Согласование метода гиперплоскостей и структуры памяти

Штейнберг Б.Я., Абу-Халил Ж.М., Аммаев С., Гервич Л.Р., Штейнберг О.Б.

Южный федеральный университет

Введение

Метод гиперплоскостей был предложен 40 лет тому назад Л. Лэмпортом в работе [1]. Впоследствии этот метод подвергался многим модификациям и обобщениям [2], [3] и др.. Этот метод был ориентирован на параллельные компьютеры того времени.

Наибольшее время выполнения в программе занимают гнезда циклов (вложенные циклы). Метод гиперплоскостей иногда позволяет распараллеливать тесное гнездо циклов в программе в тех случаях, когда ни один цикл гнезда не распараллеливается. При таком распараллеливании одновременно выполняются точки некоторых гиперплоскостей в пространстве итераций гнезда циклов.

За сорок лет существования метода гиперплоскостей сменилось много поколений компьютеров. Параллельные компьютеры стали массовыми. Самое существенное изменение, повлиявшее на эффективность высокопроизводительного программного обеспечения, это изменение соотношения времени выполнения арифметических операций и времени доступа к памяти: ежегодно обработка данных в среднем ускоряется на 30%, а обращение к памяти на 9% [4]. Это привело к созданию иерархии памяти в процессорах, с которой следует считаться при разработке быстрых программ.

Для многих задач непосредственное применение метода гиперплоскостей не дает ускорения при распараллеливании, и, даже, может вызвать замедление из-за неэффективного использования кэш-памяти.

В данной статье пойдет речь о сочетании метода гиперплоскостей с оптимизацией использования памяти. Традиционно метод гиперплоскостей применяется для распараллеливания. В данной работе представлено использование метода гиперплоскостей для повышения локальности данных в итерационных методах. Приводятся результаты численных экспериментов для задач выравнивания последовательностей и итерационного метода Якоби решения задачи Дирихле.

Пространство итераций гнезда циклов и метод гиперплоскостей.

В данной работе будем рассматривать ускорение фрагментов кода, содержащих тесные гнезда циклов. Тесное гнездо из n вложенных циклов имеет вид.

```
for(I1=L1; I1<=R1; ++I1)
for(I2=L2; I2<=R2; ++I2)

for(In=Ln; In<=Rn; ++In)
{
```

```
LOOPBODY (I1,I2,...,In)
}
```

Здесь между заголовками вложенных циклов нет операторов, и все циклы завершаются в одном месте. Кроме того, в теле цикла не меняют свои значения счетчики и границы циклов.

Пространством итераций гнезда циклов называется множество значений вектора счетчиков циклов I = (I1, I2, …, In). Если границы счетчиков циклов аффинно зависят от счетчиков вышестоящих циклов, то пространство итераций является выпуклым многогранником (во многих публикациях используется термин «полиэдр»). Если границы счетчиков циклов константы, то пространство итераций является многомерным прямоугольным параллелепипедом. Каждая точка пространства итераций соответствует некоторому допустимому значению вектора счетчиков циклов I = (I1, I2, …, In). Под выполнением точки пространства итераций будем понимать выполнение тела цикла для значений счетчиков, соответствующих этой точке пространства итераций.

Обычное выполнение представленного выше гнезда циклов состоит в выполнении всех точек пространства итераций последовательно в лексикографическом порядке следования соответствующих векторов счетчиков циклов.

Суть метода гиперплоскостей состоит в том, чтобы разбить множество точек пространства итераций на подмножества, лежащие на некоторых параллельных друг другу гиперплоскостях. При этом меняется обход точек пространства итераций, который состоит в переходе к новому гнезду циклов. В новом гнезде внешний цикл просматривает гиперплоскости, а остальные циклы просматривают точки внутри каждой гиперплоскости. Предполагается параллельное выполнение циклов, просматривающих точки гиперплоскостей.

Описанная замена одного гнезда циклов другим корректна не всегда. При такой замене *должен остаться неизменным порядок всех пар обращений ко всем ячейкам памяти, при которых хотя бы одно обращение является записью*.

Пары обращений к ячейкам памяти, при которых хотя бы одно обращение – запись, образуют информационные зависимости. Эти зависимости описываются решетчатыми графами, которые в памяти представляются в виде функций [5] – [25]. Взаимозависимость между графом информационных связей и решетчатым графом программы описана в [26].

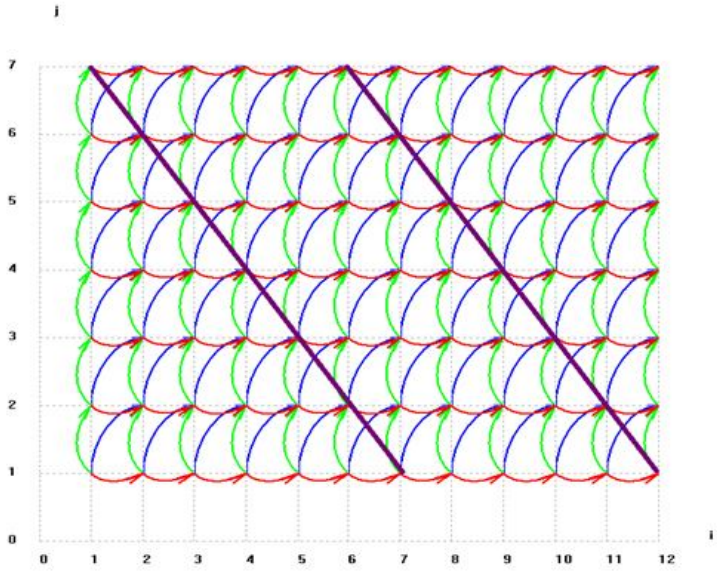

Рисунок 1. Решетчатый граф алгоритма Смита-Уотермана [27] парного выравнивания последовательностей. Граф построен автоматически Оптимизирующей распараллеливающей системой [28]. Пространство итераций разбито на 3 области, в которых различные аффинные границы циклов при методе гиперплоскостей. Точка пространства итераций начала любой дуги по прежнему выполняется раньше точки, соответствующей концу.

Сочетание распараллеливания по методу гиперплоскостей и перехода к блочному коду.

Метод гиперплоскостей в непосредственном виде бывает неэффективен и распараллеливание может привести даже к замедлению. В следующей таблице приводятся результаты численных экспериментов на двуядерном процессоре для задачи выравнивания нуклеотидных последовательностей. Непосредственное применение метода гиперплоскостей и распараллеливание на 2 ядра дает замедление в 6.5 раз по сравнению с исходным классическим алгоритмом Смита-Уотермана. В это же время, если вычисления вести блоками и метод гиперплоскостей применить к блочному коду, то на двух ядрах получится ускорение более чем двукратное. Неэффективность непосредственного метода гиперплоскостей объясняется увеличением количества циклов, усложнением границ изменения счетчиков циклов и усложнением индексных выражений массивов.

Таблица 1. Сравнение эффективности обычного и блочного метода гиперплоскостей.

Результаты измерений		
Алгоритм Смита-Уотермана (последовательный)	Алгоритм Смита-Уотермана + метод Лампорта (параллельный)	Блочный алгоритм + метод Лампорта (параллельный)
1.18	7.6	0.48

Расчеты проводились на Intel Core 2 Duo E7500, Кэш-память 2 уровня - 3MB, тактовая частота - 2.93 GHz, количество ядер - 2. Размер последовательностей – 10 000 нуклеотидов. Размер блока – 1000.

Блочные вычисления демонстрируют высокую эффективность благодаря успешному использованию кэш-памяти. Оптимизация использования кэш-памяти и блочные вычисления (известны еще под названием тайлинг) описаны в работах [29] – [46]. Комбинация метода гиперплоскостей и блочных вычислений для задачи выравнивания нуклеотидных последовательностей ранее рассматривалась в работе [47].

В следующей таблице более детально представлено сравнение ускорений, которые дают переход к блочному коду и распараллеливание по методу гиперплоскостей.

Таблица 2. Сравнение эффективности алгоритмов решения задачи выравнивания последовательностей.

Длина строк	Размер блока	Время, с				
		Алгоритм Смита - Уотермана (последо-вательный)	**Метод гиперплоскостей (последо-вательный)**	**Метод гиперплоскостей (параллельный)**	**Блочный алгоритм (последо-вательный)**	**Блочный Алгоритм + метод гиперпл. (параллельный)**
3173 3214	100	1.6	2.3	2.5	0.8	0.6
4783 4759	100	2.1	2.4	2.8	1.3	1.1
9566 9518	200	5.8	6.9	5.6	5.0	3.0
15796 15608	200	14.8	31.1	18.6	12.8	7.1
111640 111630	550	недостаточно памяти	недостаточно памяти	недостаточно памяти	600	350
223280 223260	800	недостаточно памяти	недостаточно памяти	недостаточно памяти	2522	1463
307662 310525	1000	недостаточно памяти	недостаточно памяти	недостаточно памяти	недостаточно памяти	2442

Использование метода гиперплоскостей для временнОй локализации данных в итерационных алгоритмах.

В предыдущем параграфе приводились результаты численных экспериментов, демонстрирующие неэффективность непосредственного метода гиперплоскостей Лэмпорта. Однако это не всегда так. Метод гиперплоскостей может давать ускорение даже без распараллеливания за счет повышения временной локализации данных. Продемонстрируем это на итерационном методе Якоби для решения задачи Дирихле.

Задача Дирихле представляет собой задачу поиска решений дифференциального уравнения Лапласа

$$\begin{cases} \frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} = f(x,y), & (x,y) \in D, \\ u(x,y) = g(x,y), & (x,y) \in D^0, \end{cases}$$

при заданных значениях g(x,y) искомой функции u=u(x,y) на границе D^0 области D. В данном случае, для простоты, функция f - тождественный ноль.

Метод Якоби заключается в том, что элемент u(i, j) массива, представляющего искомую функцию, при численном решении рассчитывается по формуле:

```
for (k=L1; k<=R1; ++k)
for (i=L2; i<=R2; ++i)
for (j=Ln; j<=Rn; ++j)
   u(i, j) = ( u(i+1, j)+ u(i-1, j)+ u(i, j+1)+ u(i, j-1) ) / 4
```

Классический метод гиперплоскостей применяется к совокупности двух самых вложенных циклов. Как было показано на задаче выравнивания двух последовательностей, на современных компьютерах это приводит к замедлению даже при распараллеливании, если не использовать переход к блочным вычислениям.

Идея метода данной работы состоит в том, чтобы, не завершив одной итерации алгоритма, начинать вычисления другой итерации. Это позволит в программе ближе расположить вычисления с одними и теми же данными, то есть повысить временную локализацию данных. Это равносильно применению метода гиперплоскостей к совокупности всех трех циклов. Пространство итераций при этом разбивается на точки, лежащие на гиперплоскостях, ортогональных вектору (1, 1, 1).

Проведены численные эксперименты для итерационного алгоритма решения двумерной задачи Дирихле. В этом случае пространство итераций алгоритма является трехмерным. Информационные зависимости в этом пространстве итераций направлены параллельно осям координат. Рассмотрены 2 семейства гиперплоскостей в трехмерном пространстве итераций, которые в сечении с вертикальными координатными плоскостями образуют угол, тангенс которого равен 1 (45 градусов), а в пересечении с горизонтальной координатной плоскостью получается прямая наклоненная к первой координатной оси под углами с тангенсом 1 и 2 соответственно.

Приводимые в данной работе рассуждения допускают обобщения на итерационные алгоритмы решения других линейных эллиптических уравнений второго порядка, в том числе и трех переменных:

$$A * \frac{\partial^2 u}{\partial x^2} + B * \frac{\partial^2 u}{\partial y^2} + C * \frac{\partial^2 u}{\partial z^2} + 2 * D * \frac{\partial^2 u}{\partial x \partial y} + 2 * E * \frac{\partial^2 u}{\partial x \partial z} + 2 * F * \frac{\partial^2 u}{\partial y \partial z} = 0$$

3 Результаты расчетов:

Для определения разницы в скорости выполнения алгоритмов с разными порядками вычислений метода Якоби решения задачи Дирихле было выполнено несколько тестов для разных входных данных и взяты минимальные значения:

Таблица 3 – зависимость скорости выполнения программы при стандартном порядке вычислений и вычислений по гиперплоскостям при размере матрицы 400x400 от числа итераций выполнения алгоритма.

Итерации	50	100	300	500	700
Стандартно (миллисекунды)	43	78	250	418	578
По гиперплоскостям с тангенсом угла 2 (миллисекунды)	13	26	78	125	187
По гиперплоскостям с углом 45 градусов	12	26	76	125	172

Таблица 4 – зависимость скорости выполнения программы при стандартном порядке вычислений и вычислений по гиперплоскостям при размере матрицы 800x1000 от числа итераций выполнения алгоритма.

Итерации	50	100	300	500	700
Стандартно (миллисекунды)	375	593	1438	2297	3157
По гиперплоскостям с тангенсом угла 2 (миллисекунды)	249	328	672	1078	1640
По гиперплоскостям с углом 45 градусов	234	328	656	984	1344

Таблица 5 – зависимость скорости выполнения программы при стандартном порядке вычислений и вычислений по гиперплоскостям при размере матрицы 2000x2000 от числа итераций выполнения алгоритма.

Итерации	50	100	300	500	700
Стандартно (миллисекунды)	2906	5687	15344	22579	27361
По гиперплоскостям с тангенсом угла 2 (миллисекунды)	3219	6266	17438	26719	33657
По гиперплоскостям с углом 45 градусов	3203	6234	17172	26391	33750

Из полученных результатов видно, что при малых размерах ширины или длины матрицы получаем выигрыш в скорости выполнения у алгоритма с вычислениями по гиперплоскостям, в среднем, от 20 до 70%. Однако при больших размерах обоих параметров матрицы стандартный порядок вычислений выполняется быстрее. Это происходит из-за того, что при больших размерах обоих параметров длина диагонали настолько велика, что кэш-памяти не хватает для хранения всех ее элементов, и скорость вычислений на этой диагонали падает.

Разбиение пространства итераций на части.

Приведенные выше расчеты показали эффективность метода гиперплоскостей для итерационного метода Якоби решения задачи Дирихле при сетках ограниченной размерности. Кроме того, даже при сетках малой размерности ускорение достигается при большом количестве итераций, хотя достаточная точность решений иногда достигается и при малом количестве итераций. При сетках большой размерности этот метод дает замедление.

Для решения описанной проблемы пространство итераций можно разбивать на трехмерные фигуры, которые в основании имеют полосы, лежащие в нижней координатной плоскости. Следует отметить, что данные фигуры, на которые разбивается пространство итераций, не являются прямоугольными параллелепипедами. Эти фигуры являются прямыми призмами, лежащими на боковой стороне. К каждой (длинной) трехмерной фигуре метод гиперплоскостей применяется отдельно. Количество итераций должно быть не больше ширины опорной полосы.

Численные эксперименты на компьютере с процессором Intel Core i7 - 4700HQ; частота: 2400 МГц. Объем L3 кэш памяти: 6 MB. Размерность сетки 20000*20000. Сетка разбивалась на полосы и вычисления проводились по очереди по полосам. Ширина полосы измеряется количеством точек сетки, время в миллисекундах. Гиперплоскости в каждой трехмерной фигуре рассмотрены ортогональные вектору (1, 1, 1). Время обычного вычисления для 8 итераций: 19020, для 16 итераций: 38924, для 32 итераций: 76388

Таблица 6. Зависимость ускорения вычислений от ширины полос (длинных трехмерных фигур), на которые разбивается пространство итераций.

Ширина полосы (количество итераций)	8 итераций Время метода гипер-плоскостей	Уско-рение	16 итераций Время метода гипер-плоскостей	Уско-рение	32 итерации Время метода гипер-плоскостей	Уско-рение
10	8320	2,29	–	–	–	–
20	7414	2,57	15802	2,47	–	–
30	8093	2,35	15172	2,56	–	–
40	8093	2,35	14917	2,62	30430	2,510286
50	8156	2,33	15793	2,45	30821	2,47844
60	8133	2,34	15701	2,46	30845	2,476512
70	8157	2,33	16141	2,42	30922	2,470345
80	8250	2,31	16305	2,39	31649	2,413599

Из приведенной таблицы видно, что данный метод дает ускорение примерно в 2.5 раза при последовательном выполнении.

Представленный подход может быть применен к итерационным алгоритмам численного решения других задач с другими схемами в основе (семиточечные шаблоны и пр.).

Начало дальнейших исследований: одновременное использование метода гиперплоскостей для временной локализации данных и для распараллеливания в итерационных алгоритмах.

Возникает естественная задача: усилить ускорение, полученное за счет локализации данных, ускорением, получаемым за счет распараллеливания.

Точки пространства итераций, лежащие на различных длинных фигурах, не всегда могут выполняться параллельно. Для применения параллельных вычислений рассмотренные в предыдущем параграфе длинные фигуры разбиваются на части меньшей длины. Пока этот подход недал ускорение за счет распараллеливания с использованием OpenMP.

Для распараллеливания на кластере с распределенной памятью с помощью MPI проведен пока один положительный эксперимент. Рассмотрена двумерная задача Дирихле с сеткой 10000*10000 и количеством итераций 32.
Время работы стандартного алгоритма 36120000 мкс
Время работы последовательного метода гиперплоскостей с разбиением на полосы (метод, предлагаемый в данной работе) - 20210000 мкс.
Параллельный метод гиперплоскостей при разбиении пространства итераций на полосы с перекрытием ([43], [46]): 2 узла – 10173455 мкс, 4 узла - 4981229 мкс, 8 узлов - 2534557 мкс.

Заключение

Результаты работ данного проекта направлены на повышение эффективности параллельных программ. Рассмотрен анализ взаимодействия метода гиперплоскостей с кэш-памятью. Представлено использование метода гиперплоскостей в итерационных алгоритмах в новом качестве: для повышения локальности данных, что без распараллеливания дает ускорение более, чем в два раза.

Литература.

1. Lamport L. The parallel execution of DO loops// Commun. ACM.- 1974.- v.17, N 2, p. 83-93.
2. Бабичев А.В., Лебедев В.Г. Распараллеливание программных циклов./ Программирование// 1983, N 5, с. 52-63.
3. Fernandez Augustin, Llaberia Jose M., Valero-Garcia Miguel. Loop Transformation Using Nonunimodal Matrices // IEEE Transactions on Parallel and Distributed Systems, 1995, vol. 6, № 8, pp. 832-840.
4. Корнеев В.В. Проблемы программирования суперкомпьютеров на базе многоядерных мультитредовых кристаллов // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции (21-26 сентября 2009 г., г. Новороссийск).– М.: Изд-во МГУ, 2009. - 524 с.
5. Воеводин В.В. Математические модели и методы в параллельных процессах// М.: Наука, гл. ред. физ.-мат. лит., 1986, 296 с.
6. Воеводин В.В. Математические основы параллельных вычислений, М., МГУ, 1991, 345 с.
7. Воеводин В.В. Воеводин Вл.В. Параллельные вычисления, С-Петербург «БХВ-Петербург», 2002, 599 с.
8. www.parallel.guru.ru 10.12.2010
9. Лиходед Н.А. Распределение операций и массивов данных между процессорами.// Программирование, 2003, №3, с. 73-80.

10. Штейнберг Б.Я. Оптимальные параллельные переразмещения двумерных массивов.//Программирование, N 6, 1993 г. с.81-88.
11. Штейнберг Б.Я. Оптимальные параллельные переразмещения многомерных массивов при параллельных вычислениях// Международная научно-техническая конференция <Интеллектуальные многопроцессорные системы>/ 1-5 сентября, 1999, Таганрог, Россия, Сборник трудов, с.151-155.
12. Штейнберг Б.Я. Подстановка и переименование индексных переменных в многомерных циклах.// Известия вузов. Северокавказский регион. Юбилейный выпуск. 2002 г., с. 94-99.
13. Штейнберг Б.Я. Оптимизация размещения данных в параллельной памяти. Ростов-на-Дону, изд-во Южного федерального университета, 2010, 255 с.
14. Штейнберг Р.Б., Вычисление задержки в стартах конвейеров для суперкомпьютеров со структурно процедурной организацией вычислений// Искусственный интеллект. Научно-теоретический журнал. Институт проблем искусственного интеллекта НАНУ. Украина, Донецк, ДонДИШИ, “Наука и Освита”, № 4, 2003, с. 105-112.
15. Штейнберг Р. Б. Отображение гнезд циклов на многоконвейерную архитектуру // Программирование, 2010, № 3. Steinberg R. Mapping loop nests to multipipelined architecture // MAIK Nauka/Interperiodica distributed exclusively by Springer Science+Business Media LLC. 2010. Vol 36, №3May, P 177–185.
16. Шульженко А.М. Автоматическое определение циклов ParDo в программе // Известия вузов. Северокавказский регион. Естественные науки. Приложение 11’05. с. 77-88.
17. Шульженко А.М. Исследование информационных зависимостей программ для анализа распараллеливающих преобразований. Диссертация на соискание ученой степени кандидата технических наук. РГУ. 2006 г. 200 с.
18. Шульженко А М . Преимущества определения информационных зависимостей в программе с помощью решетчатых графов // XIII Международная 190 конференция «Математика . Экономика . Образование .». Тезисы докладов . Ростов н /Д , 2005 - 95 с. - ISBN 5-94153-097-8. с. 137 54.
19. Гуда С.А., Операции над представлениями кусочно-квазиафинных функций в виде деревьев. Информатика и ее применение, 2013. т. 7. Вып 1. стр 26-37
20. Collard Jean-Francois, Feautrier Paul, Risset Tanguy Construction of DO Loops from Systems of Affine Constraints//Laboratoire de I’Informatique du Parallelisme, Lyon, Institut IMAG, Research Report № 93-15, May 1993, p. 1-26.
21. Feautrier Paul Parametric Integer Programming// Laboratoire MASI, Institut Blaise Pascal, Universite de Versailles St-Quentin, 1988, p. 25.
22. Feautrier Paul Data Flow Analysis for Array and Scalar References// International Journal of Parallel Programming/V. 20, N 1, Feb. 1991, p. 23-53.
23. Feautrier Paul. Some efficient solutions no the affine scheduling problem. Part 1. One-dimentional Time. //”International journal of Parallel Programming”, V. 21, № 5, Octouber, 1992.
24. Feautrier Paul. Some efficient solutions no the affine scheduling problem. Part 2. Multidimentional Time. // Technical Report, IBP/MASI, Number 92.78, Octouber, 1992, 28 p.
25. Pugh W. The Omega Test: a fast and practical integer programming algorithm for dependence analysis.// Comm. Of the ACM, August, 1992.
26. Штейнберг Б. Я. О взаимосвязи между решетчатым графом программы и графом информационных связей. «Известия ВУЗов. Северокавказский регион. Естественные науки», №5, 2011 г.
27. Gusfield, D.: Algorithms on Strings, Trees and Sequences. Cambrige University Press, 556 p. (1997)
28. Оптимизирующая распараллеливающая система www.ops.rsu.ru (дата обращения 01.04.2013)
29. Касперский К. Техника оптимизации программ. Эффективное использование памяти. - СПб.: БХВ-Петербург, 2003 г. – 456 с.
30. Lim Amy W., Cheong Gerald I. and Lam Monika S.. An Affine Partitioning Algorithm to Maximize Parallelism and Minimize Communication. Stanford University, Stanford, CA 94305, 10 p.

31. Lim A.W., Lam M.S. Cache Optimizing with Affine Partitioning. Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing, Portsmouth, Virginia, March, 2001. 14 p.
32. Lim Amy W., Lam Monika S. Maximizing Parallelism and Minimizing Synchronization with Affine Partitions. Parallel Computing, 24, 1998, p. 445-475.
33. Lim Amy W., Lam Monika S. Maximizing Parallelism and Minimizing Synchronization with Affine Transformations. 14 p.
34. Kulkurni D., ,Stumm M. Loop and Data Transformations: A Tutorial. Technical Report CSRI 337, Computer Systems Research Institute, Univercity of Toronto, June 1993, 53 p.
35. V.Bandishti, I.Pananilath, U.Bondugula Tiling Stansil Computations to Maximise Parallelism SC12, November 10-16, 2012, Salt Lake City, Utah, USA 978-1-4673-0806-9/12/$31.00 c 2012 IEEE
36. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем (2003, 2 изд.). Н.-Новгород, ННГУ.
37. Арыков С.Б., Малышкин В.Э. Система асинхронного параллельного программирования «Аспект»// Вычислительные методы и программирование. – 2008. – Т. 9. №1. – С. 205-209.
38. Штейнберг Б. Я. Блочно рекурсивное параллельное перемножение матриц. Известия ВУЗов. Приборостроение, т. 52, №10, 2009 г., с. 33-41.
39. Штейнберг Б.Я. Блочно-аффинные размещения данных в параллельной памяти // Информационные технологии. – 2010. – М.: из-во «Новые технологии». – №6. – С. 36-41.
40. Neungsoo Park, Wenheng Liu, Viktor K. Prasanna, Cauligi Raghavendra. Efficient Matrix Multiplication Using Cache Conscious Data Layouts, Department of Electrical Engineering-Systems, University of Southern California, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.3679
41. José R. Herrero, Juan J. Navarro. Using Non-canonical Array Layouts in Dense Matrix Operations. Applied Parallel Computing. State of the Art in Scientific Computing Lecture Notes in Computer Science Volume 4699, 2007, pp 580-588
42. Kazushige Goto, Robert A. van de Geijn. Anatomy of High-Performance Matrix Multiplication. ACM Trans. Math. Softw., Vol. 34, No. 3. (May 2008), pp. 1-25.
43. Гервич Л.Р., Штейнберг Б.Я., Юрушкин М.В. Программирование экзафлопсных систем// «Открытые системы. СУБД», 2013, №8, стр. 26-29, журнал выходит в США : Open Systems Journal, (ISSN 1028-7493)
44. Юрушкин М. Программа перемножения матриц рекордной производительности http://ops.opsgroup.ru/downloads/dgemm.zip (дата обращения 12.12.2013)
45. Штейнберг Б. Я. Блочно рекуррентное размещение матрицы для параллельного выполнения алгоритма Флойда. «Известия ВУЗов. Северокавказский регион. Естественные науки», №5, 2010 г. с.31-33 0.026
46. Гервич Л.Р., Штейнберг Б.Я. Параллельное итерационное умножение ленточной матрицы на вектор. Труды научной школы И.Б. Симоненко. Сборник статей. Ростов-на-Дону, изд.-во ЮФУ, 2010. с. 58-66.
47. Абу-Халил Ж.М., Морылев Р.И., Штейнберг Б.Я. Параллельный алгоритм глобального выравнивания с оптимальным использованием памяти // Современные проблемы науки и образования. – 2013. – № 1; (Электронный журнал http://www.science-education.ru/107-8139).