УДК 004.925.8, 004.272.2

**В.Н. ТЕРЕЩЕНКО, А.В. АНИСИМОВ**

# РЕКУРСИЯ И ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ В ЗАДАЧАХ ГЕОМЕТРИЧЕСКОГО МОДЕЛИРОВАНИЯ

**Ключевые слова:** *взаимосвязанные задачи, обобщенный параллельно-рекурсивный алгоритм, геометрическое моделирование, взвешенная сцепляемая очередь.*

## ВВЕДЕНИЕ

Современные вычислительные возможности позволяют ставить и решать новые сложные задачи, для которых необходимо создавать комплексные математические модели. Особенно это относится к задачам визуального моделирования явлений и процессов, определенных в некоторой области геометрического пространства. Речь идет о задачах, исследуемые параметры которых зависят от внутренней структуры и геометрической формы рассматриваемой области и могут изменяться со временем. К ним относятся задачи моделирования теплофизических процессов при сварке различных конструкций из металла, задачи сложного движения жидкости, исследования в ядерной физике и астрофизике, а также задачи исследования биологических и химических процессов, происходящих в живом организме, и другие задачи.

Так, при сварке пластин различных материалов происходят сложные динамические, теплофизические и термомеханические процессы, связанные с изменением фазового состояния вещества, термонапряженного и деформационного состояния, а также со структурными изменениями материала. Здесь зона сварки разделена на область расплава (сварочная ванна), сварочный шов и приграничные с ними области. Для моделирования таких процессов необходимо обрабатывать огромные объемы данных, параметров и различных структурных элементов. Чтобы построить точную визуальную модель рассматриваемых процессов, необходимо разработать такой подход, который бы позволял одновременно решать комплекс геометрических и прикладных задач. Одним из таких подходов может быть создание параллельных алгоритмов. Здесь существует два варианта.

В первом варианте используются совокупности эффективных, но не связанных между собой параллельных алгоритмов для одновременного решения нескольких задач. В этом случае имеется целый набор разных инструментов, каждый из которых может эффективно решать отдельные задачи. На сегодняшний день разработано много эффективных параллельных алгоритмов решения отдельных задач вычислительной геометрии. Так, в работе [1] с использованием схемы «разделяй и властвуй» описаны эффективные алгоритмы решения задач построения выпуклой оболочки для двух- и трехмерных пространств с оценкой сложности $O(\log N)$ и $O(\log^3 N)$ соответственно, а также построения диаграммы Вороного ($O(\log^2 N)$). Кроме того, представлен достаточно глубокий анализ других эффективных алгоритмов решения задач пересечения отрезков, триангуляции полигонов, оптимизации ($O(\log N)$). В работах [2–7] получены улучшенные результаты решения вышеуказанных задач для различных моделей вычислений (CREW, EREW) и для высших размерностей ($d \geq 4$). Ряд статей посвящен общим параллельным методам, которые применимы к решению отдельных задач вычислительной геометрии [8–11].

Однако при построении геометрических и визуальных моделей для исследования сложных явлений и процессов использование отдельных эффективных алгоритмов в общем случае не приносит желаемой эффективности. Отметим, что почти

каждый алгоритм является закрытым относительно выполнения и нуждается в собственной предварительной обработке и создании структуры данных, а также процедурах реализации. Это затрудняет возможность взаимодействия между процедурами алгоритмов и не позволяет получать решение такого класса задач с минимальным временем.

Поэтому второй вариант связан с разработкой универсального инструмента, который имел бы общие средства для эффективного решения всего комплекса взаимосвязанных геометрических и прикладных задач. Другими словами, необходимо выбрать стратегию, которая использовала бы общие средства реализации: структуры данных, отдельные этапы алгоритмов, процедур и некоторые их шаги, а также способы представления результатов. Учитывая отмеченные особенности и то, что большинство задач вычислительной геометрии имеет внутренний параллелизм и предусматривает рекурсивную природу реализации, наиболее подходящей стратегией, на наш взгляд, может быть та, в основе которой лежит параллельно-рекурсивный алгоритм, использующий схему «разделяй и властвуй». Здесь этапы предварительной обработки и разбиения множества будут общими для всего комплекса задач, а на этапе слияния предлагается использовать общую для всех задач структуру данных — взвешенную сцепляемую очередь. Кроме того, результаты отдельных этапов некоторых процедур будут использоваться другими процедурами, что обеспечит высокую эффективность решения.

В настоящей статье для совокупности задач вычислительной геометрии, которые имеют нижнюю оценку сложности $\Omega(N \log N)$, предложен обобщенный параллельно-рекурсивный алгоритм их решения, принадлежащий классу разрешимости порядка $O(\log^2 N)$. В частности, в работах [12, 13] отмечено, что большинство задач вычислительной геометрии в случае параллельной реализации принадлежит этому классу.

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

## 1. ПАРАЛЛЕЛЬНО-РЕКУРСИВНЫЙ АЛГОРИТМ НА ОСНОВЕ СТРАТЕГИИ «РАЗДЕЛЯЙ И ВЛАСТВУЙ»

Идея предложенного алгоритма базируется на стратегии «разделяй и властвуй», которая состоит из двух этапов: рекурсивного разбиения заданного множества точек на подмножества равной мощности и 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 использованию параллельной обработки на этапах разбиения и слияния удалось построить эффективный рекурсивно-параллельный алгоритм, в результате чего снимаются указанные ограничения к применению. Рассмотрим построение этого алгоритма на примере двумерных задач.

**Математическая модель алгоритма.** Математическая модель предлагаемого параллельного алгоритма включает такие основные этапы: предварительная обработка, разбиение множества точек, рекурсивное слияние результатов для подмножеств.

**Этап 1. Предварительная обработка.** Пусть заданы множество $S = \{P_1, P_2, \ldots, P_N\}$ из $N$ точек на плоскости и $O(N)$ процессоров. Формируется упорядоченный массив точек в виде списка $U = \{P_{ij}, i, j = \overline{1, N}\}$, где $i, j$ — индексы, указывающие на порядок точки в массиве по $x$ и $y$ координатам соответственно. Построение отсортированного массива с помощью $O(N)$ процессоров можно осуществить одним из алгоритмов, детально описанных в работах [14–16], которые дают оценки сложности порядка $NC_2$. Сформированный таким способом массив подается на вход алгоритма, граф (дерево) которого представлен на рис. 1. Здесь *NN*, *T*, *CH*, vor — процедуры слияния задач: ближайшие соседи, триангуляция, выпуклая оболочка, диаграмма Вороного соответственно. В этом графе каждый узел обозначен целым числом $k$, относительно которого разбивается список точек в узлах на два равномощных относительно медианы списка после сравнения первых индексов точек массива $P_{ij}$. Значение $k$ определяется за один проход по дереву, если известно количество точек заданного множества, по формуле

$$k = [(m + M)/2], \tag{1}$$

где $m$ и $M$ — соответственно номер первого и номер последнего элементов списка.

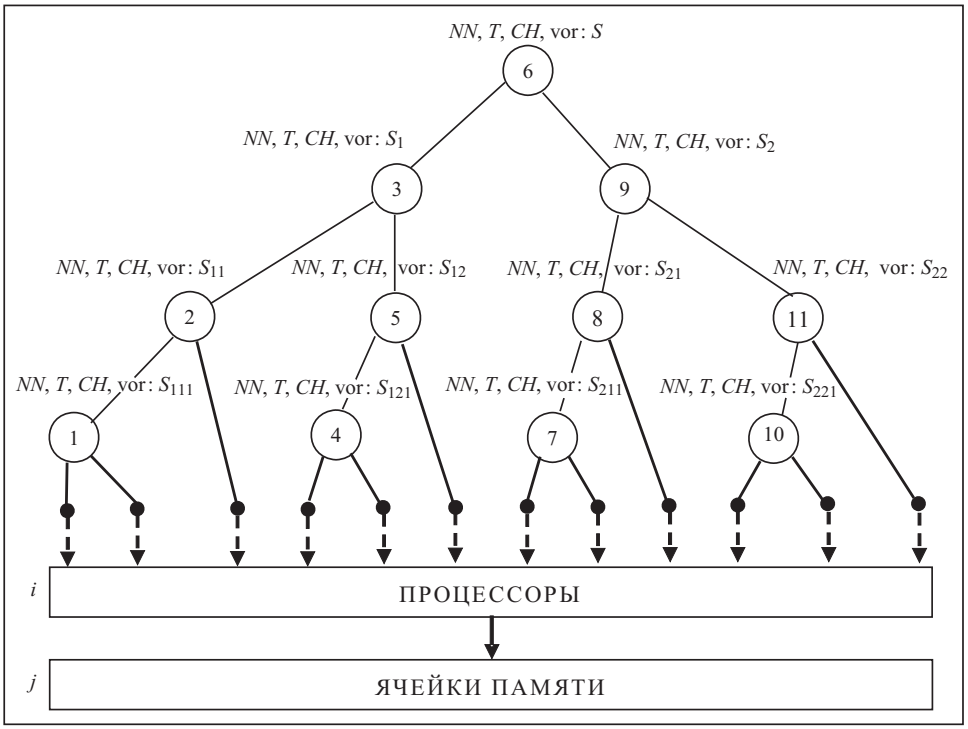


*Рис. 1.* Граф алгоритма

**Этап 2. Разбиение множества точек (рекурсивный спуск).** Происходит разбиение на каждом шаге рекурсии заданного множества точек в виде списка $U$ на подмножества $U_1, U_2$ равной мощности, поиск медианы $l$ и передача $U_1, U_2$ на следующий шаг рекурсии. Процесс разбиения заканчивается, когда в подмножествах остается такое количество точек, для которого все процедуры следующего этапа (слияния) выполняются тривиально. Поиск медианы на упорядоченном по координате $x$ индексированном массиве точек $U$ выполняется за константное время $O(1)$: $l = (P_{kj} + P_{k+1\,j})/2$, где $k$ — номер узла.

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 $O(\log N)$.

**Доказательство.** Пусть задано множество точек в виде индексированного двумерного, упорядоченного массива $U = \{P_{ij}, i, j = \overline{1, N}\}$. Для построения такой структуры данных можно воспользоваться, например, параллельным алгоритмом сортировки со временем $O(\log N)$, предложенным в [16]. Такое представление множества точек позволяет при известном количестве точек $N$ в списке $U$ построить процесс разбиения (см. рис. 1). 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. Ввиду четкой упорядоченности по обоим индексам точек $P_{ij}$ массива $U$ и взаимосвязи 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 $O(\log N)$ для наихудшего ввода данных, что и требовалось 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 на две равные части; $r$ — уровень дерева алгоритма, который соответствует шагу рекурсии; $k$ — номер узла, определяемого соотношением (1); $i$, $j$ — индексы элементов массива $U = \{P_{ij}, i, j = \overline{1, N}\}$.

При $n > 1$ всем процессам необходимо выполнить следующие операции.

1. Присвоить номер $k$ каждому узлу дерева алгоритма согласно (1).

2. На $r$-м шаге алгоритма в узле $k$ определить медиану $l$ множества $S(U)$ по формуле $l = (P_{kj} + P_{k+1\,j})/2$.

3. На $r$-м шаге алгоритма в узле $k$ необходимо сравнить значение $i$-го индекса $P_{ij}$ элементов массива $U$ с числом $k$:

а) если $i \le k$, то послать каждый элемент $P_{ij} \in U$, индекс которого удовлетворяет этому условию, в узел «ЛЕВЫЙ СЫН» по $i$-му каналу и записать в $j$-ю ячейку памяти узла;

б) если $i > k$, то послать каждый элемент $P_{ij} \in U$, индекс которого удовлетворяет этому условию, в узел «ПРАВЫЙ СЫН» по $i$-му каналу и записать в $j$-ю ячейку памяти узла.

4. При $n = 1$ происходит завершение работы этого этапа алгоритма.

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

**Этап 3. Рекурсивное слияние результатов для подмножеств (рекурсивный подъем).** Полученные результаты решения соответствующей задачи (выпуклой оболочки, диаграммы Вороного, декомпозиции и т.д.) с листовых узлов дерева алгоритма подаются в родительские узлы, где используются соответствующие процедуры слияния для построения общего решения задачи. Процесс заканчивается слиянием результатов в корневом узле. В частности, в работах [17–19] уже предложены эффективные процедуры слияния для таких задач, как построение выпуклой оболочки, нахождение ближайшей пары, триангуляция и другие. Ниже детально будут описаны процедура слияния диаграммы Вороного (поскольку сама диаграмма и алгоритм ее построения используются для решения других важных задач вычислительной геометрии) и процедура слияния задачи о всех ближайших соседях. А процедуры слияния для задач построения выпуклой оболочки и триангуляции будут описаны лишь на уровне идей. Особенность предложенных процедур состоит в применении сцепляемой очереди для представления точек в

узлах алгоритма, что позволяет выполнять все операции за логарифмическое время. Вопрос использования оптимального состава процессоров (компьютеров) в данной статье не рассматривается и нуждается в дополнительном исследовании. В то же время из работы [20] известно, что если считать $w$ общим числом операций алгоритма, а $O(F(N))$ — оценкой сложности алгоритма для неограниченного количества процессоров, то время выполнения алгоритма для $p$ процессоров можно выразить соотношением

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

## 2. ПОСТРОЕНИЕ ПРОЦЕДУР СЛИЯНИЯ

Особенность рассматриваемого алгоритма заключается в том, что между процедурами слияния представленного комплекса задач существует связь на уровне как отдельных шагов выполнения, так и операций. И связующим звеном здесь выступает процедура слияния выпуклых оболочек, поскольку она при построении всех процедур слияния используется как один из ключевых шагов алгоритма. Поэтому их построение для рассматриваемого рекурсивно-параллельного алгоритма начнем с процедуры слияния выпуклых оболочек. Центральное место, как образующая процедура, занимает процедура слияния и построения диаграммы Вороного, поскольку она позволяет строить процедуры слияния других сложных задач рассматриваемой совокупности. Изложим это детально, а кратко рассмотрим процедуры слияния триангуляции и поиска всех ближайших соседей.

**2.1. Процедура слияния задачи построения выпуклой оболочки.** Известные параллельные алгоритмы построения выпуклой оболочки множества точек в евклидовом пространстве $E^d$, использующие идею «разделяй и властвуй», так или иначе связаны с нахождением опорных ребер границ двух оболочек. Здесь (для двумерного случая) можно отметить работы Р. Миллера и К.Ф. Стоута [21] (с оценкой слияния $O(\log N)$), О. Беркмана [6] ($O(\log\log N)$), М.Т. Гудриха и М. Гоуса [22] (с оценкой $O(\log^2 N)$ с предварительной сортировкой и рандомизированный алгоритм порядка $O(\log N)$), а также алгоритм Д. Чена [5]. Для случая высших измерений уместно выделить работу [4], где для $d \geq 3$ получена высокая скорость выполнения алгоритмов — порядка $O(\log^2 N)$. Но решающим в вопросе построения эффективных параллельных алгоритмов, безусловно, является замечательная идея М. Овермарса и Дж. Ван Лювена [23], которая позволяет использовать в качестве структуры данных сцепляемую очередь.

Для построения процедуры слияния выпуклых оболочек рассматриваемого параллельного алгоритма применялись лишь некоторые элементы из последовательного алгоритма, предложенного в работе [23]. В частности, в алгоритме была предложена процедура слияния верхней ( нижней ) выпуклой оболочки множества точек при решении динамической задачи с оценкой $O(\log N)$. Такая эффективность достигается за счет использования структуры данных в виде сцепляемой очереди, что позволяет выполнять все необходимые операции на каждом шаге за логарифмическое время.

На каждом шаге рекурсивного подъема, начиная со второго, на вход некоторого родительского узла $v$ дерева алгоритма (см. рис. 1) подаются выпуклые оболочки от левого $U_L$ ($U_L = \text{ЛСЫН}[v]$) и правого $U_R$ ($U_R = \text{ПСЫН}[v]$) сыновей соответственно. Для построения общей выпуклой оболочки сыновей узла $v$ необходимо определить верхние (нижние) опорные вершины для левой и правой выпуклых оболочек. Далее следует расцепить взаимно выпуклые цепи между этими опорными точками, а оставшиеся цепи соединить соответствующими опорными отрезками. 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 $O(\log N)$. При выполнении поиска опорных вершин и построении опорных отрезков в данном случае используется такая же классификация вершин, как и в работе [23]. После определения опорных то-

чек и сцепления с помощью их выпуклых оболочек сыновей узла $v$ удаляются правая и левая части деревьев, которые поддерживают соответственно $U_L$ и $U_R$ между опорными точками. Балансирование деревьев, которые будут поддерживать верхнюю и нижнюю выпуклые оболочки узла $v$, осуществляется путем слияния оставшихся частей деревьев. Все приведенные операции выполняются за время $O(\log N)$. Полученные таким способом выпуклые оболочки передаются на следующий уровень рекурсии дерева алгоритма (см. рис. 1).

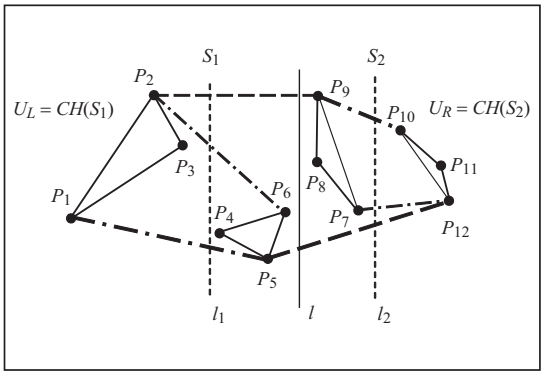


*Рис. 2.* Пример этапа слияния выпуклых оболочек

На рис. 2 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.

На рис. 3 показана схема поиска опорных вершин левой и правой выпуклых оболочек, представленных на рис. 2, до слияния (*а*) и после слияния (*б*).

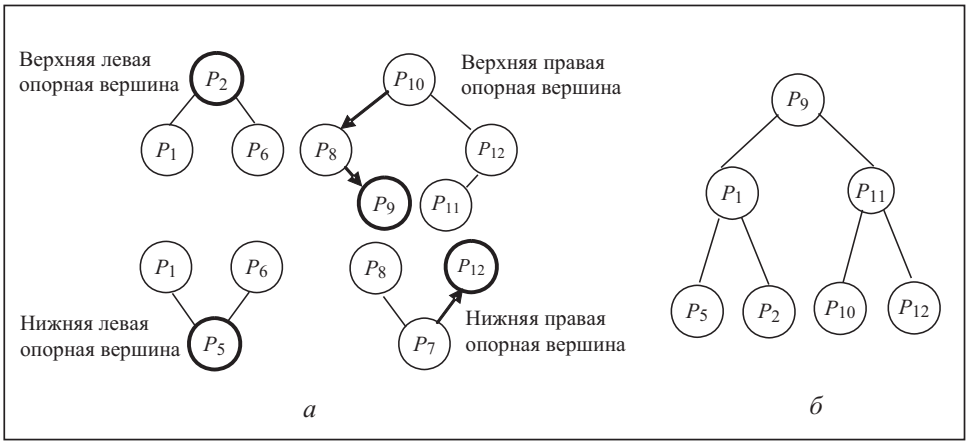


*Рис. 3.* Сцепляемые очереди выпуклых оболочек до слияния (*а*) и после слияния (*б*)

**Лемма 2.** Этап рекурсивного слияния результатов нахождения выпуклой оболочки множества $S$ из $N$ точек на плоскости с помощью $O(N)$ процессоров можно выполнить за время $O(\log^2 N)$.

Более детально процедура слияния построения выпуклой оболочки для рассматриваемого параллельно-рекурсивного алгоритма описана в работе [17].

**2.2. Процедура слияния задачи триангуляции.** Задача триангуляции на ограниченном множестве точек достаточно изучена, и для ее решения существует ряд эффективных последовательных алгоритмов. Здесь следует отметить публикации А.В. Скворцова, в частности работу [24], в которой предложено несколько быстрых алгоритмов триангуляции. Среди параллельных алгоритмов триангуляции хорошие результаты получены М.Т. Гудрихом [25]. В настоящей статье предлагается процедура триангуляции для рассматриваемого параллельного алгоритма в случае асимптотически больших множеств точек (или множеств точек, представляющих структурированные объекты). Процедура слияния задачи триангуляции детально описана в работе [19], поэтому здесь коснемся лишь ее идеи.

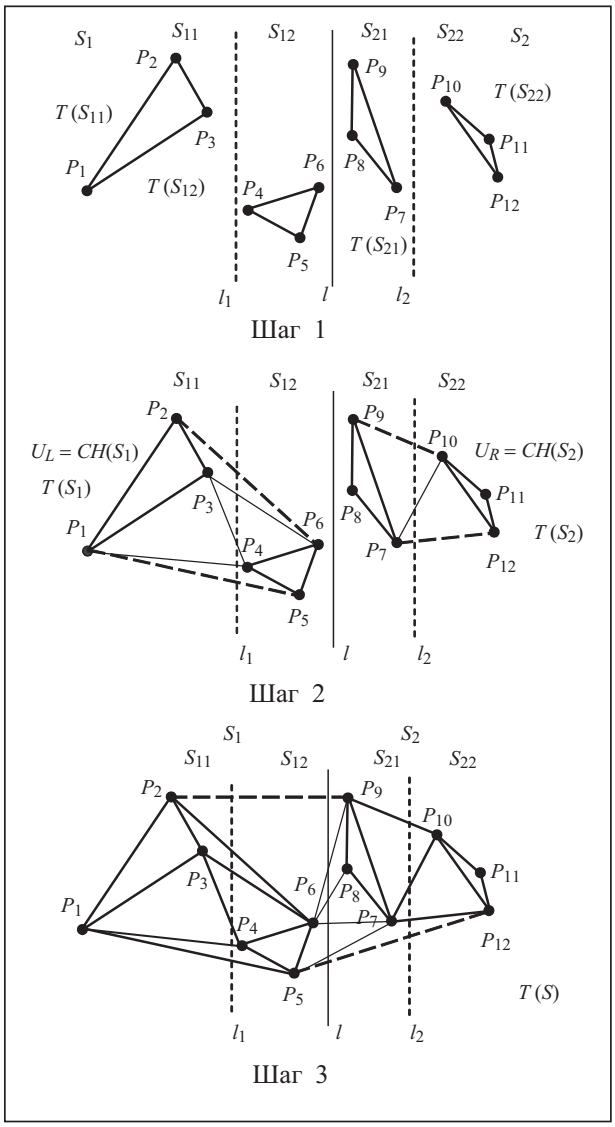


*Рис. 4*. Пример пошагового слияния триангуляций

На каждом шаге рекурсивного подъема, начиная со второго, на вход родительского узла $v$ дерева (см. рис. 1) подаются триангуляции от левого $T(S_L)$ и правого $T(S_R)$ сыновей соответственно, которые ограничены границами выпуклых оболочек $CH(S_L)$ и $CH(S_R)$. Для построения триангуляции на основе триангуляций сыновей узла $v$ необходимо определить верхние и нижние опорные вершины $CH(S_L)$ и $CH(S_R)$ соответствующих триангуляций и соединить их верхним и нижним опорными отрезками. Каждый процессор, двигаясь по одной из взаимно выпуклых цепей между верхним и нижним опорными отрезками, проводит отрезок к своей вершине другой цепи. Процесс продолжается до тех пор, пока не будет достигнут нижний опорный отрезок. Полученные таким способом триангуляции передаются на следующий уровень рекурсии.

Для выполнения вышеописанных операций наиболее подходящей (как и для предыдущей процедуры) будет сцепляемая очередь, которая реализует этап слияния триангуляций за логарифмическое время. На рис. 4 продемонстрирован пример пошагового этапа слияния триангуляций.

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

**2.3. Процедура слияния задачи построения диаграммы Вороного.** Схема «разделяй и властвуй» имеет успешное применение и при построении диаграммы Вороного для множества $S$ из $N$ точек на плоскости. В частности, М. Шеймосом [26] предложен эффективный последовательный алгоритм решения этой задачи на однопроцессорной машине со временем $\theta(N \log N)$, а в работах [1, 4, 27] описаны эффективные параллельные алгоритмы со временем $\theta(\log^2 N)$. Такая эффективность достигается в результате детального анализа границ диаграмм Вороного в зоне разделяющей прямой. В данном разделе этап слияния алгоритма построения диаграммы Вороного отличается от этапа слияния задачи выпуклой оболочки 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 о выпуклой оболочке, используются для следующих шагов задачи построения диаграммы Вороного.

**Постановка задачи.** На плоскости задано множество $S$ из $N$ точек. Необходимо 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 $\mathrm{vor}(S_L)$ и правого сына $\mathrm{vor}(S_R)$. Необходимо построить ДВ для узла $v$. Время, которое расходуется на этап рекурсивного слияния решения задачи в случае параллельного алгоритма, определяется следующей леммой.

**Лемма 4.** Этап рекурсивного слияния диаграмм Вороного множества $S$ из $N$ точек на плоскости с помощью $O(N)$ процессоров можно выполнить за время $O(\log^2 N)$.

**Procedura** (*СЛИЯНИЕ_ДИАГРАММА ВОРОНОГО*) $(\mathrm{vor}(S_L),\ \mathrm{vor}(S_R))$

Для построения слияния диаграмм Вороного в узле $v$ необходимо следующее.

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 $\mathrm{vor}(S_L)$ или $\mathrm{vor}(S_R)$ (рис. 5).

3. Провести 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 принадлежат взаимно выпуклым цепям выпуклых оболочек сыновей, а также точек, определяемых ребрами диаграмм $\mathrm{vor}(S_L)$, $\mathrm{vor}(S_R)$, которые пересекают ребра этих цепей.

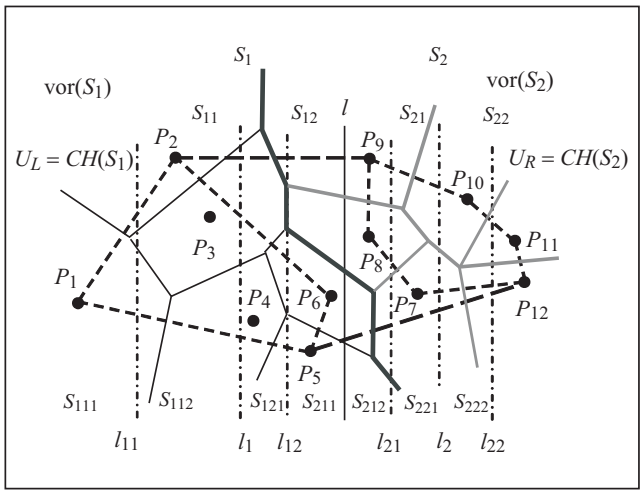


*Рис. 5.* Пример слияния диаграмм Вороного

4. Полученные после слияния таким способом диаграммы Вороного передать на следующий уровень рекурсии (см. рис. 1).

Процесс слияния завершается, когда будет построена разделяющая цепь для корня дерева, а значит и диаграмма Вороного множества точек $S$.

Рассмотрим более детально шаг 3.

**Лемма 5.** Построение разделяющей цепи $\sigma(S_1, S_2)$, которая «сшивает» диаграммы Вороного $\mathrm{vor}(S_L)$, $\mathrm{vor}(S_R)$, 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 $\mathrm{vor}(S_L)$, $\mathrm{vor}(S_R)$ некоторого узла $v$ графа алгоритма имеем две взаимовыпуклые цепи, которые и определяют область построения $D$ разделяющей цепи (рис. 6). Каждая цепь определяет упорядоченный

набор ребер диаграммы Вороного, направленных в середину области $D$ и пересекающих ребра взаимно выпуклых цепей $CH_L$, $CH_R$. Каждое ребро цепей $CH_L$, $CH_R$ может пересекаться с ребрами диаграмм Вороного и тем самым определять множество точек, разделенное этими ребрами. Обозначим множество вершин, которое состоит из вершин выпуклой цепи $CH_L$ ($CH_R$) и точек, определенных ребрами диаграмм Вороного, которые пересекают цепь, через $B_L(S_1)$ ($B_R(S_2)$) и назовем его левопредельным (правопредельным) упорядоченным множеством точек (списком). Соединив последовательно точки в списках $B_L(S_1)$ и $B_R(S_2)$, получим списки ребер $E_L(S_1)$, $E_R(S_2)$, 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; $C_L$ и $C_R$ — левая и правая монотонные цепи между опорными отрезками, для которых строится разделяющая цепь. Верхняя цепь $\sigma_A=\{a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8\}$ и нижняя цепь $\sigma_B=\{b_1,b_2,b_3,b_4,b_5\}$ соответствуют парам монотонных цепей $(C_{L1},C_{R1})$ и $(C_{L2},C_{R2})$.

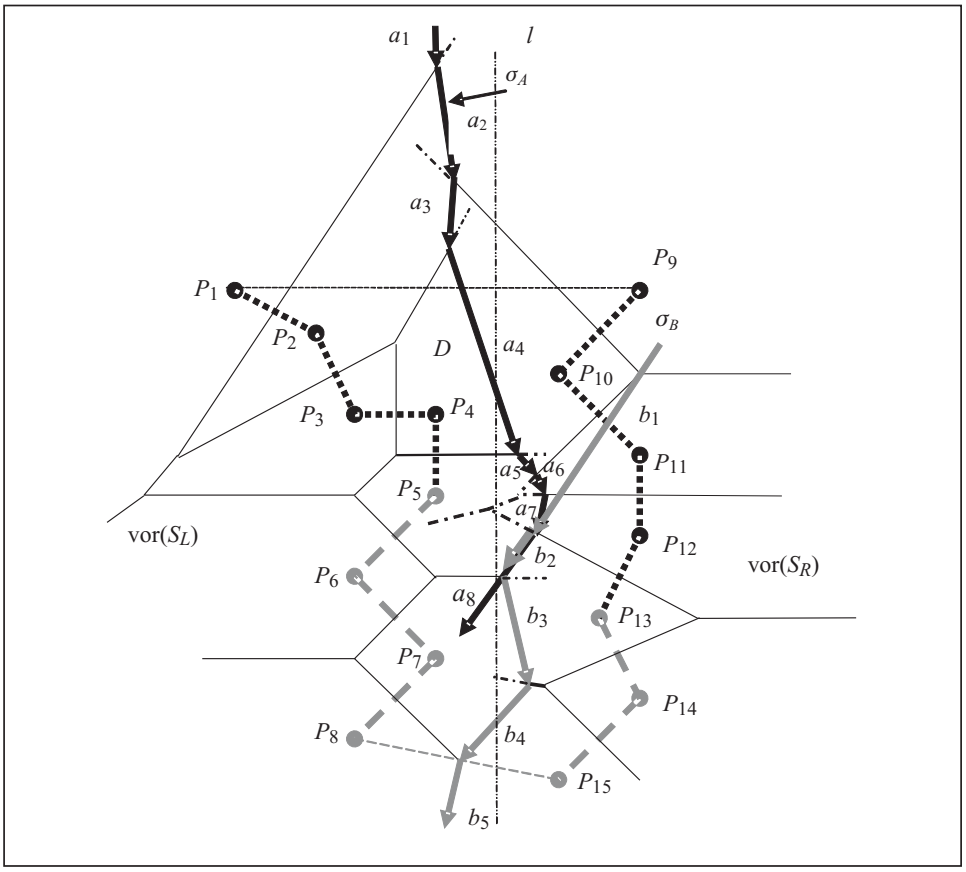


*Рис. 6.* Шаг слияния двух разделяющих цепей в области $D$

**Лемма 6.** Цепи $C_L$ и $C_R$ являются монотонными относительно прямой $l$.

**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 $\sigma$ не будет монотонной, что противоречит условию.

Следует отметить, что процедуру слияния в каждом узле дерева алгоритма можно выполнять независимо и параллельно на нескольких процессорах. Для выполнения вышеприведенных 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 необходимой информацией. Сцепляемые очереди обеих монотонных цепей представляют бинарное дерево с корнем, в узлах которого имеем координаты вершин, а дуги представляют соответствующие ребра $e_k$ $(k = \overline{1,N})$ из списков цепей $E_L(S_1)$ и $E_R(S_2)$. Кроме того, узлы загружены указателями на соответствующие ребра диаграммы Вороного. Обозначим $d_{ij}$, $i, j \in N$, имена ребер $\mathrm{vor}(S_1)$, $\mathrm{vor}(S_2)$, которые лежат в области $D$.

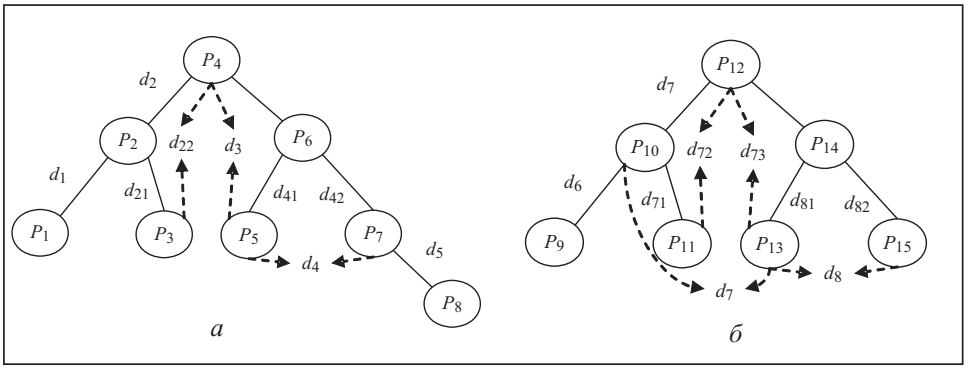


*Рис. 7.* Сцепляемые очереди для левой $C_L$ (*а*) и правой $C_R$ (*б*) цепей области слияния $D$ диаграмм Вороного $\mathrm{vor}(S_1)$ и $\mathrm{vor}(S_2)$ на основе рис. 6

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 также представить в виде бинарного дерева. В листьях дерева каждым из $O(N)$ 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 для каждого узла 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 будут поддерживать верхнюю и нижнюю выпуклые оболочки узла $v$, образуются путем слияния оставшихся частей деревьев. Полученные таким образом деревья поддерживают выпуклую оболочку и ребра диаграммы Вороного, пересекающие ее, а также определяют ребра монотонных цепей области $D$, что позволяет выполнять процедуры построения разделяющей цепи за время $O(\log N)$.

**2.4. Процедура слияния задачи о всех ближайших соседях.** В работах [1, 26] в результате детального анализа качественных возможностей диаграммы Вороного сделан достаточно важный вывод, который можно сформулировать в виде следующего обобщенного утверждения.

**Теорема 1.** С помощью диаграммы Вороного множества $S$ из $N$ точек в евклидовой плоскости такие задачи близости как ближайшая пара, все ближайшие соседи, евклидово минимальное остовое дерево, триангуляция, а также задача построения выпуклой оболочки можно решить за оптимальное время $\theta(N \log N)$.

Детальное доказательство этого результата для однопроцессорной вычислительной машины приведено в работе [26]. В основе этого доказательства лежит

принцип сводимости. Этот вывод позволяет применить диаграмму Вороного как инструмент для решения широкого класса задач вычислительной геометрии. В настоящей статье этот инструмент используется при разработке процедуры слияния для задачи «все ближайшие соседи». Так как первый и второй этапы рассматриваемого параллельно-рекурсивного алгоритма общие для всего комплекса задач, то рассмотрим лишь построение процедуры слияния этой задачи, которое состоит в следующем.

На каждом шаге рекурсивного подъема, начиная со второго, на вход родительского узла $v$ графа алгоритма (см. рис. 1) подаются диаграммы Вороного от левого и правого сыновей $\mathrm{vor}(S_1)$ и $\mathrm{vor}(S_2)$, а также ближайший сосед каждой точки подмножеств $S_L$ и $S_R$. Строится диаграмма Вороного для узла $v$ и параллельно определяются новые соседи на границах подмножеств $S_L$ и $S_R$ относительно $l$.

При построении разделяющей цепи диаграммы Вороного параллельно осуществляется поиск ближайших соседей среди точек подмножеств $S_L$ и $S_R$, которые образуют текущую пару ребер цепи $\sigma(S_L, S_R)$. При этом следующая пара, определяющая новое ребро разделяющей цепи, проверяется на наличие нового ближайшего соседа. На графе алгоритма (см. рис. 1) множество пар ближайших соседей для $S$ обозначено $NN(S)$ и для подмножеств $S_L$ и $S_R$ — соответственно $NN(S_L)$, $NN(S_R)$. На рис. 8 дан пример процесса нахождения ближайших соседей на этапе слияния. Как видим, для $P_8 \in S_R$ найден новый ближайший сосед $P_6 \in S_L$.

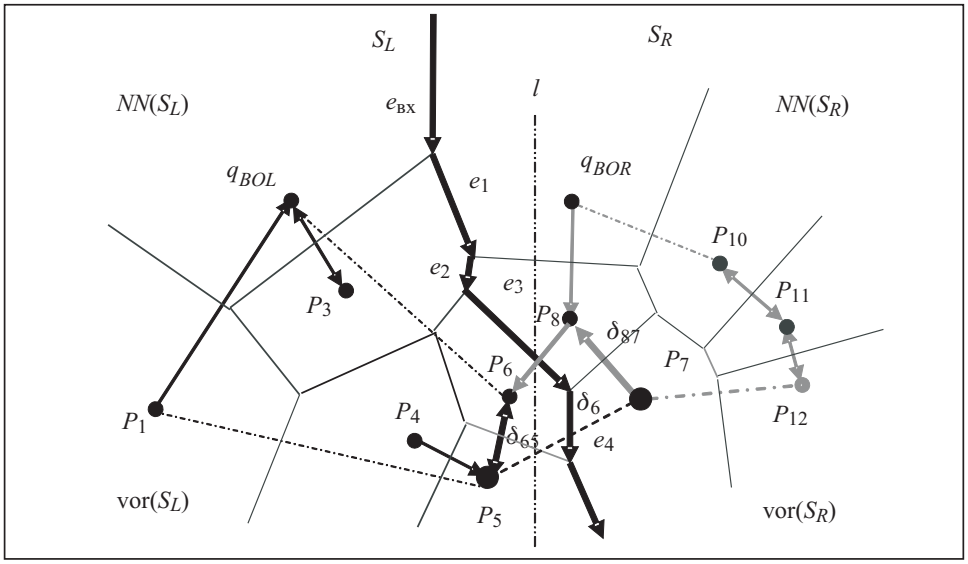


*Рис. 8.* Пример нахождения ближайших соседей для пар точек $(P_3, q_{BOR})$, $(P_3, P_8)$, $(P_6, P_8)$, $(P_6, P_7)$, $(P_5, P_7)$

Поскольку время построения монотонной разделяющей цепи на шаге слияния двух диаграмм Вороного с помощью $O(N)$ процессоров равно $O(\log N)$, то и время слияния результатов нахождения ближайших соседей имеет тот же порядок.

Действительно, каждое ребро разделяющей цепи определяет поиск ближайшего соседа для пары точек, которую это ребро разделяет. Достаточно лишь сравнить величины двух расстояний. Таким образом, можно сделать вывод.

**Лемма 7.** Этап рекурсивного слияния результатов нахождения ближайших соседей для каждой точки множества $S$ из $N$ точек на плоскости с помощью $O(N)$ процессоров можно выполнить за время $O(\log^2 N)$.

### 3. РЕАЛИЗАЦИЯ АЛГОРИТМА

Для практической реализации разработанного алгоритма применялся один из эффективных подходов, основанный на использовании технологии программирова-

ния в стандарте MPI. Этот подход позволяет достаточно легко реализовать именно рекурсивно-параллельные алгоритмы решения сложных задач как на многопроцессорных машинах (кластерах), так и в компьютерной сети. В настоящей статье проведено несколько исследований относительно скорости решения разработанного алгоритма для целого комплекса задач вичислительной геометрии, используя различные способы реализации программы.

Главная функция этой программы main (int argc, char *argv[]) условно разделена на три части. Первая часть функции — осуществление инициализации среды MPI непосредственно после объявления самой функции . Эта часть функции выполняется всеми процессорами.

Вторая часть функции — это та часть программы, которая выполняется лишь на нулевом процессоре и непосредственно не участвует в процессе слияния двух подзадач. Программа работает следующим образом: сначала идет проверка функциями is_free_proc и get_free_reg на наличие свободных процессоров и свободных регионов. Если таковые отсутствуют, то ожидается поступление результатов от других процессоров. При этом идет распаковка данных, освобождается память от уже обработанных и ненужных регионов, повышается уровень рекурсии региона, который обрабатывает другой процессор (поле level типа zone), а в массиве procs фиксируется, что $i$-й процессор освободился.

На следующем шаге происходит переход к стадии передачи данных свободному процессору свободных регионов с наименьшим возможным значением поля level и снова выполняется проверка возможности последующих вычислений и прием сообщений от других процессоров до тех пор, пока не остается один регион.

Третья часть функции — описание действия других процессоров с ненулевым номером. Здесь циклически происходит прием данных от нулевого процессора, распаковка данных, распределение данных (рекурсивный спуск), во время которого заполняется динамический список с заглавием head1. Вызываются функции слияния соответствующих задач, где сливаются результаты, полученные от нулевого процессора. При этом параллельно одна процедура слияния может использовать результаты другой. После завершения работы процедур слияния алгоритма происходит запаковка данных, которые посылаются нулевому процессору, и снова ожидаются данные от такого процессора для обработки.

Программа имеет модульный характер и позволяет в будущем использовать ее для решения трудоемких задач вычислительной геометрии и компьютерного моделирования, которые разрешимы с помощью схемы «разделяй и властвуй». С учетом сложности новых решаемых задач и расширения области применения задач вычислительной геометрии актуальность таких решений возрастает. Особенно это касается вычислительной точности и эффективности решения рассмотренных задач.

## ЗАКЛЮЧЕНИЕ

В настоящей статье 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 асинхронного 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, т.е. принадлежит классу $NC_2$. Характерным для практической реализации данного подхода является то, что разработанный алгоритм позволяет одновременно решать с помощью технологий MPI как разные шаги одной процедуры задачи на многих процессорах, так и разные процедуры в одном узле алгоритма.

Другая особенность этой модели алгоритма заключается в том, что этапы 1 и 2 являются общими для решения рассматриваемого множества классов задач и отличаются лишь процедурами на этапе слияния. Это и позволяет разработать обобщенный алгоритм решения за единой схемой целого ряда задач вычислительной геометрии и задач, которые сводятся к ним.

СПИСОК ЛИТЕРАТУРЫ


1. Parallel computational geometry / A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing, C.K. Yap // Algorithmica. — 1988. — **3.** — P. 293–327.
2. Atallah M.J., Cole R., Goodrich M.T. Cascading divide-and-conquer: A technique for designing parallel algorithms // SIAM J. Comput. — 1989. — **18.** — P. 499–532.
3. Cole R., Goodrich M.T. Optimal parallel algorithms for polygon and point-set problems // Algorithmica. — 1992. — **7.** — P. 3–23.
4. Amato N.M., Goodrich M.T., Ramos E.A. Parallel algorithms for higher-dimensional convex hulls // Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci., 1994. — P. 683–694.
5. Chen D. Efficient geometric algorithms on the EREW PRAM // IEEE Trans. Parallel Distrib. Syst. — 1995. — **6.** — P. 41–47.
6. Berkman O., Schieber B., Vishkin U. A fast parallel algorithm for finding the convex hull of a sorted point set // Internat. J. Comput. Geom. Appl. — 1996. — **6.** — P. 231–242.
7. Goodman J.E., O'Rourke J. Handbook of discrete and computational geometry. — N.Y.: Chapman and Hall/CRC Press, 2004. — 1497 p.
8. Akl S.G., Lyons K.A. Parallel computational geometry. — Englewood Cliffs: Prentice-Hall, 1993. — 224 p.
9. Jaja J. An introduction to parallel algorithms. — Amsterdam: Addison-Wesley, 1992. — 566 p.
10. Van Leeuwen J. Handbook of theoretical computer science. — Amsterdam: Elsevier/The MIT Press, 1990. — 1294 p.
11. Reif J.H. Synthesis of parallel algorithms. — San Mateo: Morgan Kaufmann, 1993. — 1011 p.
12. Ben-Or M. Lower bounds for algebraic computational trees // Proc. 15th ACM Symposium on Theory Computing, 1983. — P. 80–86.
13. Kozen D., Yap C.K. Algebraic cell decomposition in NC // Proc. 26th IEEE FOCS Symposium, 1985. — P. 515–521.
14. Ajtai M., Komlos J., Szemeredi E. An $O(n\log(n))$ sorting network // Combinatorica. — 1983. — **3,** N 1. — P. 1–19.
15. Leighton T. Tight bounds on complexity parallel sorting // Proc. 16th ACM Symposium on Theory Computing, 1984. — P. 71–80.
16. Cole R. Parallel merge sort // Proc. 27th IEEE FOCS Symposium, 1986. — P. 511–516.
17. Терещенко В.М. Один підхід в розробці ефективного рекурсивно-паралельного алгоритму побудови опуклої оболонки множини точок на площині // Тавричес. вестник информатики и математики. — 2007. — № 2. — С. 24–32.
18. Терещенко В.Н. Построение рекурсивно-параллельных алгоритмов решения задач вычислительной геометрии на основе стратегии «распределяй и властвуй» // Тр. междунар. науч. конф. ПаВТ'2008, 2008. — С. 476–482.
19. Терещенко В.М. Узагальнений підхід розв'язання деяких задач обчислювальної геометрії на основі рекурсивно-паралельної технології. Ч. 2 // Наукові нотатки. — 2008. — № 22. — С. 344–349.
20. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. — 608 с.
21. Miller R., Stout Q.F. Efficient parallel convex hull algorithms // IEEE Trans. Comput. — 1988. — **37.** — P. 1605–1618.
22. Ghouse M., Goodrich M.T. In-place techniques for parallel convex hull algorithms // Proc. 3rd Annu. ACM Sympos. Parallel Algorithms Architect, 1991. — P. 192–203.
23. Overmars M.H., Van Leeuwen J. Maintenance configurations in plane // J. Comput. and Syst. Sci. — 1981. — **23.** — P. 166–204.
24. Скворцов А.В. Триангуляция Делоне и ее применение. — Томск: Изд-во Томск. ун-та, 2002. — 128 с.
25. Goodrich M.T. Planar separators and parallel polygon triangulation // J. Comput. and Syst. Sci. — 1995. — **51**, N 3. — P. 374–389.
26. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение. — М.: Мир, 1989. — 478 с.
27. Tereshchenko V. One tool for building visual models // Proc. International Conf. on Comput. Intelligence, Modelling and Simulation. IEEE CS, 2009. — P. 59–62.


*Поступила 06.10.2009*