$$\overline{F}_{l,\beta,q}(J) = F_{l,\beta,q}J + G_{l,\beta,q}N + f^{(l,\beta,q)},$$

$J \in V_\beta$, $F_{l,\beta,q} \in \mathbf{Z}^{\nu_l \times n_\beta}$, $N = (N_1, N_2, .., N_e)$ – вектор внешних переменных алгоритма, e – число таких переменных, $G_{l,\beta,q} \in \mathbf{Z}^{\nu_l \times e}$, $f^{(l,\beta,q)} \in \mathbf{Z}^{\nu_l}$. Функция $\overline{F}_{l,\beta,q}(J)$ называется функцией доступа (к массиву a_l в операторе S_β).

Будем говорить, что цикл локален на вхождении (l,β,q), если операции алгоритма при изменении параметра цикла используют близкие с точки зрения хранения в памяти данные.

Различают временную и пространственную локальность. Если цикл на вхождении (l,β,q) переиспользует элемент массива a_l (при изменении параметра цикла используется один тот же элемент массива), то говорят о временной локальности, локальности по времени. Если при изменении параметра цикла используются разные близко хранящиеся элементы

массива a_l, то говорят о пространственной локальности.

Для пространственной локальности требуется, чтобы операции алгоритма при изменении параметра цикла использовали данные одного блока кэша. Примем, что имеет место пространственная локальность, если цикл использует элементы одной строки массива, т.е. использует элементы, отличающиеся только последним индексом массива. Предполагается, что хранение элементов массива осуществляется по строкам (как при использовании языка программирования С). Если элементы массива хранятся по столбцам (как при использовании Фортрана), то близко расположенными в памяти являются элементы массива, отличающиеся только первой координатой в индексных выражениях.

Данные называют локальными на некотором вхождении, если в ходе выполнения алгоритма они переиспользуются до исчезновения из памяти с быстрым доступом. Если говорится о временной или пространственной локальности данных на некотором вхождении, то имеется в виду локальность на этом вхождении самого внутреннего цикла.

Пример 1. Рассмотрим основную часть *ijk*-алгоритма перемножения двух квадратных матриц некоторого порядка N:

```
do i = 1, N
   do j = 1, N
      do k = 1, N
         S1  c(i,j) = c(i,j) + a(i,k) b(k,j)
```

Перестановкой циклов можно получать различные модификации этого алгоритма. Приведем типичное соотношение времени вычислений для трех случаев очередности выполнения циклов; хранение элементов массивов осуществляется по строкам.

ijk-алгоритм – 89 (нет локальности на вхождении $(b, S_1, 1)$);

ikj-алгоритм – 17 (данные локальны на всех вхождениях);

jki-алгоритм – 307 (нет локальности на вхождениях $(c, S_1, 1)$, $(c, S_1, 2)$ и $(a, S_1, 1)$). □

Для улучшения локальности мы будем использовать тайлинг, в частности, перестановку циклов. Другим способом уменьшения накладных расходов на использование иерархической памяти является переразмещение данных.

Выбор циклов для формирования тайлов (выбор циклов для улучшения локальности). Тайлы следует формировать из циклов, локальных на вхождениях операторов тела цикла. Тогда, при небольших размерах тайла, параметр (счетчик) каждого из локальных циклов изменяется через небольшое число итераций алгоритма, что позволяет эффективно использовать кэш. Самый внешний из локальных циклов

можно не разбивать: длина этого цикла не влияет на частоту изменения параметров циклов тайла (если тайлинг не производить вообще, то самый внутренний цикл фактически является локальным не разбиваемым циклом).

Пусть, например, произведен тайлинг:

do $j^{gl} = 0, Q_2 - 1$
  do $k^{gl} = 0, Q_3 - 1$
    do $i = 1, N$
      do j (r_2 итераций)
        do k (r_3 итераций)
          операторы

Если r_2 и r_3 достаточно небольшие числа, то параметры i, j, k изменяются через небольшое число итераций алгоритма. Поэтому эффективно используется кэш при изменении всех трех параметров i, j, k.

Таким образом, *при использовании тайлинга для улучшения локальности последовательного алгоритма длина циклов тайла, за исключением самого внешнего из локальных циклов, не должна быть большой.*

Обозначим: $F^{(1)}_{l,\beta,q}$ – матрица размера $(\nu_l - 1) \times n_\beta$, составленная из всех, кроме последней, строк матрицы $F_{l,\beta,q}$; $\left(F^{(1)}_{l,\beta,q}\right)^{\mathrm{T}}_{\zeta}$ и $\left(F_{l,\beta,q}\right)^{\mathrm{T}}_{\zeta}$ – столбцы матриц $F^{(1)}_{l,\beta,q}$ и $F_{l,\beta,q}$ с номером ζ; $e^{(n_\beta)}_{\zeta}$ – вектор (вектор-столбец) размера n_β, у которого координата с номером ζ равна 1, а остальные координаты нулевые.

Теорема. *Если* $\left(F^{(1)}_{l,\beta,q}\right)^{\mathrm{T}}_{\zeta} = 0$*, то на вхождении* (l,β,q) *цикл с параметром* j_ζ *локален; если* $\left(F_{l,\beta,q}\right)^{\mathrm{T}}_{\zeta} = 0$*, то локальность является временной.*

Доказательство. Рассмотрим на вхождении (l,β,q) разность значений функции доступа $\overline{F}_{l,\beta,q}(J)$ при изменении параметра цикла j_ζ на единицу:

$$\overline{F}_{l,\beta,q}(J + e^{(n_\beta)}_{\zeta}) - \overline{F}_{l,\beta,q}(J) = F_{l,\beta,q}(J + e^{(n_\beta)}_{\zeta}) + G_{l,\beta,q}N + f^{(l,\beta,q)} -$$
$$(F_{l,\beta,q}J + G_{l,\beta,q}N + f^{(l,\beta,q)}) = F_{l,\beta,q}e^{(n_\beta)}_{\zeta} = \left(F_{l,\beta,q}\right)^{\mathrm{T}}_{\zeta}.$$

Отсюда следует, что если $\left(F^{(1)}_{l,\beta,q}\right)^{\mathrm{T}}_{\zeta} = 0$, то имеет место равенство всех, кроме, возможно, последней координаты векторов $\overline{F}_{l,\beta,q}(J)$ и $\overline{F}_{l,\beta,q}(J + e^{(n_\beta)}_{\zeta})$ (из строк матрицы $F^{(1)}_{l,\beta,q}$ образуются все, кроме последней, строки матрицы $F_{l,\beta,q}$). Это означает локальность цикла с параметром j_ζ: на вхождении (l,β,q) используются элементы одной строки массива a_l. Если выполняется

$\left(F_{l,\beta,q}\right)_{\zeta}^{\mathrm{T}}=0$, то $\overline{F}_{l,\beta,q}(J)=\overline{F}_{l,\beta,q}(J+e_{\zeta}^{(n_\beta)})$, используется только один элемент строки, имеет место локальность по времени. □

Пример 2. Рассмотрим основную часть алгоритма перемножения трех матриц: A размера $N_1\times N_3$, B размера $N_3\times N_2$, D размера $N_2\times N_3$.

Алгоритм 1 (последовательное перемножение трех матриц):

do i = 1, N_1
  do j = 1, N_2
    do k = 1, N_3
      S_1 $c(i,j) = c(i,j) + a(i,k)\, b(k,j)$
do i = 1, N_1
  do k = 1, N_3
    do j = 1, N_2
      S_2 $x(i,k) = x(i,k) + c(i,j)\, d(j,k)$

Произведем (после перестановки циклов с параметрами k и j во второй части алгоритма 1) слияние циклов с параметрами i и слияние циклов с параметрами j. Все эти преобразования допустимы.

Алгоритм 2 (получен из алгоритма 1 слиянием циклов):

do i = 1, N_1
  do j = 1, N_2
    do k = 1, N_3
      S_1 $c(i,j) = c(i,j) + a(i,k)\, b(k,j)$
    do k = 1, N_3
      S_2 $x(i,k) = x(i,k) + c(i,j)\, d(j,k)$

В алгоритме 2 данные локальны на всех вхождениях за исключением $(b,S_1,1)$: на вхождениях $(c,S_1,1)$, $(c,S_1,2)$, $(c,S_2,1)$ данные локальны по времени, на вхождениях $(a,S_1,1)$, $(x,S_2,1)$, $(x,S_2,2)$, $(d,S_2,1)$ имеет место пространственная локальность.

Выпишем матрицы вхождений элементов массивов в операторы: $F_{c,S_1,1}=F_{c,S_1,2}=F_{c,S_2,1}=\begin{pmatrix}1&0&0\\0&1&0\end{pmatrix}$, $F_{a,S_1,1}=F_{x,S_2,1}=F_{x,S_2,2}=\begin{pmatrix}1&0&0\\0&0&1\end{pmatrix}$, $F_{b,S_1,1}=\begin{pmatrix}0&0&1\\0&1&0\end{pmatrix}$, $F_{d,S_2,1}=\begin{pmatrix}0&1&0\\0&0&1\end{pmatrix}$. Из утверждения следует, что циклы с параметрами k локальны на всех вхождениях за исключением $(b,S_1,1)$. На вхождении $(b,S_1,1)$ локальны циклы с параметрами i и j.

Для локализации данных на $(b,S_1,1)$ осуществим тайлинг по циклу j. Пусть размер тайлов по координате j равен r_2; обозначим $Q_2=\left\lceil\frac{N_2}{r_2}\right\rceil$.

Алгоритм 3 (получен из алгоритма 2 разбиением по циклу j):

do $i = 1, N_1$
  do $j^{gl} = 0, Q_2 - 1$
    do $k = 1, N_3$
      do $j = 1 + j^{gl} r_2, \min((j^{gl}+1)r_2, N_2)$
        S_1 $c(i,j) = c(i,j) + a(i,k)\, b(k,j)$
    do $k = 1, N_3$
      do $j = 1 + j^{gl} r_2, \min((j^{gl}+1)r_2, N_2)$
        S_2 $x(i,k) = x(i,k) + c(i,j)\, d(j,k)$

В алгоритме 3 данные локальны на всех вхождениях. Самый внутренний цикл с параметром j имеет небольшую длину, параметр k изменяется через небольшое число итераций алгоритма. Поэтому сохраняется эффективное использование кэша при изменении параметра k, и к этому добавляется локальность, порождаемая параметром j.

Были проведены вычислительные эксперименты.

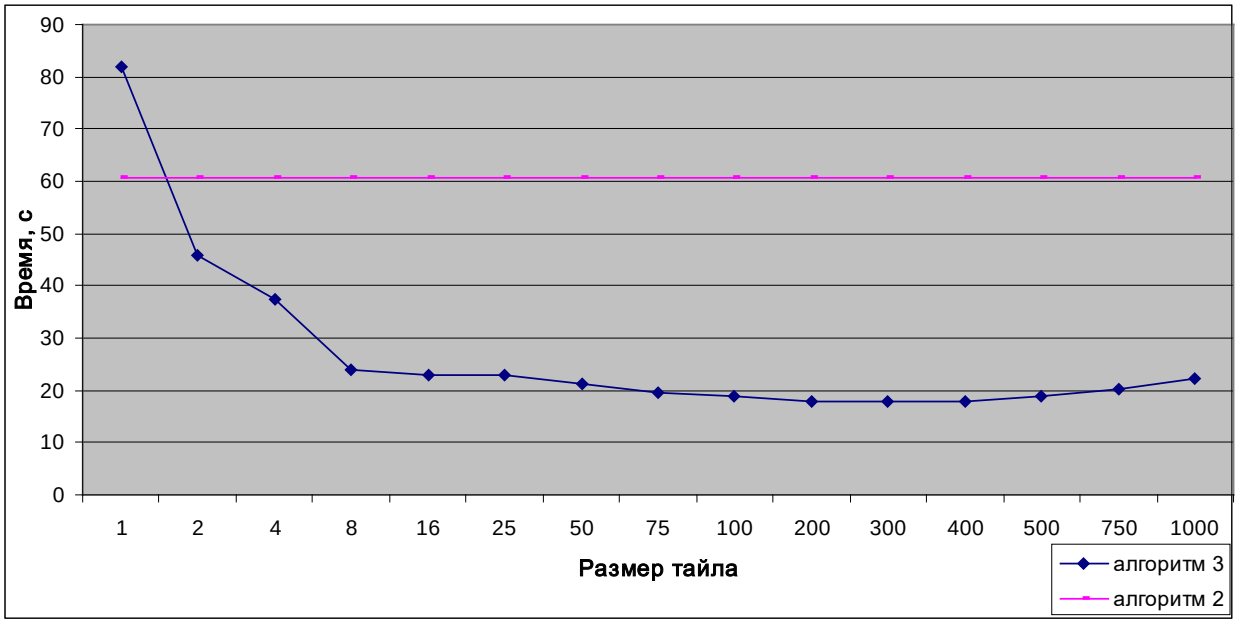

Рисунок 1 – Зависимость времени выполнения программы от размера тайлов для алгоритма 3 при размерах задачи 125x1000x8000

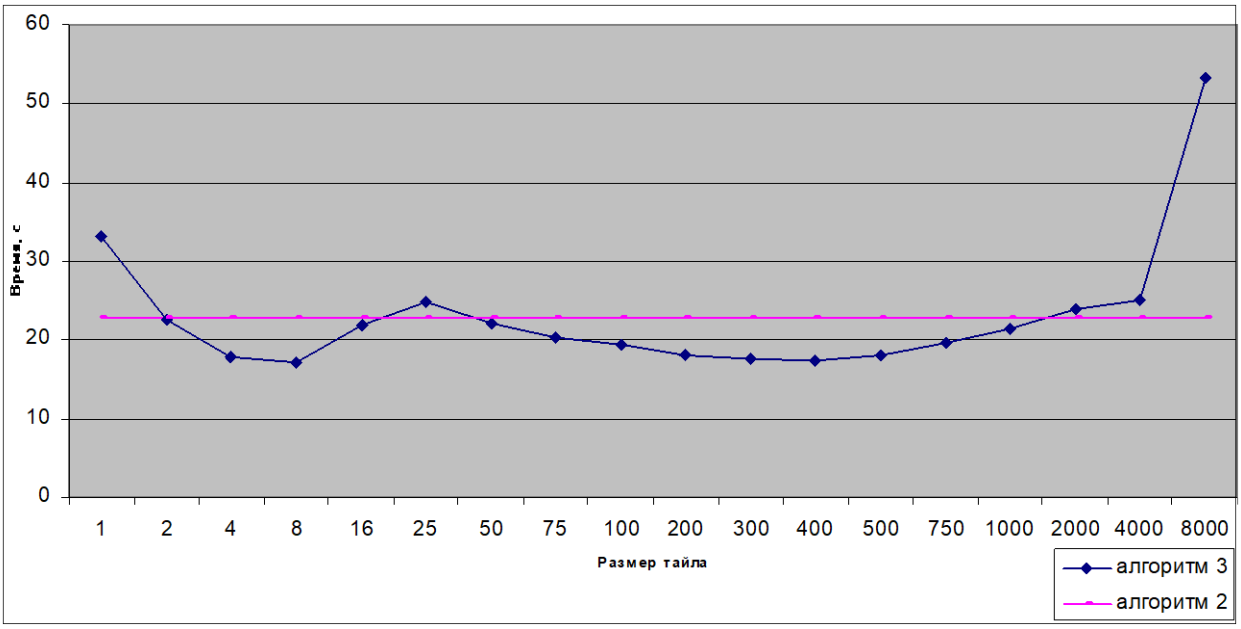

Рисунок 2 – Зависимость времени выполнения программы от размера тайла для алгоритма 3 при размерах задачи 125x8000x1000

Приведем характеристики кэша. Размер кэша первого уровня: 64 Кб двухвходовой ассоциативный кэш для инструкций, 64 Кб двухвходовой ассоциативный кэш для данных. Рамер кэша второго уровня: 512 Кб шестнадцативходовой ассоциативный кэш для данных и инструкций.

Разницу между временем реализации алгоритмов 2 и 3 на Рис. 1 можно объснить тем, что в алгоритме 2 на вхождении $(b,S_1,1)$ на каждой новой итерации алгоритма использовались элементы новой строки. Через 8 000 итераций используемые данные и данные, которые вместе с ними перемещались в кэш, уже будут находиться только в памяти с более медленным дотупом. В алгоритме 3 на всех вхождениях, в том числе и на вхождении $(b,S_1,1)$ большинство обращений к элементам массивов происходит до их исчезновения из памяти с быстрым доступом.

Незначительное отличие времени реализации алгоритмов 2 и 3 на Рис. 2 объсняется тем, что в алгоритме 2 на вхождении $(b,S_1,1)$, несмотря на теоретическое отсутствие локализации данных и использование на каждой новой итерации алгоритма элементов новой строки, через 1 000 итераций используемые данные и данные, которые вместе с ними перемещались в кэш, все еще будут находиться в памяти с быстрым дотупом.

Алгоритм 4 (получен из алгоритма 2 разбиением по циклу i):

do $i^{gl} = 0, Q_1-1$
  do $j = 1, N_2$
    do $k = 1, N_3$
      do $i = 1 + i^{gl} r_1, \min((i^{gl}+1)r_1, N_1)$
        S_1 $c(i,j) = c(i,j) + a(i,k)\, b(k,j)$

do $k = 1, N_3$

do $i = 1 + i^{gl} r_1, \min((i^{gl}+1)r_1, N_1)$

S_2 $x(i,k) = x(i,k) + c(i,j)\, d(j,k)$

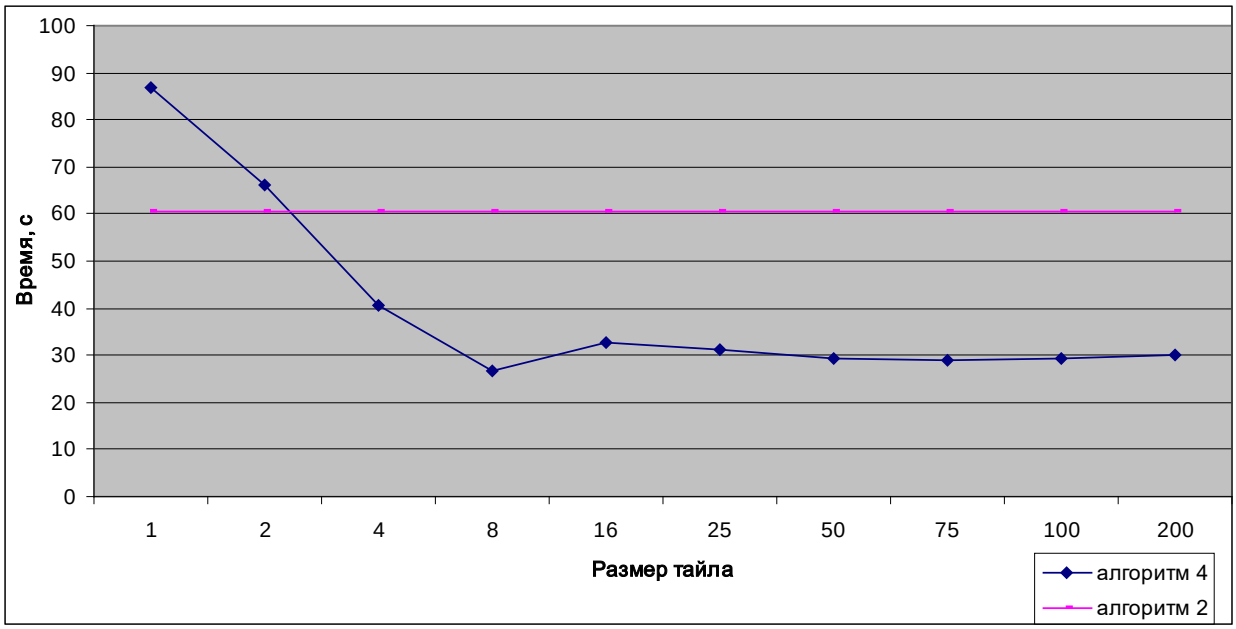

Рисунок 3 – Зависимость времени выполнения программы от размера тайла для алгоритма 4 при размерах задачи 125x1000x8000

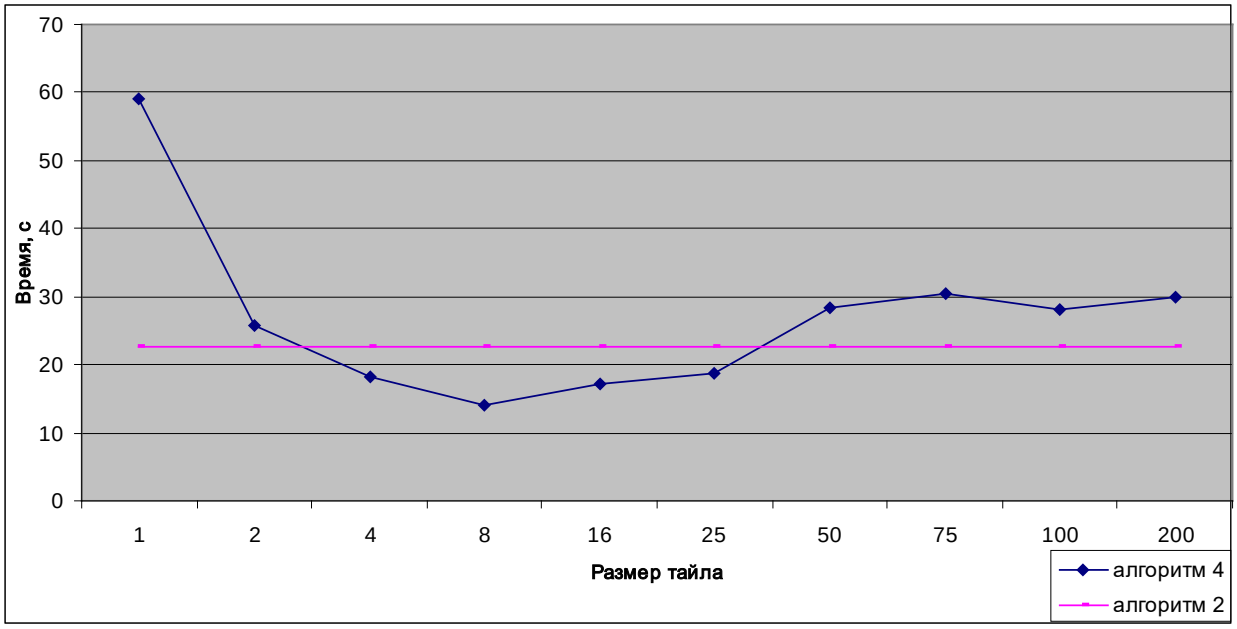

Рисунок 4 – Зависимость времени выполнения программы от размера тайла для алгоритма 4 при размерах задачи 125x8000x1000

При разбиение по циклу *i*, как и при разбиениии по циклу *j*, алгоритм выполняется значительно быстрее, если N_3достаточно большое.

Сравним время вычислений операций алгоритма после двухуровневого и после одноуровневого тайлинга по циклу *j*. Пусть 200-20 – размеры тайлов (большой-малый тайлы). Тогда время вычислений (округлено до секунд) равно:

13 – двухуровневый тайлинг, 18 – одноуровневый тайлинг

Приведем еще некоторые результаты вычислительных экспериментов:

250-25	500-25	500-20
14	14	13

Пример 3. Рассмотрим алгоритм 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.

Следующий алгоритм получен с помощью разбиения цикла j, $Q_2 = \lceil (N+P-1)/r_2 \rceil$.

```
do  k=1,N
    l(1)=1/a(1,k)
    do  p = 2,N
        l(p)= –a(p,k)
    enddo
    do  j^gl = 0, Q2 –1
        do  j = max(2 + j^gl r2 , k +1) , min(1+( j^gl + 1)r2, N + P)
            u(j)=a(1,j)l(1)
        enddo
        do  i = 2, N
            do  j = max(2 + j^gl r2 , k +1) , min(1+( j^gl + 1)r2, N + P)
                a(i–1,j)=a(i,j)+l(i)u(j)
            enddo
```

```
            enddo
            do  j = max(2 +  j^gl  r_2 , k +1) , min(1+( j^gl  + 1)r_2, N + P)
                a(N,j)=u(j)
            enddo
        enddo( j^gl )
    enddo(k)
```

Приведем результаты вычислительных экспериментов; на рисунках исходный алгоритм Гаусса-Жордана указан как алгоритм 3, после применения тайлинга – как алгоритм 4.

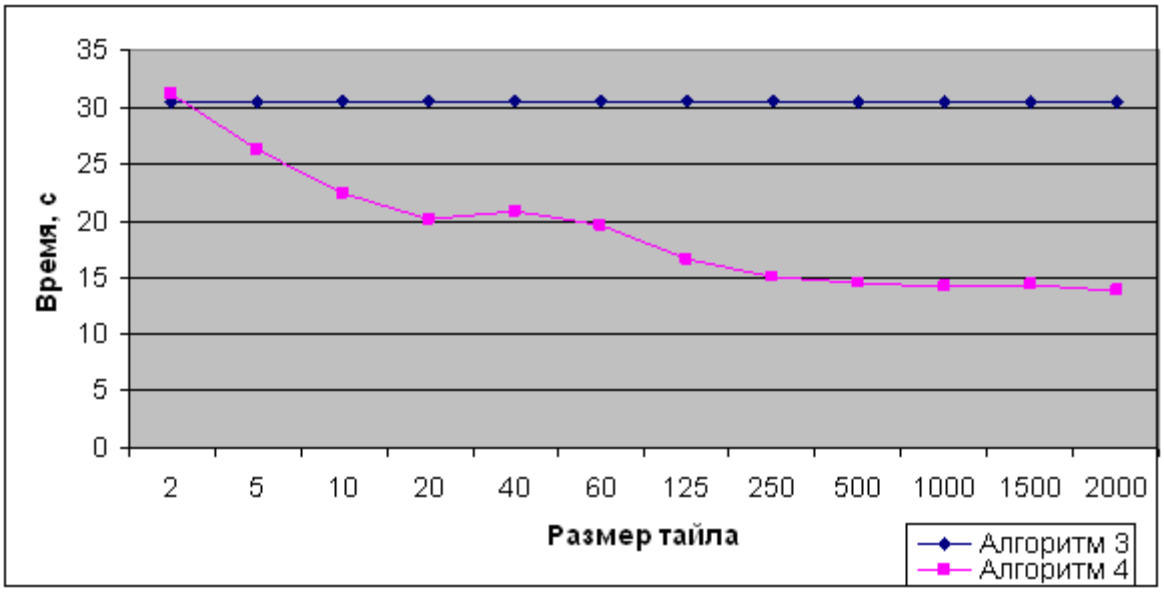

Рисунок 3 – Зависимость времени выполнения программы от размера тайла для алгоритма Гаусса-Жордана при N=1000 и P=1000

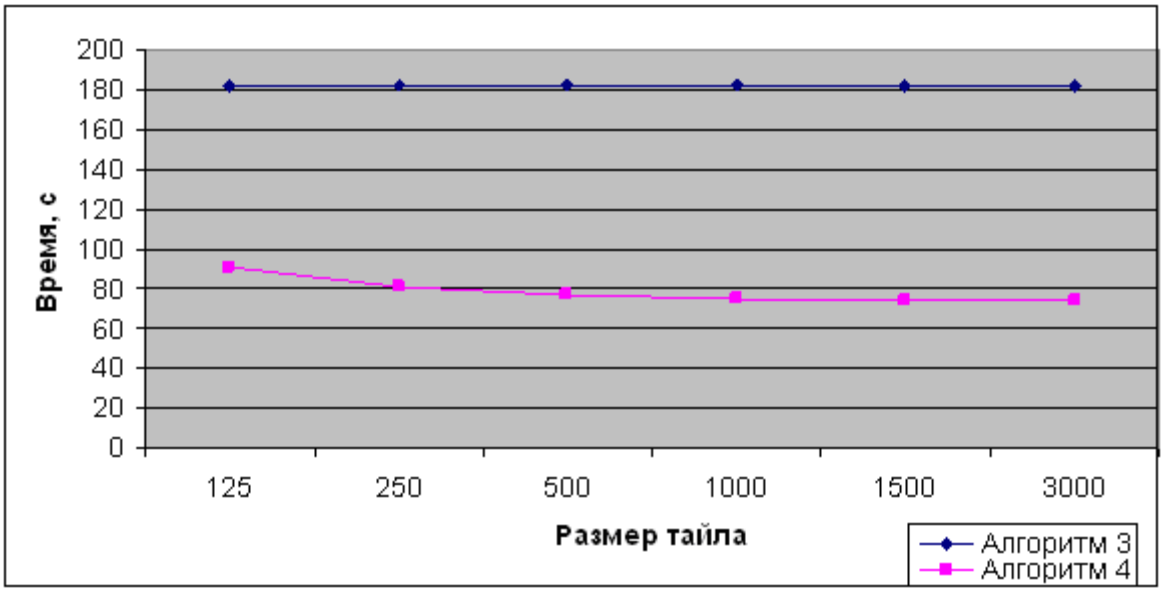

Рисунок 4 – Зависимость времени выполнения программы от размера тайла для алгоритма Гаусса-Жордана при N=2000 и P=1000

В рассматриваемом примере важен не столько тайлинг, сколько тот

факт, что после тайлинга цикл с параметром *j* становится самым внутренним.

Список использованных источников

1. Гервич Л.Р., Штейнберг Б.Я., Юрушкин М.В. Разработка параллельных программ с оптимизацией использования структуры памяти, Ростов-на-Дону, изд-во Южного федерального университета, 2014, 120 с.
2. Воеводин Вл.В., Воеводин Вад.В. Спасительная локальность суперкомпьютеров // Откры тые системы. 2013, № 9. С. 12–15.
3. Kim D.G., Rajopadhye S. Parameterized tiling for imperfectly nested loops // Technical Report CS-09-101, Colorado State University, Department of Computer Science, February 2009. 21 p.
4. 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.
5. Лиходед Н.А. Обобщенный тайлинг // Доклады НАН Беларуси. 2011. Т. 55, № 1. С. 16-21.