Построение обобщенной стратегии решения некоторых задач вычислительной геометрии

В.Н. Терещенко, В.В. Сапсай, И.С. Статкевич, А. Федоров

Статья посвящена разработке эффективного обобщенного алгоритма для решения совокупности взаимосвязанных задач вычислительной геометрии при построении точных графических моделей явлений и процессов. В частности, в задачах графического и геометрического моделирования сложных теплофизических процессов, во многих случаях, требуется точность построения модели. Для построения такой модели решается одновременно целый комплекс теплофизических и геометрических задач, что требует значительных вычислительных ресурсов. В данной статье для совокупности задач вычислительной геометрии, которые имеют на однопроцессорной модели вычислений нижнюю оценку сложности $\Omega(NlogN)$ предложен обобщенный параллельный алгоритм их решения на основе стратегии «разделяй и властвуй». Благодаря построению структурированного массива точек заданного множества и удачно выбранных структур данных: дерева алгоритма и сцепляемой очереди, удалось разработать обобщенный эффективный рекурсивно-параллельный алгоритм решения поставленной задачи.

Характерной чертой практической реализации данного подхода является то, что разработанный алгоритм позволяет одновременно решать как разные шаги одной процедуры задачи на многих процессорах, так и разные процедуры в одном узле алгоритма посредством технологий ПАРУС и MPI.

1. Введение

Современное развитие компьютерных технологий позволяет ставить и решать новые сложные задачи, которые нуждаются в построении комплексных математических моделей и способов их решения. Необходимо решать на одном и том же множестве данных одновременно не одну задачу, а целую совокупность взаимосвязанных задач. Так, при разработке и построении графической модели теплофизических процессов для сварки лопаток турбин двигателей самолетов, чрезвычайно точной должна быть модель сварочной ванны, шва и области предельной с ними. Для анализа таких процессов важным является определение параметров теплофизических процессов на границе расплава и материала. Чтобы получить точное решение этой задачи необходимо решать одновременно целый комплекс теплофизических и геометрических задач: динамические задачи термоупругости, построение графической модели сварочной ванны и близ лежащей области. Это требует значительных вычислительных ресурсов. Поэтому естественным стоит вопрос разработки общих подходов, которые бы позволяли эффективно решать одновременно ряд геометрических задач на одном и том же множестве входных данных. Один из таких подходов - это создание параллельных алгоритмов.

На сегодняшний день разработано много эффективных параллельных алгоритмов решения отдельных задач вычислительной геометрии, которые, в частности, детально описаны в достаточно емком труде [1], где также имеется много ссылок на разработки других авторов. Но при решении целого комплекса задач на одном и том же множестве данных использование отдельных эффективных алгоритмов в общем случае не приносит желаемой эффективности. Дело в том, что каждый алгоритм нуждается в собственной предварительной обработке и создании структуры данных, а также процедур реализации, что не дает минимального времени решения такого класса задач. Поэтому необходимо было выбрать такую стратегию, которая бы позволял получить самую эффективную реализацию. При построении такой стратегии учитывалось то, что большинство задач вычислительной геометрии имеет внутренний параллелизм и предусматривает рекурсивную

природу реализации. Наиболее подходящей стратегией, выходя из отмеченных особенностей, на наш взгляд, может быть известная стратегия «разделяй и властвуй».

В данной работе для совокупности задач вычислительной геометрии, которые имеют нижнюю оценку сложности $\Omega(NlogN)$, предложен обобщенный параллельно-рекурсивный алгоритм их решения, который принадлежит классу разрешимости NC_2 .

Постановка задачи. *Пусть заданное множество S из N точек в пространстве* E^d. *Разработать обобщенный эффективный рекурсивно-параллельный алгоритм решения задач вычислительной геометрии, определенных на одном и том же множестве S, нижняя оценка сложности, которых порядка* $\Omega(NlogN)$ *(для однопроцессорной машины).*

1. Алгоритм на основе стратегии «распределяй и властвуй»

Основная идея предложенного алгоритма базируется на стратегии “разделяй и властвуй“, которая состоит из двух этапов: рекурсивного разбиения заданного множества точек на равной мощности подмножества и этапа слияния, в процессе которого на каждом шаге рекурсии решается задача соответствующего множества точек путем слияния результатов для ее подмножеств.

Основной проблемой применения схемы ”разделяй и властвуй” при решении задач вычислительной геометрии есть нелинейность этапа слияния и отсутствие линейной разделимости множества точек. В рассматриваемом подходе для задач, входными данными которых является множество точек, благодаря удачному представлению входных данных на этапе предварительной обработки и использования параллельной обработки на этапах разбиения и слияния удалось построить эффективный рекурсивно-параллельный алгоритм, который снимает указанные ограничения к применению.

1.1. Математическая модель алгоритма

Для построения параллельного алгоритма решения поставленной задачи за основу можно взять идею из последовательного алгоритма [2]. Тогда математическая модель предлагаемого параллельного алгоритма будет состоять из таких основных этапов:

Этап 1. Предварительная обработка.
Этап 2. Разбиения множества точек (рекурсивный спуск).
Этап 3. Рекурсивное слияние результатов для подмножеств (рекурсивный подъем).

Этап 1. Предварительная обработка. Пусть заданное множество S из N точек $S = \{ P_1, P_2, \ldots, P_N \}$ на плоскости и $O(N)$ процессоров. Сначала построим отсортированный список точек по x координате $U_x = \{P_{1x}, P_{2x}, \ldots, P_{Nx} \}$, а затем отсортированный список точек по y координате $U_y = \{P_{1y}, P_{2y}, \ldots, P_{Ny} \}$. На рис.1 представлен случай для N=12. Построение отсортированных списков с помощью $O(N)$ процессов можно осуществить с помощью одного из алгоритмов, детально описанных в работах [3-5], которые дают оценки сложности порядка $O(log^2N)$ и $O(logN)$. На основе списков U_x, U_y формируем массив точек $U = \{P_{ij}, \ i,j = 1, N\}$, где i, j - индексы которые указывают номер точки в упорядоченных списках U_x, U_y, соответственно. Сформированный таким способом массив подается на вход алгоритма, граф которого представлен на рис.2. В этом дереве каждый узел обозначен целым числом k, относительно которого разбивается список точек в узлах на два равной мощности, относительно медианы, списка, после сравнения первых индексов точек массива P_{ij}. А каждый номер узла k определяется за один проход по дереву, если известное количество точек заданного множества, простой формулой:

$$k = [\ (m+M)/2] \qquad (1)$$

, где m –номер первого элементу списка, M –последний элемент списка.

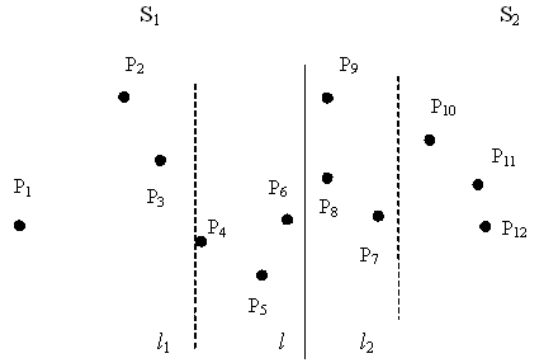


Рис.1. Упорядоченный список U={$P_{1,3}$, $P_{2,12}$, $P_{3,9}$, $P_{4,2}$, $P_{5,1}$, $P_{6,5}$, $P_{7,8}$, $P_{8,11}$, $P_{9,6}$, $P_{10,10}$, $P_{11,7}$, $P_{12,4}$}

Этап 2. Разбиение множества точек (рекурсивный спуск). Этот этап алгоритма заключается в разбиении на каждом шаге рекурсии заданного множества точек в виде списка U на равной мощности подмножества U_1, U_2, поиска медианы l и передачи U_1, U_2 на следующий шаг рекурсии. Процесс разбиения заканчивается, когда в подмножествах остается по одному элементу. Структура данных, которая бы поддерживала этот процесс, имеет вид рис.2. Поиск медианы на упорядоченном по координате x индексированном массиве точек U выполняется за константное время $O(1)$: $l = (P_{kj}+P_{k+1j})/2$, где k определяется из формулы (1), а M количество точек в списке.

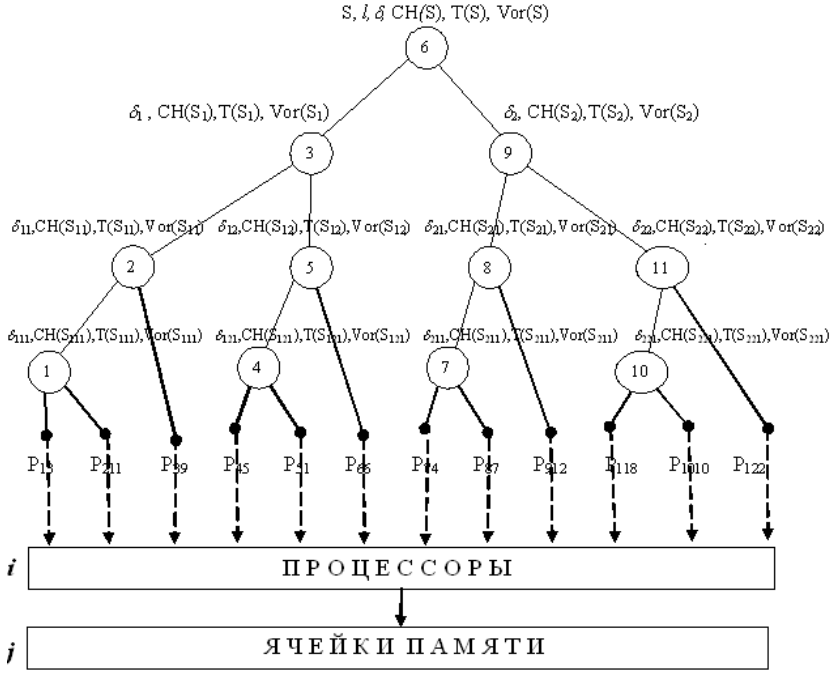


Рис.2 Граф алгоритма. Вызов процедур вычисления: *CH(S), Vor(S), T(S)* и ближайшей пары (δ_{min}).

Время, необходимое на рекурсивный спуск параллельного алгоритма, определяется следующей леммой.

Лемма 1. *Этап рекурсивного разбиения множества S из N точек на равномощные подмножества S_1 и S_2 плоскости, поиск медианы l и передачу подмножеств S_1 и S_2, с помощью O(N) процессоров, можно выполнить за время O(logN).*

Доказательство. Пусть задано множество точек в виде индексированного двумерного, упорядоченного массива $U = \{P_{ij},\ i,j = 1, N\}$. Для построения такой структуры данных можно воспользоваться, например, параллельным алгоритмом сортировки со временем *O(logN)*, предложенным Коле [5]. Такое представление множества точек позволяет, при известном количестве точек *N* в списке *U*, построить дерево разбиения, рис.2. На каждом шаге разбиения соответствующие процессоры синхронно сравнивают первые индексы из списка точек и рассылают точки в соответствующие узлы алгоритма, сохраняя при этом порядок расположения точек, который определяется их порядком в ячейках памяти. Учитывая четкую упорядоченность по обоим индексам точек P_{ij} массива *U* и взаимосвязь между процессорами и элементами памяти, время выполнения процесса слияния в каждом узле дерева не будет превышать константу, *O(*1*)*.Таким образом, общее время разбиения не превышает *O(logN)* для наихудшего ввода данных. Что и нужно было доказать.

Процессы, которые реализуют алгоритм на этапе разбиения можно описать таким образом. Пусть *n* - количество элементов списка в узлах алгоритма, которое определяется делением количества элементов родительского узла пополам; *r*– уровень дерева алгоритма, который отвечает шагу рекурсии; *k*- номер узла, который определяется соотношением (1); *i*,*j* – индексы элементов массива $U=\{P_{ij},i,j = 1, N\}$.

Пока $n > 1$, всем процессам:

1. Присвоить номер *k* каждому узлу дерева алгоритма согласно (1).
2. На *r*-ом шаге алгоритма в узле *k* определить медиану *l* множества *S(U)* по формуле $l = (P_{kj}+P_{k+1j})/2$, где *k* - номер узла;
3. На *r*-ом шаге алгоритма в узле *k* необходимо сравнить значение *i*-го индекса P_{ij} элементов массива *U* с числом *k*.

   а) если $i \le k$ – послать каждый элемент $P_{ij} \in U$, индекс которого удовлетворяет этому условию, в узел " ЛЕВЫЙ СЫН" по *i*-ому каналу и записать в *j*- тую ячейку памяти узла.

   б) если $i > k$ – послать каждый элемент $P_{ij} \in U$, индекс которого удовлетворяет этому условию, в узел " ПРАВЫЙ СЫН" по *i*-ому каналу и записать в j- тую ячейку памяти узла.
4. n = 1. Завершение работы этого этапа алгоритма.

Таким образом, в результате работы алгоритма на этом этапе будут сформированы упорядоченные по *y*, равной мощности подмножества точек U_1, U_2 относительно медианы *l* (при этом U_1 расположена слева, а U_2 – справа, относительно *l*).

Этап 3. Рекурсивное слияние результатов для подмножеств (рекурсивный подъем). На этом этапе, начиная с листовых узлов дерева алгоритма, полученные результаты решения соответствующей задачи (выпуклой оболочки, диаграммы Вороного, декомпозиции и т.д.) подаются в родительские узлы, где используются соответствующие процедуры слияния для построения общего решения задачи. Процесс заканчивается результатом слияния в корневом узле. В следующем разделе будут описаны процедуры слияния для задач: построение выпуклой оболочки, триангуляции и диаграммы Вороного. Причем центральное место занимает процедура слияния диаграммы Вороного так, как сама диаграмма и техника ее построения используются для решения других важных задач вычислительной геометрии. В работе [6] предложена процедура слияния этого алгоритма для задачи «О ближайшей паре». Особенностью предложенных процедур является применение сцепляемой очереди для представления точек в узлах алгоритма, что позволяет выполнять все операции за логарифмическое время. Вопрос использования оптимального состава процессоров (компьютеров) в данной работе не рассматривался и нуждается в дополнительном исследовании. В то же время известно из работы [7], что если взять за *w*- общее количество операций алгоритма, а за *O(F(N)*) – сложность алгоритма для неограниченного количества процессоров, то время выполнения алгоритма для *p* процессоров можно выразить следующим соотношением:

$$O(\mathrm{f}(N))= ((p-1)O(\mathrm{F}(N))+ w)/p \qquad (2)$$

2. Построение процедур слияния

2.1. Процедура слияния задачи «Построение выпуклой оболочки»

Для построения процедуры слияния рассматриваемого параллельного алгоритма использовались некоторые элементы из известного последовательного алгоритма [8]. В частности, в алгоритме была предложена процедура слияния верхней (нижней) выпуклой оболочки множества точек при решении динамической задачи, с оценкой *O(logN)*. Такая эффективность достигается за счет использования структуры данных в виде сцепляемой очереди, что позволяет выполнять все необходимые операции на каждом шаге за логарифмическое время.

Постановка задачи. *На плоскости заданно множество S из N точек. Разработать процедуру слияния выпуклых оболочек порядка O(logN) для рекурсивно– параллельного алгоритма построения выпуклой оболочки заданного множества точек.*

Пусть заданно множество *S* из *N* точек на плоскости, рис.1, и выполнены этапы 1,2 описываемого алгоритма. Для решения поставленной задачи рассмотрим лишь 3-ий этап.

Слияние результатов (рекурсивный подъем).

На каждом шаге рекурсивного подъема, начиная со второго, на вход родительского узла *v* дерева алгоритма (рис.2) подаются выпуклые оболочки от левого U_L (U_L = ЛСИН[*v*]) и правого U_R (U_R = ПСИН[*v*]) сыновей, соответственно. Необходимо построить выпуклую оболочку для узла *v*. Время, необходимое на построение выпуклой оболочки, определяется следующей леммой.

Лемма 2. *Этап рекурсивного слияния результатов определения выпуклой оболочки множества S из N точек на плоскости с помощью O(N) процессоров, можно выполнить за время O(logN).*

Доказательство. **procedura** *(СЛИЯНИЕ_ ВЫПУКЛАЯ ОБОЛОЧКА)*(*CH*(S_L), CH(S_R))

Для построения выпуклой оболочки выпуклых оболочек сыновей узла *v*, необходимо выполнить следующие операции: определить верхние (нижние) опорные вершины для левой (правой) выпуклых оболочек; расцепить взаимно выпуклые цепи между этими опорными точками; соединить оставшиеся цепи соответствующими опорными отрезками.

При выборе структуры данных, которая бы выполняла выше указанные операции за *O(logN)* и поддерживала выпуклую оболочку, воспользуемся идеей, предложенной в работе Овермарса и Ван Лювена [8] для последовательного алгоритма, согласно которой такой структурой данных может быть сцепляемая очередь, с заданной на ней процедурой СОЕДИНИТЬ(U_L, U_R). В работе доказано, что процедура СОЕДИНИТЬ(U_L, U_R), начиная с корневых вершин сбалансированных деревьев, которые поддерживают U_L и U_R, позволяет построить верхнюю и нижнюю границы выпуклой оболочки за время *O(logN)*. При выполнении поиска опорных вершин и построении опорных отрезков, в нашем случае, используется такая же классификация вершин, как и в работе [8]. Необходимо лишь к девяти представленным в работе случаев классификации для построения верхней выпуклой оболочки прибавить девять аналогичных случаев для построения нижней выпуклой оболочки. После определения опорных точек и сцепления с помощью их выпуклых оболочек сыновей узла *v*, отбрасываются правая и левая части деревьев, которые поддерживают U_L и U_R между опорными точками, соответственно. Сбалансированные деревья, которые будут поддерживать верхнюю и нижнюю выпуклую оболочку узла *v* образуются путем слияния частей деревьев, которые остались. Опять же, все приведенные операции выполняются за время *O(logN)*. Полученные таким способом выпуклые оболочки передаются на следующий уровень рекурсии дерева алгоритма, рис.2.

На рисунке 3 показан шаг слияния алгоритма. В каждом узле дерева алгоритма вызывается процедура СЛИЯНИЯ_ВЫПУКЛАЯ ОБОЛОЧКА (*CH*(U_L),*CH*(U_R)), которая находит опорные вершины (отрезки), строит и поддерживает новую выпуклую оболочку.

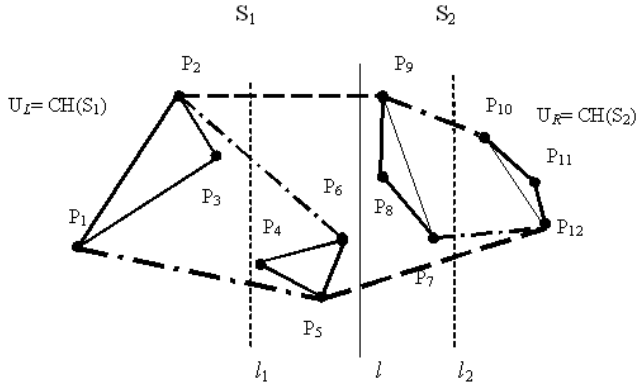
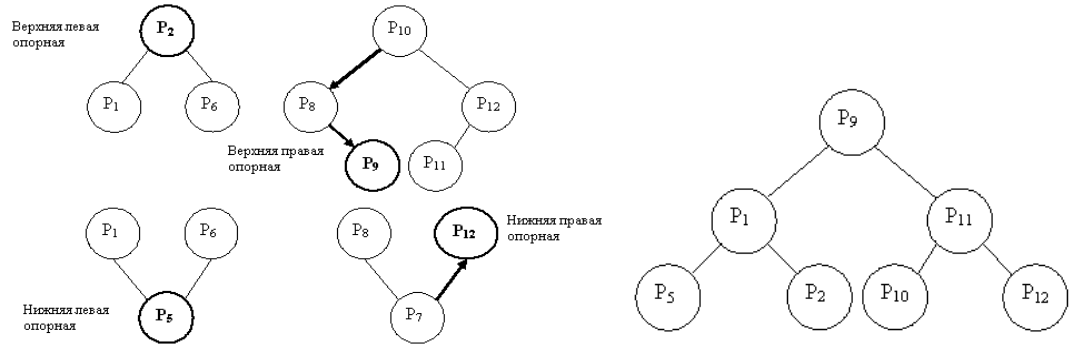


Рис..3. Пример этапа слияния выпуклых оболочек для множества точек рисунка 1.

На рис.4-а показан поиск опорных вершин левой и правой выпуклых оболочек примера (рис. 3) на шаге слияния. По вершинам P_2, P_5 расцепляется левая выпуклая оболочка $U_L = CH(S_1)$ на цепи $U_{L1} = (P_2, P_1, P_5)$ и $U_{L2} = (P_5, P_6, P_2)$, а правая выпуклая оболочка $U_R = CH(S_2)$ расцепляется по вершинам P_9, P_{12} на цепи $U_{R1} = (P_9, P_8, P_7, P_{12})$ и $U_{R2} = (P_{12}, P_{11}, P_{10}, P_9)$. Поскольку цепи $U_{L2} = (P_5, P_6, P_2)$ и $U_{R1} = (P_9, P_8, P_7, P_{12})$ взаимно выпуклые то, согласно алгоритма, они отбрасываются, а значит, и исчезают узлы деревьев которые поддерживают соответствующие выпуклые оболочки. Так в дереве, которое поддерживает левую выпуклую оболочку исчезнут узлы, которые находятся справа относительно опорных точек P_2, P_5 (это узел P_6) , а в дереве правой выпуклой оболочки исчезнут узлы, точки которых расположены слева от правых опорных точек (это узлы P_8, P_7). После этого сцепляются цепи $U_{L1} = (P_2, P_1, P_5)$ и $U_{R2} = (P_{12}, P_{11}, P_{10}, P_9)$ опорными отрезками (P_2, P_9) и (P_5, P_{12}), которые и образуют результирующую границу выпуклой оболочки $CH(S) = (P_{12}, P_{11}, P_{10}, P_9, P_2, P_1, P_5)$. Части деревьев, которые остались, сольются в единое сбалансированное дерево, которое поддерживает полученную границу конечной выпуклой оболочки $CH(S)$, рис.4-б.



а) *б)*

Рис.4. Структуры данных- сцепляемые очереди выпуклых оболочек: а- до слияния, б- в результате слияния.

2.2. Процедура слияния задачи «Построение диаграммы Вороного»

Схема “распределяй и властвуй” имеет успешное применение и при построении Диаграммы Вороного для множества S из N точек на плоскости. В частности, М. Шеймосом [2] предложен эффективный последовательный алгоритм решения этой задачи на одно процессорной машине со временем $\theta(NlogN)$, а в работе [1] предложен эффективный параллельный алгоритм со временем $\theta(log^2N)$. Такая эффективность была достигнута за счет анализа границ диаграмм Вороного в зоне разделяющей прямой. В этом разделе этап слияния алгоритма построения диаграммы Вороного будет отличаться от этапа слияния задачи “выпуклая оболочка” лишь тем, что в первой задаче на

завершающем шаге строится разделяющая цепь, которая соединяет диаграммы Вороного сыновей, а во второй - проводятся опорные отрезки к выпуклым оболочкам сыновей. При этом результаты построения выпуклых оболочек и опорных отрезков, полученные в задаче “выпуклая оболочка”, используются для следующих шагов задачи построения диаграммы Вороного.

Постановка задачи. *На плоскости задано множество S из N точек. Разработать эффективную процедуру слияния параллельно-рекурсивного алгоритма построения Диаграммы Вороного для этого множества точек.*

Слияние результатов (рекурсивный подъем).

На каждом шаге рекурсивного подъема, начиная со второго, на вход родительского узла *v* дерева подаются диаграммы Вороного (ДВ) для подмножества точек от левого сына ***vor***(S_L) и правого сына ***vor***(S_R). Необходимо построить ДВ для узла *v*. Время, которое расходуется на этап рекурсивного слияния решения задачи, в случае параллельного алгоритма, определяется следующей леммой.

Лемма 3. *Этап рекурсивного слияния диаграмм Вороного множества S из N точек на плоскости с помощью O(N) процессоров, можно выполнить за время* $O(\log^2 N)$.

Доказательство. **procedura** (*СЛИЯНИЕ _ ДИАГРАММА ВОРОНОГО*) ($VOR(S_L)$, $VOR(S_R)$)

Для построения слияния диаграмм Вороного сыновей узла *v*, необходимо:

1. Определить верхние и нижние опорные вершины левой и правой границ выпуклых оболочек $CH(S_L)$ и $CH(S_R)$ сыновей, а, следовательно, верхнее и нижнее опорные ребра, соответственно.
2. Провести входящее и выходящее ребра разделяющей цепи для верхнего и нижнего опорных отрезков до пересечения с одним из ребер диаграмм ***vor***(S_L) или ***vor***(S_R),рис.5.

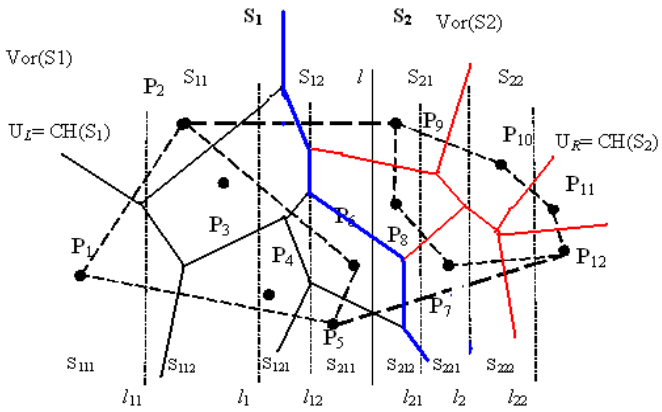


Рис. 5. Слияние диаграмм Вороного.

3. Проводится процесс построения разделяющей цепи с использованием *O(N)* процессоров для приграничных множеств точек относительно разделяющей вертикали *l*, которые расположены слева и справа от нее, и которые принадлежат взаимно выпуклым цепям выпуклых оболочек сыновей, а также точки определяемые ребрами диаграмм ***vor***(S_L), ***vor***(S_R), пересекающие ребра этих цепей.
4. Полученные после слияния таким способом диаграммы Вороного передаются на следующий уровень рекурсии, рис.2.
5. Процесс слияния завершается, когда будет построена разделяющая цепь для корня дерева, и тем самым диаграмма Вороного множества точек *S*.

Рассмотрим более детально шаг 3 указанной процедуры.

Лемма 4. *Построение разделяющей цепи* $\sigma(S_1, S_2)$, *которая «сшивает» диаграммы Вороного* $vor(S_L)$, $vor(S_R)$, *на каждом шаге этапа слияния можно выполнить за время* $O(\log N)$ *с использованием* $O(N)$ *процессоров.*

Доказательство. Между верхними и нижними опорными ребрами двух выпуклых оболочек диаграмм Вороного сыновей ***vor***(S_L), ***vor***(S_R) некоторого узла v графа алгоритма имеем две взаимовыпуклые цепи, которые, собственно, и определяют область построения D разделяющей цепи (рис.6). Каждая из этих цепей определяет упорядоченный набор ребер диаграммы Вороного, направленных в середину области D, которые пересекают ребра взаимно выпуклых цепей CH_L, CH_R. Каждое из ребер цепей CH_L, CH_R может пересекаться одним или несколькими ребрами диаграмм Вороного и, тем самым, определяет множество точек разделенных этими ребрами. Обозначим множество вершин, которое состоит из вершин выпуклой цепи CH_L (CH_R) и точек определенных ребрами ДВ, которые пересекают цепь, через $B_L(S_1)$ $(B_R(S_2))$ и назовем его *лево-предельным (право-предельным)* упорядоченным множеством точек (списком). Соединив последовательно точки в списках $B_L(S_1)$ и $B_R(S_2)$ получим списки ребер $E_L(S_1)$ і $E_R(S_2)$, которые образуют цепи C_L и C_R соответственно, рис. 6. Здесь: $B_L(S_1) = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8\}$, $(B_R(S_2)) = \{P_9, P_{10}, P_{11}, P_{12}, P_{13}, P_{14}, P_{15}\}$ - списки приграничных точек. $C_L = \{ e_1, e_2, e_3, e_4, e_5, e_6, e_7 \}$, $C_R = \{ e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13} \}$- левая и правая монотонные цепи между опорными отрезками, для которых строится разделяющая цепь. $CH_L = \{ e_1, (P_2, P_4), e_4, (P_5, P_7), e_7\}$, $CH_R = \{e_8, (P_{10}, P_{15})\}$- взаимно выпуклые цепи.

Лемма 5. *Цепи* C_L *и* C_R *будут монотонными относительно прямой* ***l***.

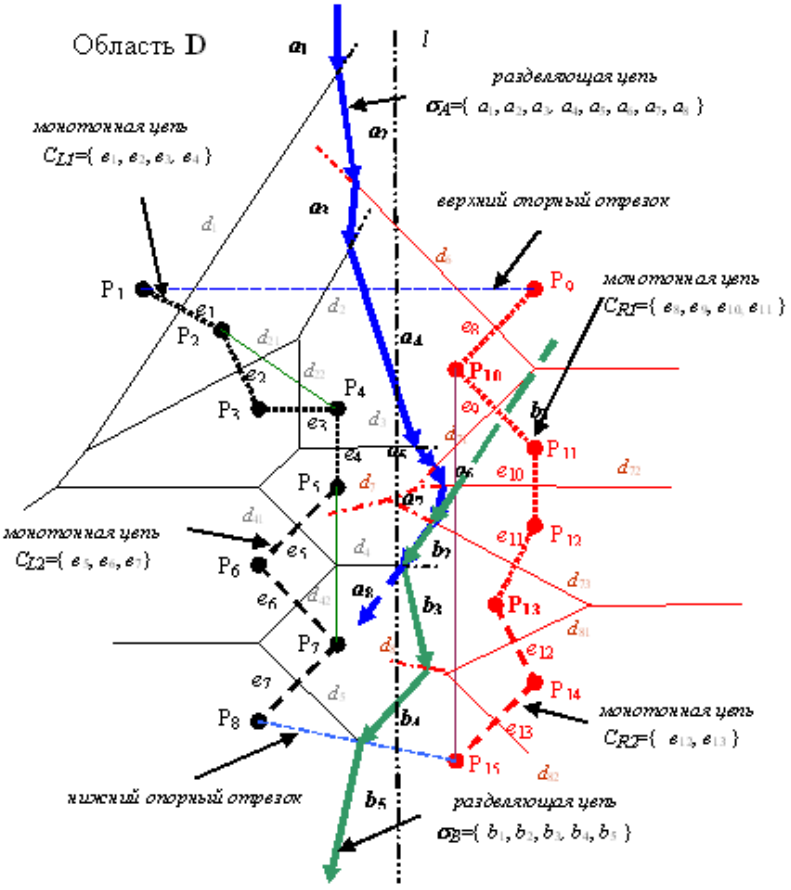
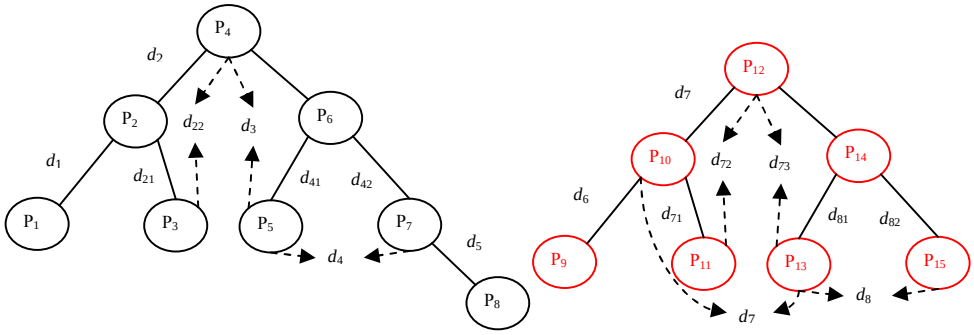


Рис.6. Шаг слияния двух разделяющих цепей в области D (верхней $\sigma_A = \{ a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8 \}$ и нижней $\sigma_B = \{ b_1, b_2, b_3, b_4, b_5 \}$ цепей) для пар монотонных цепей (C_{L1}, C_{R1}), (C_{L2}, C_{R2}), соответственно.

Доказательство. Докажем от противного. Известно, что разделяющая цепь $\sigma(S_1, S_2)$ обязательно должна быть монотонной относительно прямой l. Пусть хотя бы одна из цепей C_L и C_R будет не монотонной относительно l. Тогда найдется ребро этой цепи, которое имеет угол поворота относительно оси OX с началом в конце текущего ребра $\alpha > \pi$, а, следовательно, соответствующее ребро диаграммы Вороного не попадет в область D, и разделяющая цепь σ не будет монотонной, что противоречит условию.

Стоит отметить, что процедуру слияния в каждом узле дерева алгоритма можно выполнять независимо и параллельно на нескольких процессорах. Для выполнения, выше приведенных, операций за логарифмическое время, используется та же структура данных, что и в задаче "О выпуклой оболочке" - сцепляемая очередь, с заданной на ней процедурой СОЕДИНИТЬ(U_L, U_R). Она позволяет эффективно строить разделяющую цепь. Для организации процесса построения разделяющей цепи, на основе списков $B_L(S_1)$ и $B_R(S_2)$, создадим соответствующие структуры данных (рис.7), загрузив их необходимыми данным. Сцепляемые очереди обеих монотонных цепей представляют собой бинарное дерево с корнем, в узлах которого координаты вершин, а дуги - соответствующие ребра e_k (k=1, N) из списков цепей $E_L(S_1)$ и $E_R(S_2)$. Кроме этого, узлы загружены указателями на соответствующие ребра диаграммы Вороного (обозначим их через d_{ij}, $i,j \in N$).



а) *б)*

Рис.7. Структуры данных: сцепляемые очереди для левой C_L и правой C_R цепей области слияния D диаграмм Вороного $Vor(S_1)$, $Vor(S_2)$ рисунка 6, соответственно. d_{ij} – имена ребер $Vor(S_1)$, $Vor(S_2)$, которые лежат в области D.

Такие структуры данных, позволяют построить разделяющую цепь $\sigma(S_1, S_2)$ с помощью $O(N)$ процессоров за время $O(\log N)$. Схему алгоритма построения разделяющей цепи можно тоже представить в виде бинарного дерева. В листьях дерева каждым с $O(N)$ процессоров строится разделяющая цепь для соответствующих пар ребер (e_k, e_l) ($e_k \in E_L(S_1)$, $e_l \in E_R(S_2)$). Полученные результаты подаются на следующий уровень дерева, где осуществляется шаг слияния, в результате чего строится разделяющая цепь – объединение разделяющих цепей сыновей. Процесс слияния разделяющих цепей сыновей требует $O(1)$ времени в каждом узле дерева. Как видим, все операции при построении разделяющей цепи требуют не более чем $O(\log N)$ времени при использовании $O(N)$ процессоров. Сбалансированные деревья, которые поддерживают верхнюю и нижнюю выпуклые оболочки узла v образуются путем слияния частей деревьев, которые остались. Полученные таким образом деревья поддерживают выпуклую оболочку и позволяют выполнять процедуры построения разделяющей цепи за время $O(\log N)$.

2.3. Процедура слияния задачи «Триангуляция»

Задача триангуляции на ограниченном множестве точек достаточно изучена и для ее решения существует ряд эффективных последовательных алгоритмов. Здесь следует отметить работы Скворцова и, в частности роботу [9], в которой предложено несколько быстрых алгоритмов триангуляции. В данной работе предлагается процедура триангуляции для рассматриваемого

параллельного алгоритма в случае асимптотически больших множеств точек (или множеств точек представляемых структурированные объекты).

Постановка задачи. *На плоскости задано множество S из N точек. Разработать эффективную процедуру слияния параллельно-рекурсивного алгоритма построения триангуляции для этого множества точек.*

Поскольку процедура разбиения общая для всех задач, которые решаются предложенным алгоритмом, то рассмотрим лишь процедуру слияния в задаче триангуляции.

Слияние результатов (рекурсивный подъем).

На каждом шаге рекурсивного подъема, начиная со второго, на вход родительского узла *v* дерева, рис.2, подаются триангуляции от левого $T(S_L)$ и правого $T(S_R)$ сыновей, соответственно. Триангуляции $T(S_L)$ и $T(S_R)$, ограниченны границами выпуклых оболочек $CH(S_L)$ и $CH(S_R)$. Необходимо построить триангуляцию для узла v, как слияние триангуляций сыновей.

Лемма 6. *Этап рекурсивного слияния в задаче триангуляции множества S из N точек на плоскости с помощью O(N) процессоров, можно выполнить за время $\theta(\log N)$.*

Доказательство. **procedura** *(СЛИЯНИЕ_ ТРИАНГУЛЯЦИЯ)* ($T(S_L)$, $T(S_R)$)

- Для построения триангуляции на основе триангуляций сыновей узла v, необходимо:

1. Определить верхние и нижние опорные вершины $CH(S_L)$ и $CH(S_R)$ соответствующих триангуляций.
2. Соединить найденные опорные вершины верхним и нижним опорными отрезками.
3. Двигаясь по одной из взаимно выпуклых цепей между верхним и нижним опорными отрезками, каждый процессор проводит отрезок к своей вершине другой цепи. Процесс продолжается до тех пор, пока не будет достигнут нижний опорный отрезок.
4. Полученные таким способом триангуляции передаются на следующий уровень рекурсии.

Для того чтобы выполнить выше отмеченные операции наиболее подходящей, как для и предыдущей процедуры, тоже будет сцепляемая очередь, которая реализует этап слияния триангуляций за логарифмическое время. На рис. 8 продемонстрирован пример пошагового этапа слияния триангуляций.

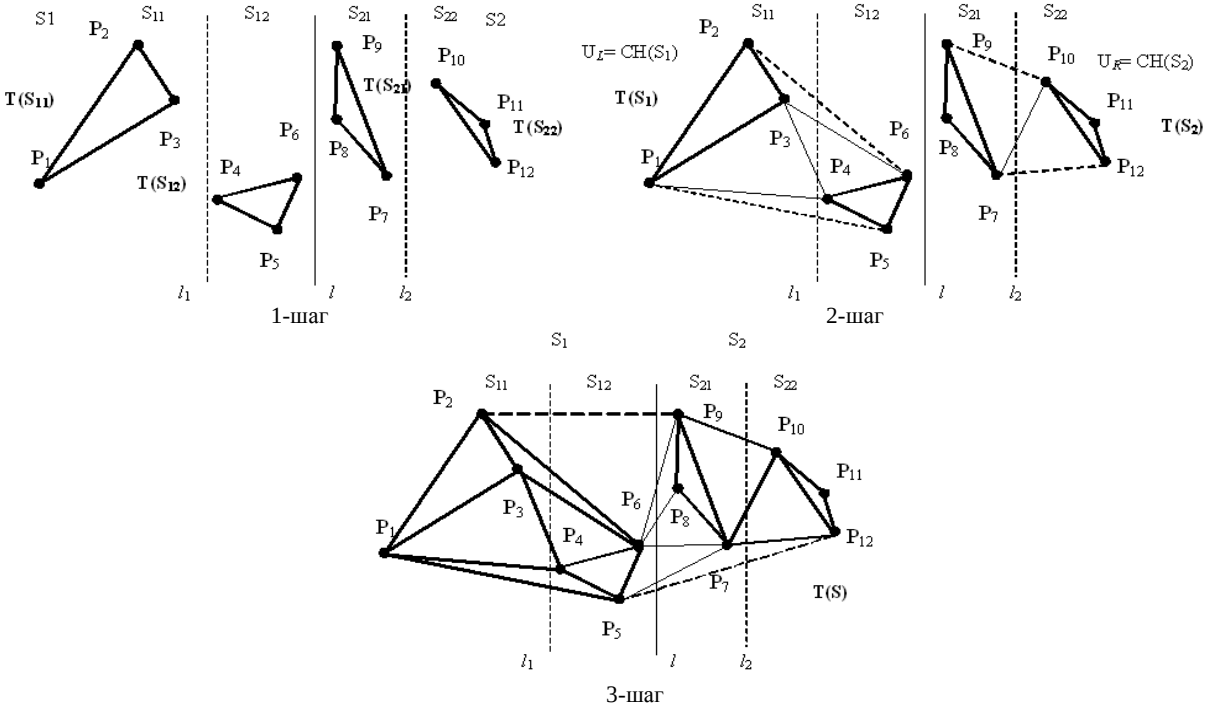


Рис.8. Пошаговое слияние триангуляций для точек из примера рисунка 1.

3. Реализация алгоритма.

Что касается практической реализации разработанного алгоритма, то одним из эффективных подходов применен подход на основе использования технологии ПАРУС (Параллельные Асинхронные Рекурсивно Управляемые Системы) [10]. Этот подход позволяет достаточно просто и эффективно реализовать именно рекурсивно-параллельные алгоритмы решения сложных задач, как на многопроцессорных машинах, так и в компьютерной сети. На рис.9 представлена функциональная схема и основные модули системы ПАРУС-JAVA. Элементы параллельного программирования для реализации этой технологии, детально описаны в работе [10].

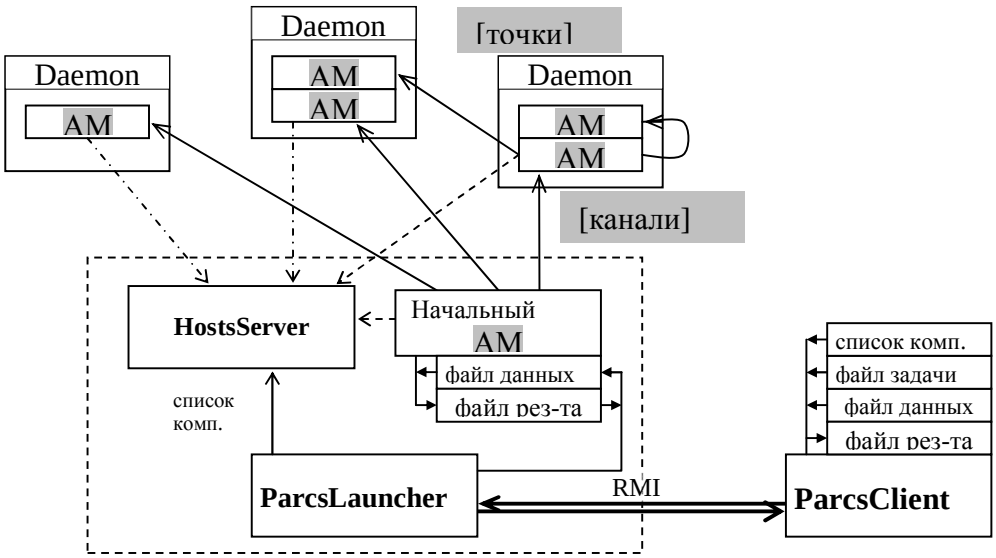


Рис.9. Схема взаимодействия в системе ПАРУС-JAVA

Функциональность основных модулей системы состоит в следующем. Библиотека **parcs** содержит классы и методы, которые используются всеми компонентами системы. Программа **Daemon** запускается в фоновом режиме на компьютерах (хостах), которые подключаются к вычислительной сети. Она запускает алгоритмические модули (АМ) вычислительных задач и тестирует производительность машин. Сервер хостов **HostsServer** — централизованная служба для управления и учета задач и точек. Этот сервер сохраняет информацию о подключенных к вычислительной сети машинах их производительность и количество точек на каждом хосте. Сервер, согласно этой информации, выделяет машины для создания новых точек, а также контролирует удаление точек и начало-конец новых задач. Вычислительная **задача**, подается в виде одного или нескольких jar—файлов, или отдельных классов АМ. На вычисление запускается исходный АМ задачи. В процессе выполнения начального АМ он загружает с физического диска программный код других АМ, которые хранятся в архиве в базе данных. RMI—сервер **ParcsLauncher** используется для удаленного запуска программ на выполнение. RMI-клиент для запуска программы должен передать код вычислительной задачи, запакованный в jar-архив, и может указать, нужно ли ждать конец вычислений, прежде чем вернуть управление клиенту. **Обмен** данными происходит через JAVA—Socket с использованием протокола TCP/IP.

Основными терминами ПАРУС—технологии являются: точка, программный канал (ПК), алгоритмический модуль (АМ). Структура управляющего пространства (УП) — граф, вершины которого - точки УП, а ребра — ПК, которые их соединяют. При этом одни и те же точки могут быть соединены с помощью нескольких ПК разного типа. К каждой точке УП приписан АМ, который является процедурой ПАРУС — расширения базисного языка.

Сначала на всех компьютерах, которые участвуют в вычислениях, запускается Daemon. RMI—клиент до запуска задачи передает на сервер файл данных и список компьютеров. Далее выполняется вызов функции, которая запускает на выполнение задачу, код которой предварительно считан клиентом с jar-файла. Для запуска задачи на сервере запускается HostServer, а потом запускается начальный AM, который передает RMI—клиент или считывается с файла манифеста, принятого jar-архива. Далее выполняются вычисления задач с участием Daemon и управление возвращается RMI—клиенту, а он, при необходимости, может запросить с сервера файл результатов. Возможен также запуск системы в локальной сети без использования RMI технологии. Кроме того, учитывая технологический прогресс в разработке многопроцессорных персональных компьютеров, осуществленный фирмой Intel, перспективным подходом является применение MPI технологии.

4. Заключение

В работе, благодаря удачно выбранным структурам данных: структурированного массива точек на входе дерева алгоритма и поддержки элементов процедур слияния в виде сцепляемой очереди, в каждом узле дерева алгоритма, удалось разработать обобщенный эффективный рекурсивно-параллельный алгоритм синхронного решения комплекса взаимосвязанных задач.

Этот алгоритм решает одновременно на $O(N)$-процессорной системе некоторую совокупность взаимосвязанных задач вычислительной геометрии с эффективностью не хуже чем за $O(\log^2 N)$ времени, то есть принадлежит классу NC_2. Характерной чертой практической реализации данного подхода является то, что разработанный алгоритм позволяет одновременно решать как разные шаги одной процедуры задачи на многих процессорах, так и разные процедуры в одном узле алгоритма с помощью технологий ПАРУС и MPI. Еще одна особенность этой модели алгоритма заключается в том, что этапы 1 и 2 будут общими для решения рассматриваемого множества классов задач, будут отличаться лишь процедуры на этапе слияния. Это и позволяет разработать обобщенный алгоритм решения за единственной схемой целого ряда задач вычислительной геометрии и задач, которые сводятся к ним.

Литература

1. A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing, and C.K. Yap. Parallel computational geometry. Algorithmica 3. 1988. 293-327. Springer-Verlag New York Inc.
2. Препарата Ф., Шеймос Г. Вычислительная геометрия: Введение. М.: Мир, 1989. – 478 с. М.
3. Ajtai, J. Komlos, and E. Szemeredi (1982). An O(nlog(n)) Sorting Network. Proc. 15th ACM Symposium *on Theory Computing, pp.1-9.* Also in Combinatorica, 3(1)(1983), pp. 1-19.
4. T. Leighton (1984). Tight bounds on complexity parallel sorting. *Proc. 16th ACM Symposium on Theory Computing,* pp.71-80.
5. R. Cole (1986). Parallel merge sort. *Proc. 27th IEEE FOCS Symposium,* pp. 511-516.
6. Терещенко В.Н. Построение рекурсивно-параллельных алгоритмов решения задач вычислительной геометрии на основе стратегии «распределяй и властвуй».//Труды международной научной конференции (Санкт-Петербург, 28 января – 1 февраля 2008 г.): Параллельные вычислительные технологии (ПаВТ'2008): стр. 476 - 482.
7. В. В. Воеводин, Вл. В. Воеводин. Параллельные вычисления, СПб., БХВ-Петербург, 2004, 608 с.
8. M. H. Overmars and J. van Leeuwen. Maintenance configurations in plane, *J.Cjmput. and Syst. Sci.* 23, 166-204(1981).
9. А.В. Скворцов. Триангуляция Делоне и ее применение. Томск: из-во Том.ун-та,2002.-128 с.
10. Анисимов А.В., Глушков В.М. Управляющие пространства в асинхронных параллельных вычислениях. // Кибернетика №5, 1980. - С.1-9.