УДК: 519.17, 004.932

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ПОСТРОЕНИЯ МИНИМАЛЬНОГО ЕВКЛИДОВА ДЕРЕВА В ЗАДАЧЕ КЛАССИФИКАЦИИ ОБЪЕКТОВ НА ИЗОБРАЖЕНИЯХ ДЛЯ GPU

А.А. БАРСУК
О.Н. ИВАНОВ

Белгородский государственный национальный исследовательский университет

e-mail:
barsuk_alex@mail.ru

В статье рассматривается подход к реализации быстрого параллельного алгоритма поиска евклидова дерева в задаче классификации объектов на изображениях. Производится краткий обзор используемых в настоящее время алгоритмов, выдвигаются рекомендации по их модификации, формулируются требования к вычислительной системе. Проведены вычислительные эксперименты с GPU реализацией параллельного модифицированного алгоритма Борувки.

Ключевые слова: минимальное остовное дерево, евклидово дерево, классификация, алгоритм Прима, алгоритм Крускала, алгоритм Борувки, GPU, CUDA.

Одним из подходов к решению задачи классификации (кластеризации) является подход, основанный на использовании теории графов. Кластеризацию можно рассматривать как задачу расчленения графа на части. По сути, каждый классифицируемый объект соотносится с вершиной взвешенного графа, где весовые коэффициенты ребер графа между элементами малы, если элементы подобны, и велики в противном случае. Далее граф необходимо разрезать на связанные компоненты с относительно малыми внутренними весовыми коэффициентами (компоненты соответствуют кластерам), разрезая для этого ребра с относительно большими весовыми коэффициентами [4].Отличительной особенностью результатов работы таких алгоритмов является то, что они выделяют классы с формами, сильно отличающимися от сферических, то есть с вычурными формами или линейно неразделимыми.

Классифицируемые объекты представляют собой вектора в n-мерном признаковом пространстве с заданной на нем функцией расстояния. Они формируют множество вершин графа. Ребрами является подмножество множества соединений между всеми парами объектов, выбранное в соответствии с некоторым принципом. Одним из используемых принципов является условие минимальности суммы весов ребер в графе. Получаемый в результате использования такого подхода граф называется минимальным остовным деревом (minimal spanning tree), а задача нахождения такого графа – задачей построения (нахождения) минимального остовного дерева.

Мы можем смоделировать эту задачу при помощи связного неориентированного графа $G = (V, E)$, где V — множество вершин (точек), E — множество возможных соединений между парами вершин (ребер). Каждое ребро $(u, v) \in E$ обладает весом $w(u, v)$ определяющим расстояние между u и v. Задача поиска минимального остовного дерева сводится к нахождению ациклического подмножества $T \subseteq E$, которое соединяет все вершины и чей общий вес минимален [2].

$$w(T) = \sum_{(u,v)\in T} w(u, v) = \min$$

Существует три классических алгоритма для решения задачи поиска минимального остовного дерева: алгоритм Прима, алгоритм Крускала, алгоритм Борувки.

Алгоритм Прима имеет очень простой вид. Искомый минимальный остов строится постепенно, добавлением в него рёбер по одному. Изначально остов пола-

гается состоящим из единственной вершины (её можно выбрать произвольно). Затем выбирается ребро минимального веса, исходящее из этой вершины, и добавляется в минимальный остов. После этого остов содержит уже две вершины, и теперь ищется и добавляется ребро минимального веса, имеющее один конец в одной из двух выбранных вершин, а другой — наоборот, во всех остальных, кроме этих двух. И так далее, т.е. всякий раз ищется минимальное по весу ребро, один конец которого — уже взятая в остов вершина, а другой конец — ещё не взятая, и это ребро добавляется в остов (если таких рёбер несколько, можно взять любое). Этот процесс повторяется до тех пор, пока остов не станет содержать все вершины. На рисунке 1 представлена схема работы алгоритма Прима, каждое пронумерованное изображение иллюстрирует строящееся дерево на некоторой итерации алгоритма. Предполагается, что между каждой парой вершин существует ребро.

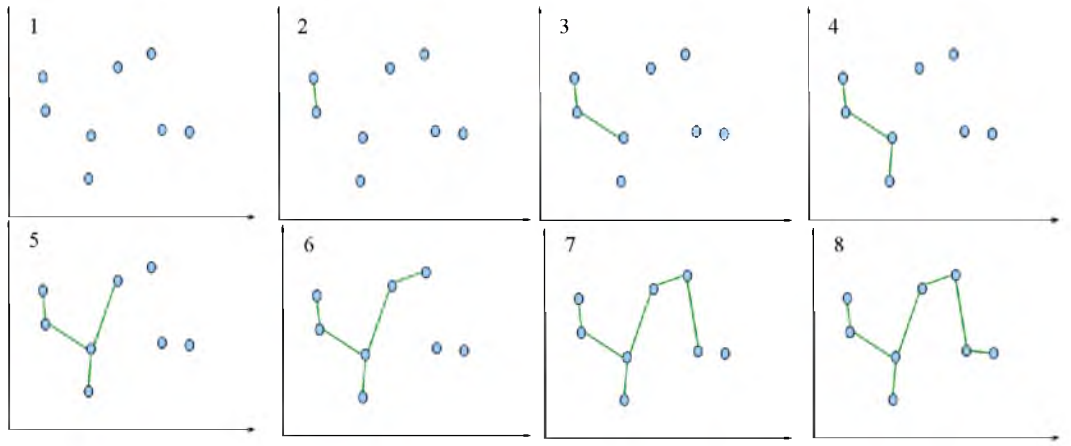

Рис. 1. Иллюстрация работы алгоритма Прима

Алгоритм Крускала изначально помещает каждую вершину в своё дерево, а затем постепенно объединяет эти деревья, объединяя на каждой итерации два ближайших дерева (соединенных ребром с наименьшим весом). Перед началом выполнения алгоритма, все рёбра сортируются по весу (в порядке возрастания). Затем начинается процесс объединения: перебираются все рёбра от первого до последнего (в порядке сортировки), и если у текущего ребра его концы принадлежат разным деревьям, то эти деревья объединяются, а ребро добавляется к ответу. По окончании перебора всех рёбер все вершины окажутся принадлежащими одному поддереву, и ответ найден [2] (рис. 2).

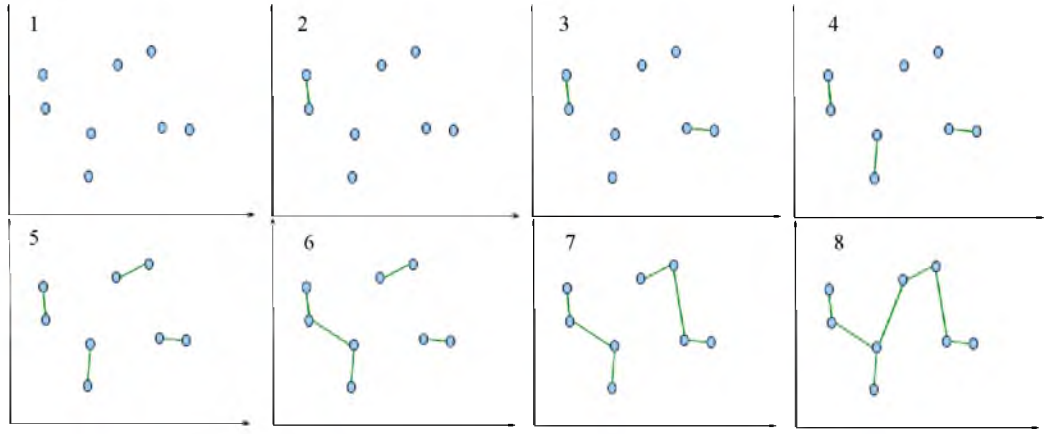

Рис. 2. Иллюстрация работы алгоритма Крускала

Алгоритм Борувки использует подход схожий, с используемым в алгоритме Крускала. На первом этапе каждая вершина помещается в отдельное дерево. На каждой итерации происходит объединение ближайших деревьев до тех пор, пока все объекты не будут принадлежать одному дереву. Отличительной особенностью является тот факт, что на каждой итерации алгоритма количество деревьев сокращается минимум в два раза.

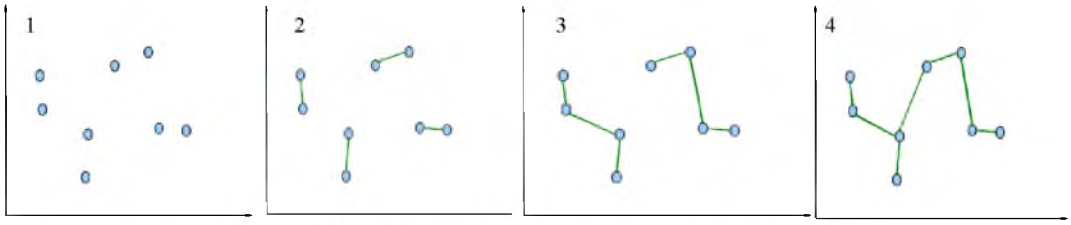

Рис. 3. Иллюстрация работы алгоритма Борувки

Отдельно следует рассмотреть процесс поиска ближайших деревьев для алгоритма Борувки. Пусть на некоторой итерации нужно найти для заданного дерева ближайшее к нему дерево. Для этого необходимо для каждой компоненты исходного дерева найти ближайшую компоненту из другого дерева (Рисунок 4, минимальное ребро выделено штриховкой).

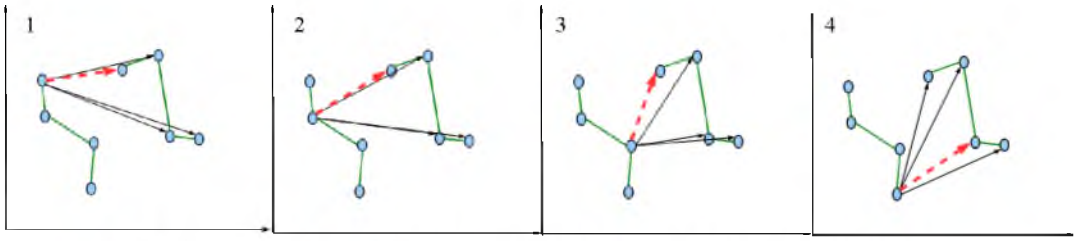

Рис. 4. Иллюстрация поиска ближайшего дерева в алгоритме Борувки

Далее из всех найденных элементов выбрать минимальный, который и станет ребром, объединяющим дерево (рисунок 5).

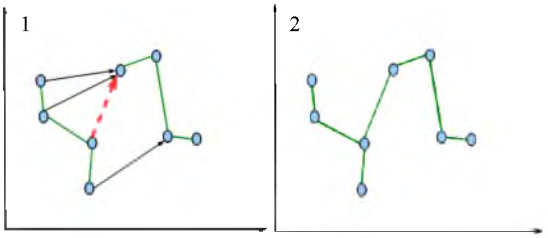

Рис. 5. Иллюстрация поиска ближайшего дерева в алгоритме Борувки

В худшем случае время на операцию поиска ближайшего дерева будет пропорционально V^2, но такой подход обладает очень высокой потенциальным параллелизмом. Поиск ближайшего дерева для каждого элемента исходного дерева можно производить независимо на отдельном вычислительном узле.

Говоря о задаче классификации объектов на изображениях [1] следует учитывать ряд особенностей. В качестве классифицируемых объектов, то есть вершин графа, выступают вектора пиксельных интенсивностей в цветовом пространстве RGB,

где функцией расстояния является евклидова метрика. Поэтому задача построения минимального остовного дерева сводится к задаче поиска евклидова дерева. Один из подходов к ее решению предусматривает построение полного графа с V вершинами, где V - количество объектов, и V(V-1)/2 ребрами - одно ребро соединяет каждую пару вершин, а вес этого ребра равен расстоянию между соответствующими точками. Затем на этом графе производится поиск минимального остовного дерева [3].

Процесс классификации объектов на изображении можно разделить на следующие этапы.

1. Каждый пиксель изображения представляет из себя объект в трехмерном признаковом пространстве RGB. Совокупность всех различных векторов пиксельных интенсивностей, присутствующих на изображении, составляет множество классифицируемых объектов.

2. Строится минимальное евклидово дерево.

3. Ребра дерева разрезаются, в результате чего множество объектов разбивается на непересекающиеся подмножества. Процесс разрезания контролируется в соответствии с поведением функционала качества разбиения [1].

Существенной проблемой при использовании данного метода является ограничение, накладываемое количеством обрабатываемых объектов. На изображении в формате RGB может присутствовать до 2^{24} (порядок 10^8) различных векторов интенсивностей. Процедура поиска минимального евклидова дерева предполагает решение задачи поиска минимального остовного дерева на полном графе, вершинами которого являются классифицируемые объекты, а ребрами - соединения между каждой парой из них. При решении задачи классификации объектов на изображениях количество вершин графа обычно имеет порядок 10^6, а следовательно число ребер графа, на котором будет производится поиск минимального остовного дерева, составит приблизительно $10^{12}/2$. Даже при использовании типов данных одинарной точности, размером 4 байта, для хранения ребер потребуется примерно 1,862Тб памяти, что невозможно. Если вместо хранения весов ребер производить их вычисление, то такой подход приводит к повышению асимптотической сложности алгоритмов. В качестве примера следует рассмотреть модифицированный под данную задачу алгоритм Крускала. Классический алгоритм Крускала включает в себя следующие шаги.

1. Отсортировать все ребра по возрастанию.

2. Начиная с наименьшего ребра, добавлять в граф такие ребра, которые не создают цикла.

Сложность такого алгоритма определяется в первую очередь сложностью сортировки. Т.е. при использовании хороших методов сортировки время работы алгоритма Крускала будет пропорционально O(N log N), где N-количество ребер графа.

Сортировка предполагает хранение весов ребер, что, как было сказано выше, невозможно. Можно переформулировать алгоритм Крускала следующим образом: на каждой итерации находить наименьшее ребро, не создающее цикла, и добавлять его в граф. Поиск такого ребра будет требовать времени пропорционально V^2, таких ребер будет (V-1). Таким образом асимптотическая вычислительная сложность алгоритма будет примерено $O(V^3)$. Кроме того, вычисление евклидовых расстояний между объектами предполагает использование достаточно медленной операции вычисления квадратного корня. В таблице 1 приведена зависимость времени построения евклидового дерева от количества объектов (V). В первой строке таблицы указано количество объектов, со второй по четвертую – время работы алгоритма для различных испытаний (I1,I2,I3), в последней строке указано среднее время работы алгоритма (Avarage). Результаты всех экспериментов получены на процессоре Intel(R) Core(TM)2 Quad CPU Q6600 2.40GHz. Для вычислений использовалось одно ядро, оптимизация компилятором была выключена. Вычисления для каждого набора объектов производились трижды, время усреднялось.

Таблица 1

Зависимость времени работы модифицированного алгоритма Крускала от количества объектов

N	10	100	500	1000	2000	3000
I1	0,00008	0,02928	3,63130	29,00961	232,26556	785,12453
I2	0,00008	0,02923	3,63906	29,04869	232,64933	783,74037
I3	0,00008	0,02923	3,63054	29,14089	232,18983	783,02213
Avarage	0,00008	0,02925	3,63364	29,06639	232,36824	783,96234

Как видно из табл. 1, данный алгоритм является чрезвычайно сложным в вычислительном отношении, и его использование при количестве объектов большем 500 становится затруднительным. Алгоритм Прима при данных условиях (невозможность хранения данных о ребрах) так же будет иметь временную сложность пропорциональную V^3.

Отдельно следует рассмотреть алгоритм Борувки. Его отличительной особенностью этого является то, что на каждой итерации алгоритма количество деревьев сокращается минимум в два раза, а его асимптотическая сложность имеет порядок $O(V^2 \log_2 V)$. Время работы алгоритма Борувки, модифицированного для решения задачи поиска минимального евклидова дерева приведено в табл. 2 (рис. 6).

Таблица 2

Зависимость времени работы модифицированного алгоритма Борувки от количества объектов

N	500	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
I1	0,08	0,32	1,27	2,82	5,95	7,74	13,21	17,92	23,48	29,54	36,44
I2	0,08	0,32	1,26	2,82	5,90	7,75	13,20	17,94	23,44	29,57	36,46
I3	0,08	0,32	1,26	2,82	5,91	7,74	13,20	17,95	23,44	29,58	36,41
Avarage	0,08	0,32	1,26	2,82	5,92	7,74	13,21	17,93	23,45	29,56	36,44

Как видно из табл. 2 и рис. 6, модифицированный алгоритм Борувки существенно быстрее модифицированного алгоритма Крускала. Но все же, он не достаточно быстр для обработки нужного количества объектов.

Отдельно следует рассмотреть процесс поиска ближайших деревьев в алгоритме Борувки. Во-первых, поиск этих деревьев может происходить для каждого дерева независимо, т.е. параллельно. Во-вторых, для того чтобы найти ближайшее дерево, нужно найти ближайший объект из другого дерева для каждого объекта из исходного дерева, что тоже можно делать параллельно. Таким образом, модифицированный алгоритм Борувки обладает очень высоким потенциальным параллелизмом.

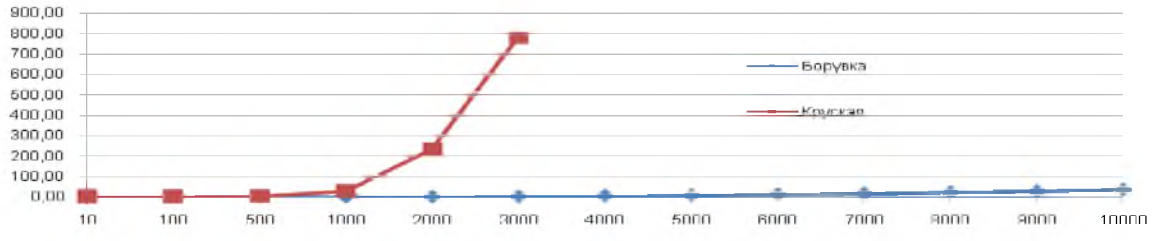

Рис. 6. График зависимости времени выполнения алгоритмов Крускала и Борувки от количества объектов

Для реализации параллельного модифицированного алгоритма Борувки требуется вычислительная система с общей памятью и как можно большим числом вычислительных узлов. Такими системами являются современные графические ускорители. Алгоритм был реализован для графических карт производства Nvidia, с использованием технологии CUDA. Эксперименты проводились на ускорителе Tesla C2050, содержащим 448 вычислительных ядер и 2687 Мб оперативной памяти. Оптимизация компилятором была выключена. Время работы алгоритма на GPU приведено в табл. 3.

Таблица 3

Зависимость времени работы модифицированного алгоритма Борувки на GPU от количества объектов

N	2000	3000	4000	5000	6000	7000	8000	9000	10000
I1	0,05	0,11	0,15	0,24	0,33	0,50	0,59	0,84	0,92
I2	0,05	0,11	0,15	0,24	0,38	0,45	0,66	0,84	0,92
I3	0,05	0,11	0,16	0,24	0,38	0,45	0,59	0,84	0,92
Avarage	0,0	0,1	0,2	0,2	0,4	0,5	0,6	0,8	0,9
N	20000	30000	40000	50000	60000	70000	80000	90000	100000
I1	4,21	11,93	17,36	27,54	39,57	60,19	79,15	100,87	113,20
I2	4,21	11,93	17,35	27,56	39,58	60,20	79,17	91,48	113,18
I3	4,21	11,93	17,37	27,55	39,62	54,52	79,12	100,91	113,16
Avarage	4,2	11,9	17,4	27,5	39,6	58,3	79,1	97,8	113,2

Как видно из табл. 3, построение минимального евклидова дерева средствами GPU позволяет получить хорошие временные показатели. Более того, при применении оптимизации работы с памятью видеокарты можно получить еще более лучшие временные показатели (табл. 4).

Таблица 4

Зависимость времени работы модифицированного алгоритма Борувки на GPU, оптимизированного по памяти, от количества объектов

N	2000	3000	4000	5000	6000	7000	8000	9000	10000
I1	0,02	0,04	0,04	0,05	0,06	0,06	0,08	0,09	0,10
I2	0,02	0,04	0,04	0,05	0,05	0,06	0,07	0,10	0,10
I3	0,02	0,03	0,04	0,06	0,05	0,06	0,08	0,09	0,10
Avarage	0,02	0,03	0,04	0,05	0,06	0,06	0,08	0,09	0,10
N	20000	30000	40000	50000	60000	70000	80000	90000	100000
I1	0,30	0,72	0,87	1,40	2,07	2,42	3,29	4,77	5,41
I2	0,30	0,65	0,87	1,25	2,07	2,42	3,29	4,77	6,00
I3	0,30	0,72	0,87	1,40	2,07	2,42	3,29	4,77	6,00
Avarage	0,30	0,70	0,87	1,35	2,07	2,42	3,29	4,77	5,80
N	200000	300000	400000	800000	1000000				
I1	24,38	48,57	85,14	372,64	525,47				
I2	22,19	48,58	76,69	372,65	630,00				
I3	22,19	48,58	85,14	372,65	525,47				
Avarage	22,92	48,57	82,32	372,65	560,31				

Результаты, представленные в табл. 4, демонстрируют возможность практически моментальной обработки на GPU до 100000 объектов, что является весьма неплохим показателем. На рис. 7 приведен график ускорения алгоритма для GPU при использовании оптимизации по памяти. По оси абсцисс отложено количество объектов, по оси ординат – количество раз, в которое оптимизированной алгоритм быстрее.

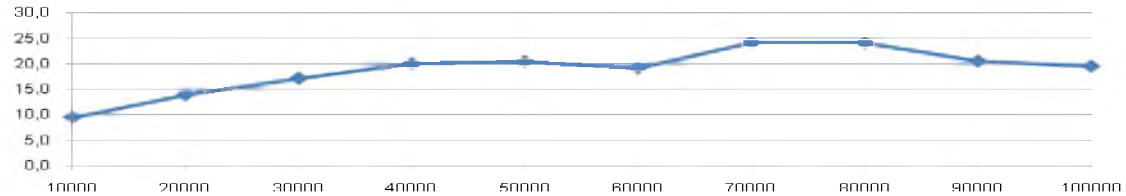

Рис. 7. График зависимости ускорения оптимизированного алгоритма для GPU, по сравнению с неоптимизированным, от количества объектов

Еще более интересным выглядит сравнение модифицированного алгоритма для GPU с алгоритмом для CPU.

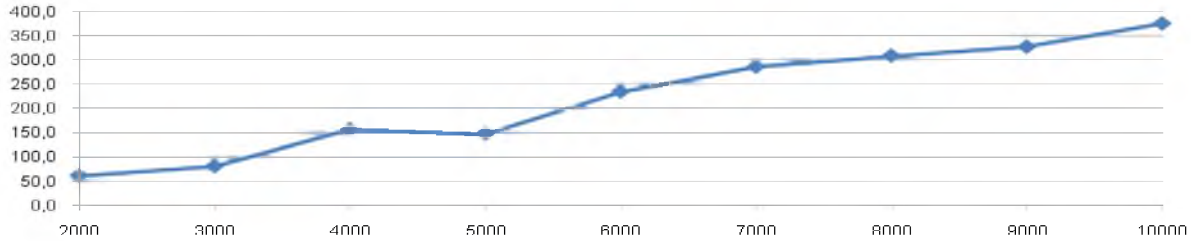

Рис. 8. График зависимости ускорения оптимизированного алгоритма для GPU, по сравнению с алгоритмом для CPU, от количества объектов

Как видно из графиков, оптимизированный алгоритм Борувки для GPU быстрее того же алгоритма для CPU в сотни раз. Сравнение с модифицированным алгоритмом Крускала не проводилось по причине крайне низкой скорости работы последнего. Еще большего ускорения можно достичь за счет учета особенностей задачи поиска минимального евклидова дерева (за счет учета особенностей евклидова пространства).

Работа выполнена при поддержке ФЦП «Научные и научно-педагогические кадры для инновационной России» на 2009-2013 годы, гос. контракт № 14.740.11.0390.

Список литературы

1. Жиляков Е.Г., Барсук А.А. О компьютерной реализации автоматической вариационной классификации объектов на спутниковых фотографиях земной поверхности // Вопросы радиоэлектроники. – 2010. – Выпуск 1. – С. 166-171.

2. Кормен, Томас Х. Алгоритмы: построение и анализ, 2-е издание [Текст]/ Кормен, Томас Х., Лейзерсон, Чарльз И., Ривест, Рональд Л., Штайн, Клиффорд. – М.: Издательский дом "Вильяме", 2005. — 1296 с.

3. Седжвик Роберт Фундаментальные алгоритмы на C++. Алгоритмы на графах: Пер. с англ. – СПб: ООО «ДиаСофтЮП». 2002 – 496 с.

4. Форсайт, Дэвид А., Понс, Жан Компьютерное зрение. Современный подход: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 928с: ил.

THE PARALLEL ALGORITHM OF SEARCHING THE MINIMAL EUCLIDIAN SPANNING TREE IN THE TASK OF OBJECTS CLASSIFICATION AT THE IMAGES for GPU

A.A. BARSUK
O.N. IVANOV

Belgorod National Research University

e-mail: barsuk_alex@mail.ru

The approach to implementation of the fast parallel algorithm of searching the minimal Euclidian spanning tree in the task of object classification at the images is considered in the report. A brief overview of currently used algorithms is produced, recommendations of their modifications are put forward, requirements for the computer system are formulated. The computer experiments with GPU realization of modified Boruvka's algorithm are made.

Key words: minimal spanning tree, Euclidian tree, classification, Prim's algorithm, Kruskal's algorithm , Boruvka's algorithm, GPU, CUDA.