ЛИТЕРАТУРА

1. *Скворцов А.В.* Триангуляция Делоне и её применение. – Томск: Изд-во Том. ун-та, 2002. – 127 с.
2. *J.Mark Ware.* A procedure for automatically correcting invalid flat triangles occuring in triangulated contour data // Computers & Geosciences. – 1998. – V. 24. – No. 2. – P. 141–150.
3. ARC/INFO User's Guide: Surface Modeling With TIN – ESRI Inc., NY, 1992. – 240 p.
4. *Brandli M.* A triangulation-based method for geomorphological surface interpolation from contour lines // Proceeding of 3rd European Conference and Exhibition on GIS. – Münich, 1992. – V. 1. – – P. 691–700.
5. *Vozenilek V.* Generating surface models using elevations digitised from topographical maps // Proc. of 5th European Conference and Exhibition on GIS. – Utrecht, 1994. – V. 1. – P. 972–982.
6. *Fuchs H., Kedem Z.M., Uselton S.P.* Optimal surface reconstruction from planar contours // Communications of the ACM. – 1977. – V. 20. – No. 10. – P. 693–702.
7. *Ekoule A.B., Peyrin F.C., Odet L.* A triangulation algorithm from arbitrary shaped multiple planar contours // ACM Transactions on Graphics. – 1991. – V. 10. – No. 2. – P. 182–199.
8. *Christensen A.H.J.* Fitting a triangulation to contour lines // Proc. 8th Intern. Symp.of Computer Assisted Cartography. – Baltimore, Maryland, 1987. – P. 57–67.
9. *Ganthapy S., Dennehy T.G.* A new general triangulation method for planar contours // ACM Computer Graphics. – 1982. – V. 13. – No. 3. – P. 311–319.
10. *Tipper J.C.* A method and Fortran program for the computerized reconstruction of three dimensional objects from serial sections // Computer & Geosciences. – 1977. – V. 3. – No. 4. – P. 579–599.
11. *Фукс А.Л., Костюк Ю.Л.* Построение цифровой модели рельефа местности на основе структурных линий и высотных отметок // Вестник ТГУ. – 2003. – Т. 280.
12. *Петренко Д.А., Скворцов А.В., Куленов Р.О.* Сравнение триангуляций с помощью хеш-функций // Вестник ТГУ. – 2003. – Т. 280. – С. 306–309.

УДК 519.688

ОБЗОР АЛГОРИТМОВ ПОСТРОЕНИЯ ВЫПУКЛОЙ ОБОЛОЧКИ НА ПЛОСКОСТИ

Р.В. Чаднов, А.В. Скворцов, Н.С. Мирза

Томский государственный университет

E-mail: chadnov@newmail.ru; skv@indorsoft.ru; mirza@indorsoft.ru

Рассматриваются различные алгоритмы построения выпуклой оболочки на плоскости. Проводится сравнение различных алгоритмов. Показываются преимущества и недостатки алгоритмов на различных наборах данных. Предлагается использовать комбинированный алгоритм для достижения максимальной эффективности при построении выпуклой оболочки вне зависимости от видов исходных данных.

Ключевые слова: *выпуклые оболочки, вычислительная геометрия.*

Задача построения выпуклых оболочек является одной из центральных для вычислительной геометрии. Она позволяет разрешить целый ряд других, иногда с первого взгляда не связанных с ней вопросов: построение диаграмм Вороного, построение триангуляций и т.д. Построение выпуклой оболочки конечного множества точек на плоскости довольно широко исследовано и имеет множество приложений в распознавании образов, обработке изображений, в задаче раскроя и компоновки материалов. Очень широко алгоритмы построения выпуклой оболочки используются в геоинформатике и геоинформационных системах.

В настоящее время известно достаточно большое число алгоритмов построения выпуклой оболочки, однако существуст проблема, связанная с недостаточным количеством работ, посвященных анализу и/или обзору этих алгоритмов.

В настоящей работе выполнен анализ наиболее распространенных современных алгоритмов построения выпуклой оболочки. Кроме того, в работе приведены результаты сравнения скорости работы алгоритмов в различных условиях.

1. Постановка задачи

Понятие выпуклой оболочки определяется следующим образом [1]:

Определение 1. *Выпуклой оболочкой* (ВО) множества точек S называется наименьшее выпуклое множество, содержащее S.

В случае, когда S – конечное множество точек на плоскости, это определение можно представить наглядно. Предположим, что множество точек охвачено большой растянутой резиновой лентой. Эта лента имеет форму выпуклой оболочки.

Однако приведенное выше простое определение выпуклой оболочки неконструктивно. Для более формального описания необходимо ввести некоторые дополнительные понятия. Будем рассматривать 2-мерное евклидово пространство E^2.

Определение 2. Область D, принадлежащая пространству E^2, будем называть *выпуклой*, если для любой пары точек d^1 и d^2, принадлежащих D, отрезок d^1d^2 целиком принадлежит D.

Определение 3. *Выпуклой оболочкой* множества точек S, принадлежащих пространству E^2, называется граница наименьшей выпуклой области в E^2, которая охватывает S.

Выпуклая оболочка множества точек L обычно обозначается как $CH(L)$ [1].

Задача построения выпуклой оболочки обычно ставится в двух вариантах.

Задача *CH*1. В E^2 задано множество S, содержащее N точек. Требуется определить те из них, которые являются вершинами выпуклой оболочки $CH(S)$.

Задача *CH*2. В E^2 задано множество S, содержащее N точек. Требуется построить их выпуклую оболочку (т.е. найти полное описание границы $CH(S)$).

Задача $CH1$ обычно возникает в теории, а на практике она используется редко ввиду низкой практической полезности результатов решения этой задачи.

2. Алгоритм обхода Грэхема

Алгоритм Грэхема часто называют ([1]) первым алгоритмом вычислительной геометрии. Этот алгоритм был описан в одной из первых работ Грэхема [2], специально посвященной вопросу разработки эффективных геометрических алгоритмов. Он показал, что, выполнив предварительно сортировку точек, крайние точки можно найти за линейное время. Использованный им метод стал очень мощным средством в области вычислительной геометрии. Рассмотрим этот алгоритм.

Пусть центр координат находится в какой-нибудь внутренней точке выпуклой оболочки. Упорядочим точки относительно полярного угла, а если таковые совпадают, то относительно расстояния от центра координат. Так как обе точки лежат на одной прямой, проходящей через центр координат, то для сравнения нам нет необходимости вычислять расстояние, а можно сравнивать сумму абсолютных значений координат.

Отсортированные точки следует поместить в двусвязный список, так как внутренние точки принадлежат некоторому треугольнику (Opq), где p и q – соседние вершины точке выпуклой оболочки. Суть алгоритма состоит в последовательном просмотре отсортированного списка и удалении вершин, не входящих в выпуклую оболочку на основании измерения углов. Оставшиеся точки будут являться вершинами выпуклой оболочки.

Просмотр начнем с точки, являющейся вершиной выпуклой оболочки. Для этого необходимо взять точку с минимальной абсциссой, а если их несколько, то с минимальной ординатой, и пометить её как начальную. После чего обходим список, начиная с начальной точки, против часовой стрелки и проверяем внутренний угол для текущей точки. Если он больше либо равен π, то удаляем точку, а иначе переходим к следующей (рис. 1). Так как за каждый просмотр мы или удаляем одну точку, или переходим к следующей, а просмотр заканчиваем при достижении начальной точки, которая не удалится, то мы выполняем не более N шагов.

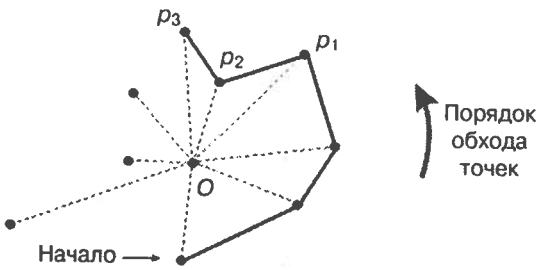

Рис. 1. Начало обхода точек в методе Грэхема. Вершина p_2 удаляется, если угол $p_1p_2p_3$ оказывается вогнутым

Трудоёмкость алгоритма Грэхема составляет $O(n\log n)$.

К недостаткам алгоритма Грэхема следует отнести то, что: 1) теоретически алгоритм является оптимальным в худшем случае, однако он не оптимален в среднем; 2) алгоритм не является открытым, т.е. для его работы необходимо априорное знание всего набора точек; 3) неизвестны обобщения алгоритма на пространства размерности больше двух; 4) использование тригонометрических операций значительно снижает скорость работы алгоритма.

3. Алгоритм монотонных цепочек Эндрю

Алгоритм Эндрю, описанный в [3], является модификацией алгоритма Грэхема, которая использует лексикографическое упорядочение точек по координатам. Это является большим преимуществом, так как в этом случае не приходится использовать вещественные числа и тригонометрические операции. Алгоритм Эндрю по отдельности вычисляет верхнюю и нижнюю оболочки из последовательных цепей точек.

Сначала алгоритм находит самую левую и самую правую точки исходного множества. После этого точки исходного множества разбиваются на два подмножества (нижнее и верхнее) в зависимости от того, по какую сторону они располагаются от прямой, образованной самой левой и самой правой точками. Затем по отдель-

ности строится выпуклая оболочка верхней и нижней половин. Алгоритм, используемый для этого, практически идентичен тому, что используется в алгоритме Грэхема. Точки упорядочиваются в соответствии с возрастанием абсциссы, и к полученным последовательностям применяется метод обхода Грэхема (рис. 2).

В сущности, алгоритм Эндрю является частным случаем алгоритма Грэхема, когда центральная точка выбирается бесконечно удаленной в отрицательном направлении по оси ординат, так что в этом случае упорядоченность по абсциссе совпадает с упорядоченностью по полярному углу.

Как и алгоритм Грэхема, этот алгоритм имеет трудоёмкость $O(n \log n)$.

К недостаткам алгоритма Грэхема следует отнести то, что он: 1) не является открытым; 2) не имеет обобщения на пространства размерности больше двух.

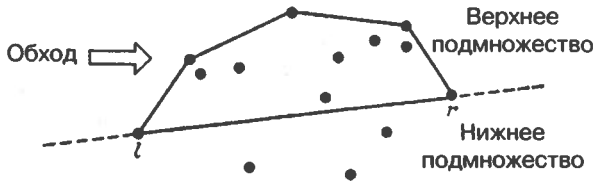

Рис. 2. Схема работы алгоритма Эндрю

4. Алгоритм обхода Джарвиса

Многоугольник с одинаковым успехом можно задать упорядоченным множеством как его ребер, так и его вершин. В предыдущих алгоритмах обращалось внимание главным образом на изолированные крайние точки. Однако если вместо этого попытаться определить ребра выпуклой оболочки, то такой подход может привести к созданию практически пригодного алгоритма. Если задано множество точек, то довольно трудно быстро определить, является ли некоторая точка крайней. Однако если даны две точки, то непосредственно можно проверить, является ли соединяющий их отрезок ребром выпуклой оболочки.

Джарвис использовал идею проверки ребер на вхождение в выпуклую оболочку и на её основе предложил алгоритм построения выпуклой оболочки [4]. Предположим, что найдена наименьшая в лексикографическом порядке точка p_1 заданного множества точек. Эта точка заведомо принадлежит оболочке, и теперь хотелось бы найти следующую за ней точку p_2 выпуклой оболочки. Точка p_2 — это точка, имеющая наименьший положительный полярный угол относительно точки p_1 как начала координат. Аналогично следующая точка p_3 имеет наименьший полярный угол относительно точки p_2 как начала координат, и каждая последующая точка выпуклой оболочки может быть найдена за линейное время. Алгоритм Джарвиса обходит кругом выпуклую оболочку (отсюда и соответствующее название — обход Джарвиса), порождая в нужном порядке последовательность крайних точек по одной точке на каждом шаге (рис. 3). Таким образом строится часть выпуклой оболочки (ломаная линия) от наименьшей (p_1 на рис. 3) до наибольшей точки в лексикографическом порядке (p_4 на том же рисунке). Построение выпуклой оболочки завершается нахождением другой ломаной, идущей из наибольшей точки в наименьшую в лексикографическом порядке. Ввиду симметричности этих двух этапов необходимо изменить на противоположные направления осей координат и иметь дело теперь с полярными углами, наименьшими относительно отрицательного направления оси x.

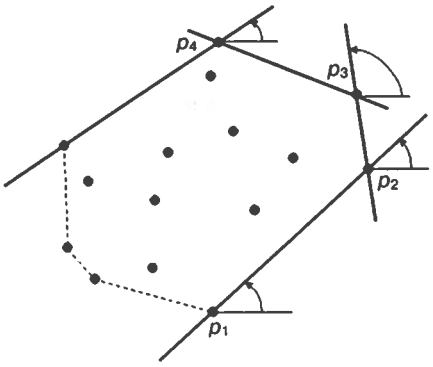

Рис. 3. Построение выпуклой оболочки методом Джарвиса. Алгоритм Джарвиса находит последовательные вершины оболочки путем многократного вычисления угла поворота. Каждая новая вершина определяется за время $O(n)$

Так как все n точек множества могут лежать на его выпуклой оболочке, а алгоритм Джарвиса затрачивает на нахождение каждой точки оболочки линейное время, то время выполнения алгоритма в худшем случае равно $O(n_2)$, что хуже, чем у алгоритма Грэхема. Если в действительности число вершин выпуклой оболочки равно h, то время выполнения алгоритма Джарвиса будет $O(hn)$, и он очень эффективен, когда заранее известно, что значение h мало.

5. Алгоритм «Разделяй и властвуй»

Алгоритм построения выпуклой оболочки, основанный на методе «Разделяй и властвуй», описан в [5].

В данном алгоритме множество S разбивается на два примерно равномощных подмножества S' и S'', выпуклые оболочки которых не пересекаются, затем рекурсивно строятся отдельно оболочки для каждого из них и объединяются. В случае, когда количество точек в подмножествах невелико, используется любой более простой способ построения выпуклой оболочки.

Сложность этого метода во многом зависит от эффективности нахождения слияния двух выпуклых оболочек. Пусть у нас есть выпуклые непересекающиеся многоугольники P' и P''. Нам требуется найти P – их слияние. Для этого находятся крайние точки каждого многоугольника (самая правая для левого, самая левая

для правого и т.д.). После этого путем последовательных проверок выпуклости строятся верхняя и нижняя касательные к многоугольникам (рис. 4). Это занимает не больше n проверок.

Как видно, и в этом случае на слияние требуется время $O(n)$, где n – это общее количество точек в многоугольниках.

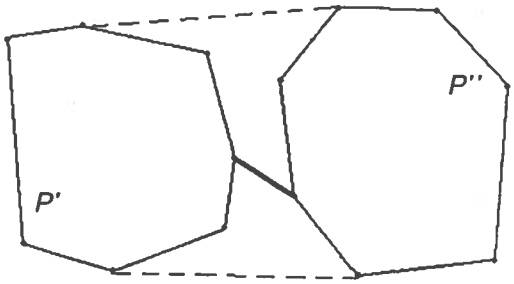

Рис. 4. Схема построения касательных в алгоритме «Разделяй и властвуй»

6. Алгоритм «быстрого построения»

Для построения выпуклой оболочки были созданы алгоритмы, напоминающие быструю сортировку. Такие алгоритмы называются быстрыми методами построения оболочки. Одними из первых такой алгоритм предложили Эдди [6] и Бикат [7].

Суть алгоритма состоит в том, что исходное множество S из n точек разбивается на два подмножества, каждое из которых будет содержать одну из двух ломаных, которые при соединении образуют выпуклую оболочку. Для начала нужно определить две точки, которые будут являться соседними вершинами выпуклой оболочки. Можно взять самую левую (a) и самую правую (b) вершины. После чего нужно найти точку c, максимально удаленную от прямой ab. Все точки, лежащие в треугольнике abc, исключаются из дальнейшего рассмотрения. Для того чтобы на каждом шаге исключать как можно больше точек, точка c выбирается по критерию наибольшей площади треугольника abc. Остальные точки будут делиться на два подмножества: точки, которые лежать левее ac, и точки, которые лежат правее, и cb. Каждое из них содержит ломаные, которые в сочетании с a, b и c дают выпуклую оболочку. С каждым из них проделываем то же самое. В подмножестве точек, лежащих левее cb, выбираем c', максимально удаленную от cb, которая делит его на три части. Из них одна выбрасывается, а остальные делятся опять (рис. 5). Это реализуется рекурсивной процедурой, которая для данного ей множества возвращает соответствующую часть выпуклой оболочки.

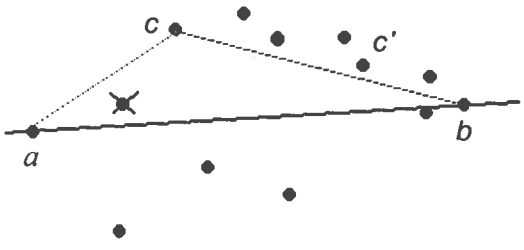

Рис. 5. Схема работы быстрого алгоритма построения выпуклой оболочки

В случае, когда мощность каждого из подмножеств, на которое делится множество, не превосходит некоторой константы, умноженной на мощность множества, получаем сложность алгоритма, как и в быстрой сортировке $O(n \log n)$. Но в худшем случае может потребоваться время $O(n^2)$.

7. Экспериментальное сравнение алгоритмов

Для количественного анализа описанных алгоритмов автором было проведено моделирование работы различных алгоритмов построения выпуклых оболочек на различных распределениях точек. При этом анализировалась скорость работы алгоритмов.

Для сравнения алгоритмов в различных условиях в экспериментах строились наборы данных, располагавшиеся в единичном квадрате $(0,1)^2$, со следующими характеристиками (рис. 6):

1. *Равномерное распределение* (рис. 6, *а*). Все точки были распределены равномерно и независимо в единичном интервале.

2. *Нормальное распределение* (рис. 6, *б*). Все точки размещались в единичном квадрате по нормальному распределению с центром в точке (0,5; 0,5) и среднеквадратическим отклонением 0,1. Точки, попадавшие за пределы единичного квадрата, отбрасывалась.

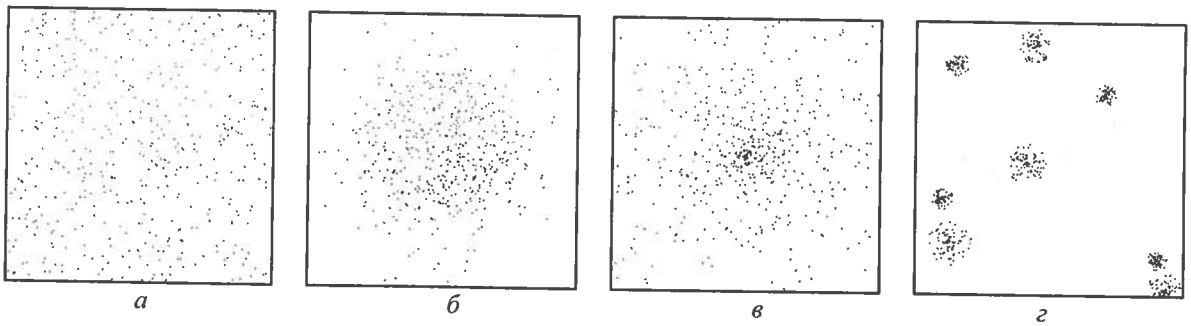

Рис. 6. Примеры тестовых данных, использованных при тестировании алгоритмов построения выпуклой оболочки

3. *Распределение Лапласа* (рис. 6, *в*). Все точки размещались в единичном квадрате по распределению Лапласа с центром в точке (0,5; 0,5). Точки, попадавшие за пределы единичного квадрата, отбрасывалась.

4. *Кластерное распределение* (рис. 6, *г*). Внутри единичного квадрата случайно выбиралось (равномерно и независимо по обеим координатам) случайное количество точек, которые становились центрами кластеров. Внутри кластеров точки распределялись по равномерному распределению в окружности случайного радиуса, описанной вокруг центра кластера.

Таблица 1

Сравнительное время работы алгоритмов при равномерном распределении исходного множества точек

Алгоритмы	Количество точек				
	50 000	100 000	150 000	200 000	250 000
Грэхема	165.53	387.66	590.73	754.92	929.69
Эндрю	121.97	290.64	456.40	551.38	695.45
Джарвиса	311.08	550.05	778.04	931.54	1001.31
«Разделяй и властвуй»	364.12	865.79	1495.28	2218.80	3214.23
Быстрый	43.71	99.90	153.31	196.96	235.97

Таблица 2

Сравнительное время работы алгоритмов при нормальном распределении исходного множества точек

Алгоритмы	Количество точек				
	50 000	100 000	150 000	200 000	250 000
Грэхема	150.02	345.13	506.10	718.97	915.65
Эндрю	117.16	250.15	401.74	571.94	747.06
Джарвиса	567.44	1257.93	1678.19	2157.91	2867.38
«Разделяй и властвуй»	643.88	1465.72	2279.89	3793.65	5077.91
Быстрый	43.82	92.27	134.3000	187.53	242.60

Таблица 3

Сравнительное время работы алгоритмов при лапласовском распределении исходного множества точек

Алгоритмы	Количество точек				
	50 000	100 000	150 000	200 000	250 000
Грэхема	142.21	317.33	495.42	683.28	888.42
Эндрю	98.42	227.95	352.57	486.57	640.71
Джарвиса	417.17	707.93	1013.28	1258.85	1395.14
«Разделяй и властвуй»	554.93	1395.50	2107.35	2753.12	3964.16
Быстрый	187.76	340.87	518.00	663.14	830.14

Таблица 4

Сравнительное время работы алгоритмов при кластерном распределении исходного множества точек

Алгоритмы	Количество точек				
	50 000	100 000	150 000	200 000	250 000
Грэхема	134.03	310.14	489.27	691.64	733.19
Эндрю	95.85	220.85	354.71	524.71	674.01
Джарвиса	732.14	1935.28	3196.57	4114.05	5562.57
«Разделяй и властвуй»	852.91	2215.51	3905.19	4548.14	6320.64
Быстрый	203.06	379.57	538.31	743.28	1163.14

В табл. 1–4 приведены результаты моделирования работы различных алгоритмов построения выпуклой оболочки на различных типах экспериментальных данных. Эксперименты проводились на машине с процессором Athlon 750, среднее время построения выпуклой оболочки представлено в миллисекундах. Во всех таблицах данные верны с относительной точностью 1% при доверительной вероятности 0,95.

Заключение

Как видно из приведённых таблиц, описанный в работе алгоритм быстрого построения выпуклой оболочки наиболее хорошо работает на равномерных и нормальных распределениях, когда объекты мало пересекаются между собой и их разброс велик. Но на неравномерных распределениях заметно ухудшение рабо-

ты этого алгоритма по сравнению с алгоритмом Эндрю. Тем не менее, оба эти алгоритма достаточно хороши по сравнению с другими описанными алгоритмами. Практически во всех тестах они намного обходят остальные алгоритмы по скорости работы.

Из представленных результатов экспериментального моделирования видно, что правильный выбор алгоритма для конкретных задач позволяет повысить скорость построения выпуклой оболочки.

Таким образом, представляется разумным использование комбинированного алгоритма, выбирающего тот или иной базовый алгоритм в зависимости от типа распределения и количества точек.

ЛИТЕРАТУРА

1. *Препарата Ф., Шеймос М.* Вычислительная геометрия: Введение: Пер. с англ. – М.: Мир, 1989. – 478 с.
2. *Graham R.L.* An efficient algorithm for determining the convex hull of a finite planar set // Information Processing Letters. – 1972. – V. 1. – P. 132–133.
3. *Andrew A.M.* Another efficient algorithm for convex hulls in two dimensions // Information Processing Letters. – 1979. – V. 9. – P. 216–219.
4. *Jarvis A.* On the identification of the convex hull of a finite set of points in the plane // Information Processing Letters. – 1973. – V. 2. – P. 18–21.
5. *Preparata F.P., Hong S.J.* Convex hulls of finite point sets in two and three dimensions // Communicat. of the ACM. – 1977. – V. 2 (20). – P. 87–93.
6. *Eddy W.* A new convex hull algorithm for planar sets // ACM Transactions on Mathematical Software. – 1977. – V. 3 (4). – P. 398–403.
7. *Bykat A.* Convex Hull of a Finite Set of Points in Two Dimensions // Information Processing Letters. – 1978. – V. 7. – P. 296–298.

УДК 519.688

ПАКЕТ ПРИКЛАДНЫХ ПРОГРАММ ДЛЯ СОРТИРОВКИ НА ПЭВМ

С.Г. Шестаков, Н.А. Лукин

Институт машиноведения УрО РАН, г. Екатеринбург

E-mail: fop@imach.uran.ru

Рассмотрен опыт создания пакета программ для измерения производительности компьютеров типа IBM PC под управлением ОС Windows.

Ключевые слова: *исследование архитектур, производительность, бенчмарк, сортировка.*

Введение

Непрерывное развитие архитектур процессоров общего назначения, связанное, в первую очередь, с необходимостью увеличения производительности, приводит к их изменению в сторону специализации. В частности, реализация в Pentium4 технологии расслоения памяти (HyperTrading), широко применяемая в бортовых ЦВМ, начиная еще с 70-х годов, а также кэширование команд и данных фактически дают возможность осуществлять глубокую конвейеризацию. Это может существенно изменить производительность ЦП при решении прикладных задач и затруднить оценку производительности. Поэтому все большее значение как для проведения оценок, так и экспериментального исследования архитектур, приобретают тестовые задачи (benchmarks), время решения которых на выбранной архитектуре не только говорит о ее производительности, но и дает возможность оценить ее эффективность и сравнить ее с другими архитектурами.

Для оценки производительности и проведения экспериментов с архитектурами нами была выбрана задача сортировки. Основные причины состояли в следующем:

- сортировка является актуальной задачей для реализации на перспективных параллельных архитектурах (например, на систолических процессорах), при этом положительного эффекта от распараллеливания удается достичь, как правило, на массивах данных порядка гига- и терабайт. Поэтому необходимо иметь удобную, гибкую к изменениям входных параметров программу, которая будет обеспечивать сортировку максимально больших массивов и являться источником эталонных результатов;
- в настоящее время, несмотря на обилие различных тестовых задач, как ни странно, предназначенных для измерения и оценки производительности, стандартные программы сортировки массивов размерностью порядка даже десятков мегабайт отсутствуют, что не дает возможности как образо-