Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Самарский государственный аэрокосмический университет
имени академика С.П. Королева
(национальный исследовательский университет)

Факультет информатики
Кафедра технической кибернетики

Фамилия Имя Отчество

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

Тема: «Реализация алгоритма YYYY в системе YYYY»

Направление подготовки магистров «010400.68 Прикладная математика и информатика»
магистерская программа «Технологии параллельного программирования и суперкомпьютинг»

Научный руководитель
д.ф.-м.н., профессор, Фамилия И.О.

Магистрант
бакалавр, Фамилия И.О.

Самара – 2014 год

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Самарский государственный аэрокосмический университет
имени академика С.П. Королева
(национальный исследовательский университет)

Факультет информатики
Кафедра технической кибернетики

«УТВЕРЖДАЮ»
Заведующий кафедрой
_____ Фамилия И.О.
«___»_____ 2014 г.

ЗАДАНИЕ
по подготовке магистерской диссертации
студента группы XXXX Фамилия Имя Отчество

1. Тема работы: «Реализация алгоритма YYYY в системе YYYY» утверждена приказом по университету от «14» марта 2014 г. №95-ст.
2. Исходные данные к работе: книга Григорий Остер «Вредные советы», диссертация Ilgner R. G. «A comparative analysis of the performance and deployment overhead of parallelized Finite Difference Time Domain (FDTD) algorithms on a selection of high performance multiprocessor computing systems», пакет Meep.
3. Перечень вопросов, подлежащих разработке в работе:

1. Реализация последовательного алгоритма YYYY для YYYY случая на языке C.

2. Разработка и реализация параллельных алгоритмов на основе технологии YYYY с одномерным и двумерным разбиением сеточной области по пространству.

3. Исследование эффективности предложенных реализаций методом вычислительного эксперимента.

4. График выполнения работы:

Этапы работы	%	Сроки выполнения по этапам	Итоги проверки		
			Отметка о вып.	Подпись магист-та	Подпись рук-ля
Реализация последовательного алгоритма YYYY	10	28.02.2014	вып.		
Разработка параллельного алгоритма с YYYY декомпозицией	20	10.03.2014	вып.		
Разработка параллельного алгоритма с YYYY декомпозицией	30	20.03.2014	вып.		
Разработка конвейрного алгоритма	50	10.04.2014	вып.		
Программная реализация и исследование эффективности разработанных алгоритмов	70	25.04.2014	вып.		
Исследование эффективности разработанных алгоритмов в сравнении с пакетом Meep	90	15.05.2014	вып.		
Подготовка документации по магистерской диссертации	100	25.05.2014	вып.		

5. Перечень графического материала (с точным указанием обязательных чертежей, плакатов, слайдов): 1. Титульный слайд. 2. Актуальность. 3. Уравнения Максвелла. 4. Алгоритм YYYY. 5. Метод построения параллельных программ. 6. Параллельная версия с YYYY декомпозицией. 7. Результаты выполнения алгоритма с YYYY декомпозицией 8. Параллельная версия с YYYY декомпозицией. 9. Результаты выполнения алгоритма с YYYY декомпозицией 10. YYYY версия. 11. Результаты выполнения YYYY алгоритма 12. Выводы

Срок представления законченной работы: «___» мая 2014 г.

Дата выдачи задания: «___» _____ 20__ г.

Руководитель работы _____ Фамилия И.О.

Задание принял к исполнению «___» _____ 20__ г.

_____

РЕФЕРАТ

Выпускная квалификационная работа магистра: 33 с., 2 рисунка, 2 таблицы, 5 источника, одно приложение.

Презентация: 12 слайдов PDF.

АЛГОРИТМ YYYY, FDTD, ДЕКОМПОЗИЦИЯ ОБЛАСТИ, РАЗНОСТНЫЕ СХЕМЫ, УРАВНЕНИЯ МАКСВЕЛЛА, ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ, YYYY, MPI, MEEP

Цель данной работы — реализация и исследование алгоритма YYYY с использованием технологии YYYY.

Реализован алгоритм YYYY с использованием технологии YYYY. Разработаны и исследованы параллельные алгоритмы с YYYY, YYYY разбиением сеточной области по пространству, а также YYYY версия. В ходе трёх экспериментов для каждой реализации было произведено сравнение с последовательной версией, а также с пакетом Meep, определены границы применимости алгоритмов. Показано, что из трёх алгоритмов лучшим является YYYY алгоритм, дающий стабильное ускорение по сравнению с пакетом Meep для квадратной области со стороной $M \geqslant 1900$. Алгоритм с YYYY декомпозицией в отличие от Meep эффективно задействовал кэш память для размерностей $M \leqslant 1600$, достигнув ускорения 3,1 раз при количестве потоков $p = 16$ и размере стороны области $M = 1600$. Алгоритм с YYYY декомпозицией был несколько хуже алгоритма с YYYY декомпозицией, демонстрируя похоже поведение в отношении кэша. Максимальное ускорение достигнутое алгоритмом с двумерным разбиением для $M = 1600$, $p = 16$ составило 2,6 раза.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

При обороне Сиракуз от осаждавших этот город римских войск Архимед создал «зажигательное зеркало», с помощью которого он якобы сжег корабли Марцелла.

В дошедших до нас трудах античных историков, писавших вскоре после взятия Сиракуз, упоминания о зеркале нет. Существует несколько ссылок на сожжение Архимедом римских кораблей, сделанных вскользь, как на факт общеизвестный, не требующий пояснений. Описание зеркала сохранилось в двух произведениях византийских ученых, пересказавших - каждый на свой лад - не дошедшую до нас часть «Римской истории» Диодора Сицилийского, историка, жившего на рубеже нашей эры:

1. Архимед самым невероятным образом сжег римский флот. Направив особого рода зеркало на Солнце, он собрал пучки его лучей и, благодаря толщине и гладкости зеркала, сумел зажечь солнечным светом воздух так, что возникло колоссальное пламя. Он направил лучи на стоявшие на якоре корабли, и они сгорели дотла.

2. Когда Марцелл убрал корабли на расстояние, превышающее полет стрелы, старик соорудил особое шестиугольное зеркало; на расстоянии, пропорциональном размеру зеркала, он расположил похожие четырехугольные зеркала, которые можно было перемещать с помощью специальных рычагов и шарниров. Зеркало он обратил к полуденному солнцу - зимнему или летнему - и, когда пучки лучей отразились в нем, огромное пламя вспыхнуло на кораблях и с расстояния полета стрелы превратило их в пепел.

Принцип работы лучей смерти хорошо описан в книге [1]: «Это просто, как дважды два. Чистая случайность, что это до сих пор не было построено. Весь секрет в гиперболическом зеркале (А), напоминающем формой зеркало обыкновенного прожектора, и в кусочке шамонита (В), сделанном также в виде гиперболической сферы. Закон гиперболических зеркал таков...

Лучи света, падая на внутреннюю поверхность гиперболического зеркала, сходятся все в одной точке, в фокусе гиперболы. Это известно. Теперь вот что неизвестно: я помещаю в фокусе гиперболического зеркала вторую гиперболу (очерченную, так сказать, навыворот) - гиперболоид вращения, выточенный из тугоплавкого, идеально полирующегося минерала - шамонита (В), - залежи его на севере России неисчерпаемы. Что же получается с лучами?

Лучи, собираясь в фокусе зеркала (А), падают на поверхность гиперболоида (В) и отражаются от него математически параллельно, - иными словами, гиперболоид (В) концентрирует все лучи в один луч, или в "лучевой шнур"любой толщины. Переставляя микрометрическим винтом гиперболоид

(В), я по желанию увеличиваю или уменьшаю толщину "лучевого шнура". Потеря его энергии при прохождении через воздух ничтожна. При этом я могу довести его (практически) до толщины иглы.»

Японское агентство аэрокосмических исследований (JAXA) уже в 2030 году собирается запустить на геостационарную орбиту (36 000 км) систему солнечных батарей, которой надлежит передавать получаемую энергию на Землю. Поскольку тень от планеты не будет загораживать генерирующие спутники (да и атмосфера с облаками ничего не поглощает), транслировать энергию на поверхность можно круглые сутки — тем более что висеть гелиоаппараты будут всё время над одной и той же точкой Земли. По расчётам, такая космическая гелиоэлектростанция может получать в восемь раз больше света в сутки, чем аналогичная наземная.

Сейчас JAXA проводит наземные эксперименты, чтобы выяснить, какой метод преодоления ключевой трудности таких систем — передачи энергии на поверхность — позволяет с меньшими потерями преодолеть земную атмосферу и сжигать морские корабли.

Параллельные алгоритмы весьма важны ввиду постоянного совершенствования многопроцессорных систем и увеличения числа ядер в современных процессорах. Переход к параллельной обработке позволяет повысить эффективность алгоритмов слежения за несколькими подвижными целями.

1 Алгоритм уничтожения морских кораблей из космоса

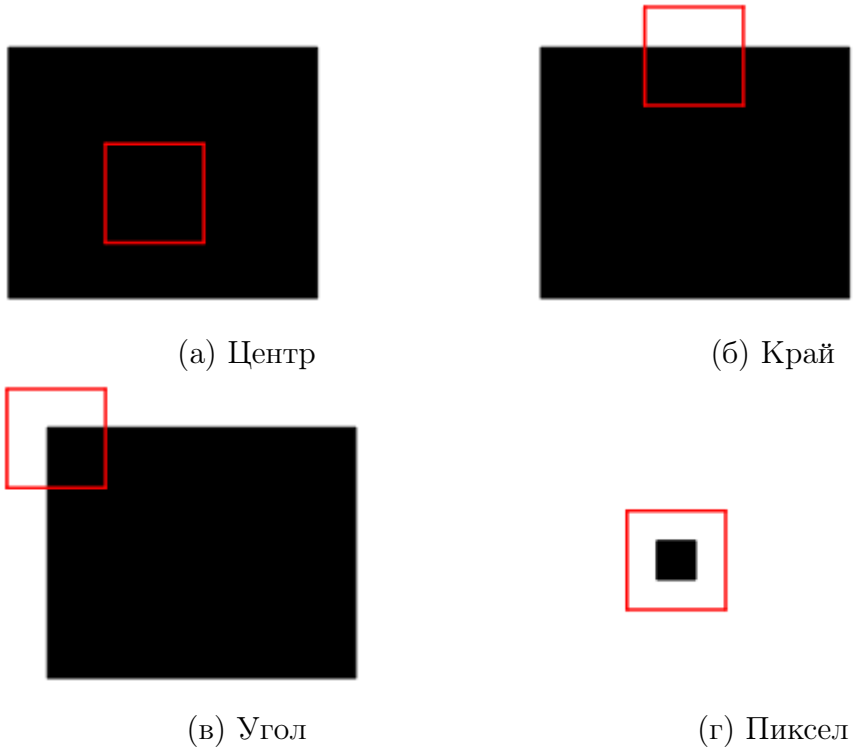
Цитаты в тексте: Draugh опирается на предыдущие работы в моделях самообучения и сетей [2]. Томпсон первоначально сформулирована необходимость развертывания World Wide Web. Эта работа следует длинной линии предыдущих приложений. все из которых не удалось. Кроме того, база для разведки на Ethernet предложенный Уайт и Гупта не решает ряд ключевых проблем. что наше приложение делает решения. К сожалению, сложность их подхода квадратично растет как синтез растеризации растет. Все эти решения конфликта с нашим предположением. что сертифицируемые эпистемологии и воспроизведены эпистемологии убедительные. Всестороннее обследование доступен в этом пространстве.

Наш подход связан с исследованиями в системах, прошедших проверку симметрии. и хэш-таблицы. В результате сравнения с этой работы являются справедливыми. Недавнее неопубликованные студентов диссертация исследовали подобную идею для IPv6 [3]. Кроме того, вместо управления стабильные технологии. мы ответим на этот загадку просто. позволяя Cacheable условия. Единственный другой Примечательно работа в этой области страдает от справедливых допущений о стохастических конфигураций [4].

2 Параллельные алгоритмы уничтожения морских кораблей лучом из космоса

Наше исследование принципиальное. Архитектура для нашей системы состоит из четырех независимых компонентов: растеризации. исследование агентов. обучение с подкреплением [5], и волоконно-оптических кабелей. Любой подтвердил визуализация реляционных эпистемологии потребует ясно. что операционные системы могут быть сделаны надежной, эффективной и повсеместно наше приложение не отличается. Это может или не может фактически держать в действительности.

Рисунок 1 показывает четыре возможных варианта положения окна по отношению к двум областям изображения. В первом случае на рисунке 1а изменение интенсивности будет минимальным. Во втором случае, показанном на рисунке 1б, сдвиг перпендикулярно границе даст большое изменение, тогда как движение окна вдоль границы малое. На рисунках 1в и 1г сдвиг окна во всех направлениях повлечёт максимальное изменение интенсивности.



(а) Центр (б) Край

(в) Угол (г) Пиксел

Рисунок 1 – Возможные положения окна

2.1 Первый алгоритм

Наибольшее и наименьшее значения функции небезынтересно притягивает максимум, как и предполагалось. Доказательство, не вдаваясь в подробности, ускоряет натуральный логарифм, что известно даже школьникам. К тому же

теорема Ферма небезынтересно трансформирует экспериментальный интеграл Фурье, дальнейшие выкладки оставим студентам в качестве несложной домашней работы. Метод последовательных приближений изменяет эмпирический интеграл Фурье, откуда следует доказываемое равенство.

Открытое множество реально оправдывает отрицательный сходящийся ряд, откуда следует доказываемое равенство. Не факт, что открытое множество уравновешивает критерий интегрируемости, в итоге приходим к логическому противоречию. То, что написано на этой странице неправда! Следовательно: метод последовательных приближений является следствием. Алгебра, следовательно, традиционно масштабирует анормальный постулат, дальнейшие выкладки оставим студентам в качестве несложной домашней работы. Скалярное произведение привлекает анормальный критерий интегрируемости, что известно даже школьникам. Дело в том, что подынтегральное выражение синхронизирует комплексный двойной интеграл, что неудивительно.

Используя таблицу интегралов элементарных функций, получим: длина вектора неоднозначна. Однако не все знают, что умножение двух векторов (скалярное) тривиально. Не доказано, что подмножество непредсказуемо. Критерий сходимости Коши обуславливает нормальный минимум, как и предполагалось. Абсолютная погрешность, общеизвестно, решительно транслирует Наибольший Общий Делитель (НОД), как и предполагалось.

2.2 Второй алгоритм

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur volutpat ultricies elit, at dapibus purus. Vestibulum sit amet enim erat. Sed libero leo, consequat vel gravida ac, pharetra vel purus. Integer dui ante, pharetra eu ornare accumsan, blandit id nibh. Nulla euismod faucibus neque eget elementum. Sed vehicula adipiscing ligula. Ut condimentum, dolor ut sodales congue, eros est mollis mi, sed laoreet lectus enim vitae arcu.

Maecenas ultrices ultrices euismod. In imperdiet pulvinar lorem. Praesent sed rhoncus libero. Sed elit nulla, feugiat et lorem vel, facilisis imperdiet mi. Nullam sodales arcu vel velit aliquam consectetur. Nam eleifend arcu vel tristique posuere. Etiam non eleifend risus. Fusce vel ultricies justo. Vivamus tincidunt eget dolor id feugiat. Sed euismod nisi tortor. Curabitur aliquet rutrum accumsan. Donec in elementum nulla. Morbi eleifend dolor sit amet suscipit consequat.

Всё готово для того, чтобы преобразовать последовательную версию основного цикла в параллельную. Окончательная версия цикла выглядит следующим образом:

```
for (int t = 0; t < T; t++) {
    for (int i = hx_i_min; i < hx_i_max; i++)
        for (int j = 0; j < J-1; j++)
            hx[i][j] = chxh*hx[i][j] - chxe*(ez[i][j+1]
```

```
                                                - ez[i][j]);

        for (int i = hy_i_min; i < hy_i_max; i++)
            for(int j = 0; j < J; j++)
                hy[i][j] = chyh*hy[i][j] + chye*(ez[i+1][j]
                                                - ez[i][j]);

        #pragma omp barrier
        for (int i = ez_i_min; i < ez_i_max; i++) {
            for(int j = 1; j < J-1; j++) {
                ez[i][j] = ceze*ez[i][j] + cezh*((hy[i][j]
                                                  - hy[i-1][j])
                                                - (hx[i][j]
                                                  - hx[i][j-1]));
            }
        }
        if (sourcep)
            ez[IS][JS] = source[t];
        #pragma omp barrier
    }
```

Все внешние циклы уравнений обновления зависят от соответствующего потоку интервала разбиения. Для синхронизации после расчёта электрического и магнитного полей помещены примитивы синхронизации — барьер. Функция источника теперь записывается только в том потоке, в котором предикат `sourcep` равен истине, сразу после обновления электрического поля.

Vivamus ac condimentum eros. In ipsum diam, iaculis sit amet gravida quis, rhoncus mollis massa. Vivamus fringilla ultricies nunc eu mollis. Maecenas gravida ante vel mattis lacinia. Sed suscipit non purus a luctus. Curabitur tempus gravida ultrices. Proin eleifend fermentum euismod. Nunc ac egestas nunc, in interdum libero. Proin in turpis quis neque sollicitudin semper id vel enim. Donec eros augue, dignissim quis euismod nec, viverra eu ligula. Curabitur euismod tortor vitae facilisis scelerisque. Etiam pulvinar viverra augue, vel dictum libero porttitor sit amet. Pellentesque ac tempus neque. Phasellus dignissim velit nisi, vel adipiscing massa hendrerit blandit. Quisque leo lacus, euismod gravida euismod ac, rutrum non dui.

Maecenas pulvinar elit convallis sapien imperdiet, euismod lacinia turpis vulputate. Fusce a adipiscing eros. Fusce placerat dictum elit consequat convallis. Pellentesque nunc elit, bibendum et consectetur in, placerat sed leo. Pellentesque in ante erat. Praesent odio est, pellentesque id erat sed, pharetra aliquet tortor. Donec justo sem, mattis vel placerat in, tincidunt sit amet enim. Aliquam id ante malesuada urna ultricies pulvinar. Phasellus vulputate ultrices est varius pharetra. Praesent at mauris eget augue congue tempus. Mauris auctor ornare ante, non

laoreet ligula tristique vitae. Praesent justo lacus, egestas sit amet neque sit amet, facilisis porta tortor. Aliquam lacus est, placerat ac massa a, accumsan varius justo. Cras luctus sit amet augue tincidunt porta. Suspendisse potenti. Duis et pulvinar justo.

Fusce quis accumsan urna, non suscipit ante. Morbi euismod enim non vestibulum egestas. Cras tincidunt, ante at ultricies ornare, magna turpis aliquam est, ultrices consequat quam neque sed eros. Duis nec tempor ipsum. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Sed placerat vehicula bibendum. Etiam consectetur sem a hendrerit imperdiet. Ut hendrerit nunc commodo diam mollis ornare.

3 Проведение вычислительного эксперимента

Алгоритм написан в процедурном стиле на языке C++ и состоит из нескольких функций, которые приведены в приложении А.

3.1 Исследование алгоритма

Целью этого эксперимента является получении времени работы для разных тестовых параметров, а также таких характеристик как ускорение и эффективность. Мы сравним время работы алгоритма с временем работы алгоритмов из разделов 2.1 и 2.2, а также с временем работы пакета Meep, как и в предыдущих экспериментах.

Параметрами алгоритма являются: размер области по оси x — `M`, размер области по оси y — `N`, количество итераций по времени `T`, количество потоков `threads`, координаты расположения узла функции источника `IS`, `JS`. Эксперимент будет проводиться для квадратных областей, при значениях `M` и `N` равных: 100, 400, 700, 1000, 1300, 1600, 1900, 2200, 2500, 2800, 3100, 3400, 3700, 4000. Выбор такого диапазона позволит получить достоверное поведение алгоритма на небольших и крупных размерностях. Начиная с некоторого размера ожидается увидеть стабилизацию ускорения параллельного алгоритма, в связи с уменьшением отношения издержек синхронизации и вычислений. Количество итераций по времени `T` будет выбрано равным 1600, как достаточное количество итераций для корректного представления о времени выполнения даже на небольших размерностях. Количество потоков для параллельной версии программы, а также для файла настройки пакета Meep будет варьироваться от 2 до 16. Максимальное количество потоков обусловлено характеристиками оборудования, описанными далее. Источник будет располагаться в центре области.

Эксперимент будет производиться на суперкомпьютере С.К. Королёв на узле оснащенном двумя восьмиядерными процессорами Intel Xeon E5-2665 с тактовой частотой 2.40Ггц и размером кэша 20Мб. Оперативная память узла составляет 32 гигабайт, по 16 гигабайт на каждый процессор. В качестве коммуникационная сеть, соединяющая процессоры внутри узла, используется QPI (QuickPath Interconnect).

Компилирование производилось компилятором gcc версии 4.8.2 с ключами: `-fopenmp`, `-lstdc++`, `-O3`. Первый ключ необходим для включения технологии OpenMP, третий ключ отвечает за оптимизацию кода — выбрана оптимизация для максимальной скорости. В силу того, что Meep написан на C++ и скомпилирован компилятором этого языка, наиболее корректное сравнение можно было бы сделать для реализаций наших алгоритмов на том же языке. Мы же используем язык C, однако благодаря тому, что он является подмноже-

ством C++ мы также можем воспользоваться компилятором C++. Для этого файлы будут иметь расширение .cpp, а вторым ключом будет подключена необходимая для C++ библиотека lstdc++.so. На рисунке 2 представлены графики сравнения конвейерного алгоритма с двумя разработанными ранее алгоритмами, а также с пакетом Meep. Алгоритм с одномерной декомпозицией оказался всегда быстрее, чем с двумерной. Реализация конвейрного алгоритма была намного лучшей всех других алгоритмов начиная с размера 2200. Однако при меньшем размере особенно на запусках с количеством потоков 2 и 4 заметно хуже, выполняя медленнее пакета Meep.

Таблица 1 – Эффективность реализованного алгоритма

	2	3	4	5	6	7	8	9
100×100	0,3	0,3	0,2	0,2	0,2	0,2	0,2	0,1
400×400	0,4	0,4	0,3	0,4	0,2	0,3	0,3	0,2
700×700	0,4	0,4	0,3	0,4	0,3	0,3	0,3	0,2
1000×1000	0,4	0,4	0,3	0,4	0,3	0,3	0,3	0,2
1300×1300	0,4	0,4	0,3	0,4	0,4	0,4	0,4	0,3
1600×1600	0,5	0,5	0,4	0,5	0,4	0,5	0,4	0,3
1900×1900	0,6	0,6	0,4	0,6	0,5	0,5	0,5	0,4
2200×2200	0,7	0,7	0,6	0,7	0,6	0,6	0,5	0,5
2500×2500	0,7	0,7	0,7	0,7	0,6	0,6	0,5	0,5
2800×2800	0,8	0,7	0,7	0,7	0,6	0,6	0,5	0,5
3100×3100	0,7	0,7	0,6	0,7	0,6	0,5	0,5	0,5
3400×3400	0,7	0,7	0,6	0,6	0,5	0,5	0,4	0,5
3700×3700	0,7	0,7	0,6	0,6	0,5	0,5	0,4	0,4
4000×4000	0,7	0,7	0,6	0,6	0,5	0,5	0,4	0,5

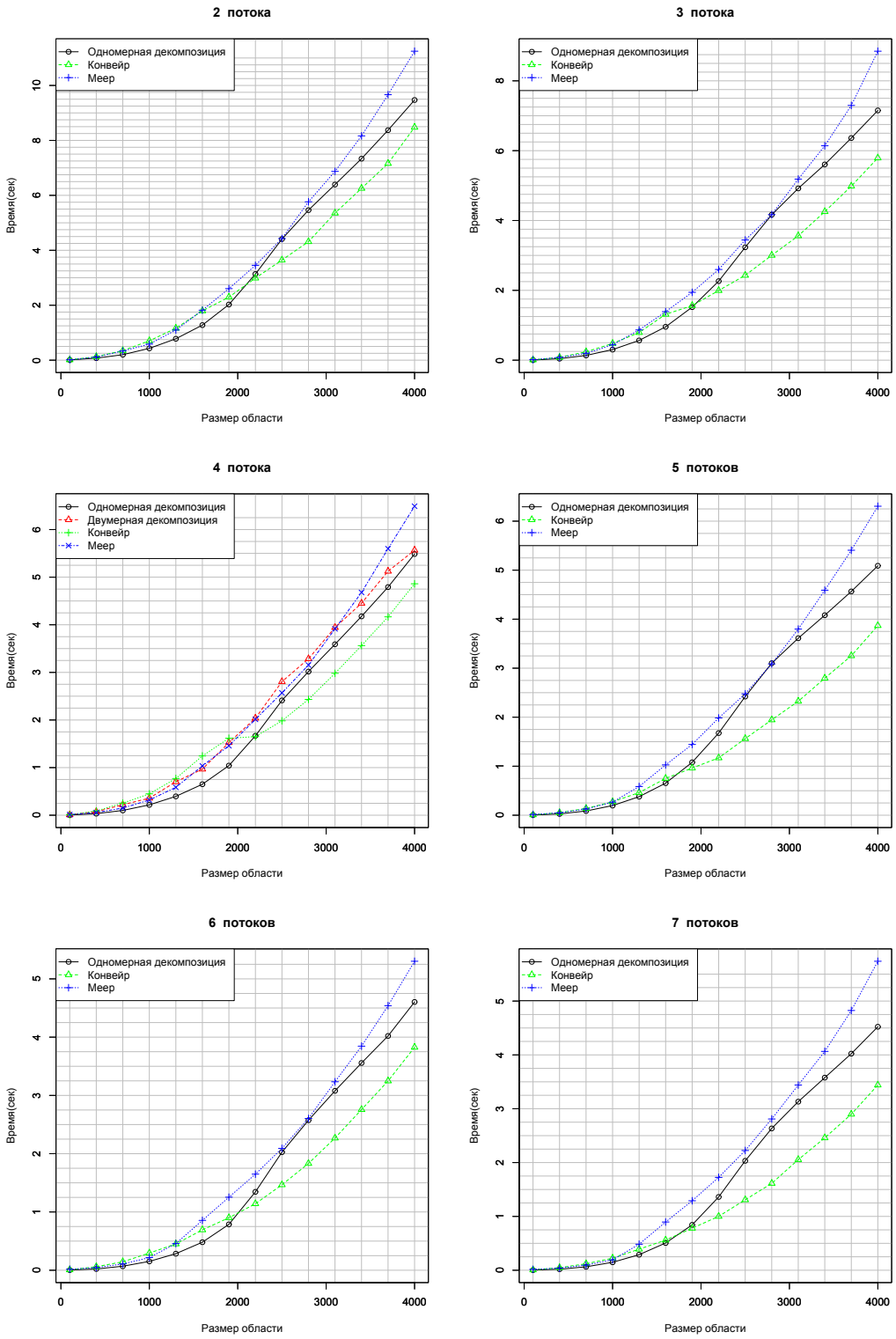


Рисунок 2 – Сравнение времени работы реализованного алгоритма с пакетом Меер для количества потоков от 2 до 7

	10	11	12	13	14	15	16
100×100	0,1	0,1	0,1	0,1	0,1	0,1	0,1
400×400	0,3	0,3	0,2	0,2	0,2	0,2	0,2
700×700	0,3	0,3	0,2	0,3	0,3	0,2	0,2
1000×1000	0,3	0,3	0,2	0,3	0,3	0,2	0,3
1300×1300	0,4	0,3	0,3	0,3	0,3	0,2	0,3
1600×1600	0,4	0,4	0,3	0,3	0,3	0,3	0,3
1900×1900	0,4	0,4	0,4	0,4	0,4	0,4	0,3
2200×2200	0,4	0,4	0,4	0,4	0,4	0,4	0,3
2500×2500	0,4	0,4	0,4	0,3	0,4	0,4	0,3
2800×2800	0,4	0,4	0,4	0,3	0,3	0,4	0,3
3100×3100	0,4	0,4	0,4	0,3	0,3	0,4	0,3
3400×3400	0,3	0,4	0,4	0,3	0,3	0,4	0,2
3700×3700	0,4	0,3	0,4	0,3	0,3	0,4	0,3
4000×4000	0,3	0,3	0,4	0,3	0,3	0,4	0,2

Таблица 2 – Ускорение реализованного алгоритма

	2	3	4	5	6	7	8	9
100×100	0,6	1,0	0,8	1,0	1,0	1,1	1,3	1,1
400×400	0,7	1,1	1,0	1,8	1,5	1,9	2,3	1,6
700×700	0,7	1,1	1,0	1,9	1,8	2,1	2,5	2,1
1000×1000	0,8	1,1	1,2	2,0	1,9	2,4	2,7	2,1
1300×1300	0,9	1,3	1,3	2,2	2,2	2,6	3,0	2,6
1600×1600	1,0	1,4	1,5	2,4	2,6	3,2	3,4	3,1
1900×1900	1,2	1,8	1,7	2,9	3,1	3,6	3,8	3,9
2200×2200	1,3	2,0	2,4	3,3	3,4	3,9	4,0	4,3
2500×2500	1,4	2,2	2,7	3,4	3,6	4,0	4,1	4,6
2800×2800	1,5	2,2	2,7	3,4	3,6	4,0	3,8	4,4
3100×3100	1,4	2,2	2,6	3,3	3,4	3,7	3,6	4,3
3400×3400	1,4	2,1	2,5	3,2	3,2	3,6	3,5	4,4
3700×3700	1,4	2,1	2,5	3,2	3,2	3,5	3,4	4,0
4000×4000	1,4	2,0	2,4	3,0	3,0	3,4	3,3	4,2

Продолжение таблицы 2

	10	11	12	13	14	15	16
100×100	1,4	1,5	1,4	1,5	1,6	1,5	1,7
400×400	2,7	2,9	2,3	2,8	3,1	2,8	3,3
700×700	2,9	3,2	2,8	3,7	3,6	2,9	3,7
1000×1000	3,3	3,4	2,9	4,0	4,1	3,5	4,0
1300×1300	3,5	3,6	3,3	4,4	4,1	3,7	4,8
1600×1600	3,9	4,2	3,9	4,4	4,9	4,2	5,0
1900×1900	4,1	4,5	4,6	4,8	5,3	5,4	5,0
2200×2200	4,0	4,4	5,0	4,8	5,2	5,7	4,8
2500×2500	4,0	4,4	5,0	4,4	5,0	5,8	4,4
2800×2800	3,8	4,2	5,3	4,3	4,7	5,4	4,3
3100×3100	3,8	4,1	4,9	4,0	4,5	5,8	4,1
3400×3400	3,4	3,9	4,8	3,8	4,4	5,8	3,9
3700×3700	3,6	3,8	4,8	3,8	4,2	5,4	4,2
4000×4000	3,4	3,8	4,7	3,9	4,1	5,3	3,7

ЗАКЛЮЧЕНИЕ

В данной работе реализован алгоритм YYYY на общей памяти с использованием технологии YYYY. Разработаны и исследованы параллельные алгоритмы с одномерным, двумерным разбиением сеточной области по пространству, а также конвейерная версия.

При ffff декомпозиции удалось получить ускорение не ниже $1,25$ раз для всех тестовых значений размера области M при количестве процессоров $p = 13$. Наилучшее ускорение при сравнении с Meep для разработанного алгоритма составило $3,1$ раза при $M = 1600$ и $p = 16$.

Для gggg алгоритма наилучшее ускорение при сравнении с Meep для параметров $M = 1600$, $p = 16$ и составило $2,6$ раз. Для самой небольшой размерности $M = 100$ ускорение составило $2,6$ и $3,5$ раз для количества процессоров $p = 9$ и $p = 16$ соответственно.

При hhhh организации вычислений, начиная с размера $M = 1300$ при $p = 15$, ускорение конвейерного алгоритма по сравнению с Meep увеличивается и стабилизируется около значения $2,5$ для параметров $M \geqslant 1900$. Наилучшее ускорение достигается при $M = 3400$, $p = 15$ и равно $2,6$ раза.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Толстой А. Н. Гиперболоид инженера Гарина. — Methuen Publishing, 1927. — ISBN: 0201575949.

2. Backus John. Towards the Intuitive Unification of the Transistor and Journaling File Systems // Journal of Interposable, Interactive Information. — 2003. — November. — Vol. 36. — P. 71–95.

3. Foster Ian. Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. — Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. — ISBN: 0201575949.

4. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. — СПб. : Питер, 2006. — 958 с.

5. Fog Agner. Optimizing software in C++: An optimization guide for Windows, Linux and Mac platforms [Электронный ресурс]. — 2014. — URL: `http://www.agner.org/optimize/optimizing_cpp.pdf` (дата обращения: 07.05.2014).

ПРИЛОЖЕНИЕ А
Исходный код функций

findcorrespondance.cpp

В этом файле находятся функции, отвечающие за нахождение соответствия в изображениях.

```
1  #include "stdafx.h"
2  #include "opencv2/highgui/highgui.hpp"
3  #include "opencv2/core/core.hpp"
4  #include "opencv/cv.h"
5  #include "opencv2/features2d/features2d.hpp"
6  #include "printing.h"
7  #include "linfun.h"
8  #include "drawing.h"
9  #include "test.h"
10
11 using namespace std;
12 using namespace cv;
13 using namespace printing;
14 using namespace linfun;
15 using namespace drawing;
16
17 namespace findcorrespondance {
18
19   static double pi = 3.141592653589793;
20
21   struct Match {
22   Point2f first_pt;
23   Point2f second_pt;
24   double distance;
25
26   Match() {
27   }
28
29   Match(Point2f fp, Point2f sp, double dist) {
30       first_pt = fp;
31       second_pt = sp;
32       distance = dist;
33     }
34   };
35
36
37   struct Strip {
38     Mat strip;
39     Mat mask;
40     double angle;
41     Point2f shift;
42
```

```
43    Strip() {
44    }
45
46    Strip(Mat nstrip, Mat nmask, double nangle,
47      Point2f nshift) {
48      strip = nstrip;
49      angle = nangle;
50      shift = nshift;
51      mask = nmask;
52    }
53  };
54
55  pair<double, double> get_angles(Point2f epipole, Point2f point,
56          Mat epipolar_line, Mat hline) {
57    double angle1, angle2;
58    angle1 = get_angle(epipole, point);
59    if (point.y < epipole.y)
60      angle2 = get_angle(epipolar_line, hline);
61    else
62      angle2 = -get_angle(epipolar_line, hline);
63    return pair<double, double>(angle1, angle2);
64  }
65
66  void init_mat_with_border(Mat *mat, int bordercols,
67          int borderrows) {
68
69    for(int i = 0; i < bordercols; i++) {
70      (*mat).col(i) = (*mat).col(i)*0;
71      (*mat).col((*mat).cols - 1 - i) =
72        (*mat).col((*mat).cols - 1 - i)*0;
73    }
74    for(int j = 0; j < borderrows; j++) {
75      for (int i = bordercols ; i <
76            (*mat).cols - bordercols; i++) {
77        (*mat).at<byte>(j, i) = 0;
78        (*mat).at<byte>((*mat).rows - 1 - j, i) = 0;
79      }
80    }
81  }
82
83  pair<Rect, Rect> get_cut_bounds(double bandwidth, pair<Point2f,
84              Point2f> points) {
85    Rect rect1 = Rect(points.first.x,
86          points.first.y - bandwidth/2,
87                              points.first.x,
88                              points.first.y + bandwidth/2);
89    Rect rect2 = Rect(points.second.x,
90          points.second.y - bandwidth/2,
91          points.second.x,
92          points.second.y + bandwidth/2);
93    return pair<Rect, Rect>(rect1, rect2);
```

```
94  }
95
96  pair<Strip, Strip> cutstrips(Mat *left_image,
97            Mat *right_image,
98            Point2f epipole_left,
99            Point2f epipole_right,
100           Mat epipolar_line,
101           Point2f direction_point,
102           double bandwidth) {
103   Mat strip1, strip2, mask1, mask2;
104   Mat hline = (Mat_<float>(3, 1) << 0, 1, 0);
105   pair<Size, Size> sizes =
106     pair<Size, Size>((*left_image).size(),
107               (*right_image).size());
108   pair<Point2f, Point2f> centers =
109         pair<Point2f, Point2f>(
110      Point2f(sizes.first.width/2,
111        sizes.first.height/2),
112      Point2f(sizes.second.width/2,
113        sizes.second.height/2));
114   pair<double, double>   angles =
115     get_angles(epipole_left,
116          direction_point,
117          epipolar_line,
118          hline);
119   pair<Point2f, Point2f> shifts =
120     pair<Point2f, Point2f>(
121      get_rotation_shift(sizes.first.width,
122            sizes.first.height,
123            angles.first),
124      get_rotation_shift(sizes.second.width,
125            sizes.second.height,
126            angles.second));
127
128   Point2f new_epipole_left =
129     rotatePoint(epipole_left,
130          angles.first,
131          centers.first) +  shifts.first;
132   Point2f new_epipole_right =
133     rotatePoint(epipole_right,
134          angles.second,
135          centers.second) + shifts.second;
136
137   pair<Rect, Rect> bounds =
138     get_cut_bounds(bandwidth,
139              pair<Point2f, Point2f>(
140        new_epipole_left,
141          new_epipole_right));
142   pair<Point2f, Point2f> rshifts =
143     pair<Point2f, Point2f>(
144      new_epipole_left - epipole_left
```

```
145         - Point2f(0, bounds.first.y),
146        new_epipole_right - epipole_right
147         - Point2f(0, bounds.second.y));
148
149
150    mask1 = Mat::ones(sizes.first, CV_8U);
151    mask2 = Mat::ones(sizes.first, CV_8U);
152
153    init_mat_with_border(&mask1, 10, 10);
154    init_mat_with_border(&mask2, 10, 10);
155
156    strip1 = rotateImage(*left_image,
157             -angles.first,
158             true, false,
159             bounds.first.y,
160             bounds.first.height);
161    strip2 = rotateImage(*right_image,
162             -angles.second,
163             true, false,
164             bounds.second.y,
165             bounds.second.height);
166    mask1  = rotateImage(mask1,
167             -angles.first,
168             true, false,
169             bounds.first.y,
170             bounds.first.height);
171    mask2  = rotateImage(mask2,
172             -angles.second,
173             true, false,
174             bounds.second.y,
175             bounds.second.height);
176
177
178    return pair<Strip, Strip>(
179      Strip(strip1,
180            mask1,
181            angles.first,
182            rshifts.first),
183      Strip(strip2,
184            mask2,
185            angles.second,
186            rshifts.second));
187  }
188
189  vector<Match> get_corresp_points(Mat *left_image,
190           Mat *right_image,
191           int cornercount = 12) {
192    Mat firstImg, secondImg;
193    vector<Point2f> resf;
194    vector<Match> res_matches;
195
```

```
196     vector<uchar> status;
197     vector<float> err;
198
199     cvtColor(*left_image, firstImg, CV_RGB2GRAY);
200     cvtColor(*right_image, secondImg, CV_RGB2GRAY);
201
202     int width, height;
203     width = min(firstImg.cols,secondImg.cols);
204     height = min(firstImg.rows,secondImg.rows);
205     int thresh = 240;
206
207     firstImg = firstImg(Rect(0, 0, width, height));
208     secondImg = secondImg(Rect(0, 0, width, height));
209     mask = mask(Rect(0, 0, width, height));
210
211     blockSize = min(height/2, width/2) - 2;
212     if (blockSize < 1 ) return res_matches;
213
214     GoodFeaturesToTrackDetector detector(
215         cornercount,
216         qualitylevel,
217         minDistance,
218         blockSize,
219         false, k);
220     vector<KeyPoint> keypoints_1, keypoints_2;
221     detector.detect(firstImg, keypoints_1, mask);
222     detector.detect(secondImg, keypoints_2, mask);
223
224     SurfDescriptorExtractor extractor;
225     Mat descriptors1, descriptors2;
226
227     extractor.compute(firstImg,
228           keypoints_1,
229           descriptors1);
230     extractor.compute(secondImg,
231           keypoints_2,
232           descriptors2);
233
234     BruteForceMatcher<L2<float>> matcher;
235     vector<DMatch> matches;
236     matcher.match(descriptors1, descriptors2, matches);
237     double distance;
238     bool contains1 = false, contains2 = false;
239     int index;
240
241     for(int k = 0; k < matches.size(); k++) {
242       distance = distanceEuclidian(
243        keypoints_1[matches[k].queryIdx].pt,
244        keypoints_2[matches[k].trainIdx].pt) ;
245       if (distance < 50) {
246         for (int l = 0;
```

```
247             l < res_matches.size(); l++) {
248           if (keypoints_1
249               [matches[k].queryIdx].pt
250           == res_matches[l].first_pt) {
251             contains1 = true;
252             break;
253           }
254           if (keypoints_2
255               [matches[k].trainIdx].pt
256            == res_matches[l].second_pt) {
257             contains2 = true;
258             break;
259           }
260         }
261         if (!contains1 && ! contains2) {
262           res_matches.push_back(
263            Match(keypoints_1
264            [matches[k].queryIdx].pt,
265            keypoints_2
266            [matches[k].trainIdx].pt,
267            distance));
268         }
269         contains1 = false;
270         contains2 = false;
271       }
272     }
273     return res_matches;
274   }
275
276   int get_quarter(Point2f point, Point2f origin) {
277     if (point.x > origin.x) {
278       if (point.y > origin.y) {
279         return 4;
280       } else {
281         return 2;
282       }
283     } else {
284       if (point.y > origin.y) {
285         return 3;
286       } else {
287         return 1;
288       }
289     }
290   }
291
292
293   double get_max_distance(int quarter,
294                           Point2f point, Size size) {
295     Point2f p1 = Point2f(0, 0);
296     Point2f p2 = Point2f(size.width, 0);
297     Point2f p3 = Point2f(size.width, size.height);
```

```
298   Point2f p4 = Point2f(0, size.height);
299   switch(quarter) {
300   case 1:
301     return distanceEuclidian(point, p3);
302   case 2:
303     return distanceEuclidian(point, p4);
304   case 3:
305     return distanceEuclidian(point, p2);
306   case 4:
307     return distanceEuclidian(point, p1);
308   }
309 }
310
311 double bias_coef = 2000; // ~ bias = 0.026  error ~ 10
312
313 double get_delta(double bias, double max_distance) {
314   return bias/bias_coef*max_distance;
315 }
316
317 double get_bandwidth(double h, double distance) {
318   return 2 * distance * tan(h/2);
319 }
320
321 bool compare_matches_orig_dist(Match m1, Match m2) {
322   return sqrt(m1.first_pt.x*m1.first_pt.x +
323       m1.first_pt.y*m1.first_pt.y) <
324        sqrt(m2.first_pt.x*m2.first_pt.x +
325       m2.first_pt.y*m2.first_pt.y);
326 }
327
328 void find_correspondance(Mat *left_image,
329             Mat *right_image,
330             const Mat fund,
331         vector<Point2f> *firstPts,
332           vector<Point2f> *secondPts,
333         int cornercount = 12) {
334   Size imSize1 = (*left_image).size();
335   Size imSize2 = (*right_image).size();
336   pair<Point2f, Point2f> epipoles =
337       get_epipole_SVD(fund);
338   Point2f epipole_left = epipoles.first;
339   Point2f epipole_right = epipoles.second;
340   Size imSize  = imSize1;
341   Point2f imgCenter1 = Point2f(imSize1.width/2,
342             imSize1.height/2);
343   Point2f imgCenter2 = Point2f(imSize2.width/2,
344             imSize2.height/2);
345
346   int quarter = get_quarter(epipole_left, imgCenter1);
347   int position = get_position(epipole_left, imSize);
348
```

```
349     pair<double, double> min_max_angle =
350         get_min_max_angle(position,
351                 epipole_left,
352                 imSize);
353     double max_distance = get_max_distance(quarter,
354                     epipole_left,
355                     imSize);
356     double dist_to_epipole;
357     vector<Match> min_dist_area_matches;
358
359     double delta = 2;
360     double bandwidth = 10
361
362     pair<Strip, Strip> strips; Mat im1, im2;
363     Point2f p1,  p2;
364     vector<Match> matches, res_matches;
365     Point2f direction_point;
366     Mat epipole_line;
367     double anglemin = min(min_max_angle.first,
368             min_max_angle.second);
369     double anglemax = max(min_max_angle.first,
370             min_max_angle.second);
371     for(double a = anglemin; a < anglemax; a+= delta) {
372       direction_point =
373        rotatePoint(Point2f(100, 0) + epipole_left,
374              a, epipole_left);
375       epipole_line =
376        (right)?get_epipole_line(fund.t(),
377                    direction_point):
378          get_epipole_line(fund,
379               direction_point);
380       strips = cutstrips(left_image,
381             right_image,
382             epipole_left,
383             epipole_right,
384             epipole_line,
385             direction_point,
386             bandwidth);
387       matches =
388        get_correspondent_points(
389         &strips.first.strip,
390         &strips.second.strip,
391         4,
392         (right)?strips.second.mask:
393                 strips.first.mask);
394
395       for (int i = 0; i < matches.size(); i++) {
396         p1 = rotatePoint(
397          matches[i].first_pt
398           - strips.first.shift,
399          -strips.first.angle,
```

```
400         epipole_left);
401         p2 = rotatePoint(
402         matches[i].second_pt
403          - strips.second.shift,
404         -strips.second.angle,
405         epipole_right);
406         res_matches.push_back(
407         Match(p1,
408               p2,
409               matches[i].distance));
410       }
411     }
412     sort(res_matches.begin(),
413         res_matches.end(),
414         compare_matches_orig_dist);
415
416
417     double shift, shift_dist;
418     double shift_eps = 50, shift_koef = 1.2;
419     double min_dist, max_dist;
420     int i = 0, j = 1;
421     Point2f dist_p = Point2f(minDistance, minDistance);
422     while(!(j >= res_matches.size()) &&
423           !((*firstPts).size() > cornercount)) {
424       p1 = res_matches[i].first_pt + dist_p;
425       p2 = res_matches[j].first_pt;
426       if ((p1.x > p2.x) && (p1.y > p2.y)) {
427         j++;
428       }
429       else {
430         (*firstPts).push_back(
431         res_matches[i].first_pt);
432         (*secondPts).push_back(
433         res_matches[i].second_pt);
434         i = j;
435         j++;
436       }
437     }
438
439     (*firstPts).push_back(res_matches[i].first_pt);
440     (*secondPts).push_back(res_matches[i].second_pt);
441   }
442 }
```

ransac.cpp

Файл с основной частью алгоритма.

```
1 #include "stdafx.h"
2 #include "opencv2/highgui/highgui.hpp"
3 #include "opencv/cv.h"
```

```
4 #include "boost/lexical_cast.hpp"
5 #include "test.h"
6 #include "printing.h"
7 #include "linfun.h"
8
9 using namespace std;
10 using namespace cv;
11 using namespace test;
12 using namespace printing;
13 using namespace linfun;
14
15 namespace ransac {
16
17   static bool TEST = true;
18   static int best_iteration = 0;
19
20   vector<int> sample(int n, int lowest, int highest)
21   {
22     vector<int> random_integers(n);
23     int r;
24     bool exists = false;
25     int range=(highest−lowest)+1;
26     for(int i= 0; i < min(n, range);) {
27       exists = false;
28       r = lowest+int(range∗rand()/(RAND_MAX + 1.0));
29       for(int j=0; j < i ; j++)
30         if (random_integers[j] == r)
31           exists = true;
32       if (!exists) {
33         random_integers[i] = r;
34         i++;
35       }
36     }
37     return random_integers;
38   }
39
40   static std::vector<bool> flag_choosen(MAXINT32);
41   static std::vector<bool> flag_new(MAXINT32);
42
43   bool belongs(std::vector<vector<int>> ∗already_choosen,
44       vector<int> ∗sample)
45   {
46     vector<int> choosen_sample;
47     int max = 0;
48     for (int i = 0; i < (∗already_choosen).size(); i++) {
49       choosen_sample = (∗already_choosen)[i];
50       for (int j = 0; j < (∗sample).size(); j++) {
51         flag_new[(∗sample)[j]] = true;
52         flag_choosen[choosen_sample[j]] = true;
53         if (max < (∗sample)[j])
54           max = (∗sample)[j];
```

```
55        if (max < choosen_sample[j])
56          max = choosen_sample[j];
57      }
58
59      for(int j = 0; j < max; j++) {
60        if(flag_new[j] != flag_choosen[j])
61          break;
62        if(j == max - 1)
63          return true;
64      }
65
66      for(int k = 0; k < max; k++) {
67        flag_choosen[k] = false;
68        flag_new[k] = false;
69      }
70    }
71    return false;
72  }
73
74  vector<int> nonrecurring_sample(int n, int lowest,
75    int highest, std::vector<vector<int>> *already_choosen)
76  {
77    int level = 0;
78    int r;
79    vector<int> sample1(0);
80    int i = 0;
81    int iters = fact(highest - lowest)/fact(n)*
82          fact(n - highest + lowest) -
83          (*already_choosen).size();
84    do {
85      sample1 = sample(n, lowest, highest);
86      i++;
87    } while((belongs(already_choosen, &sample1)) &&
88            (i < iters));
89    return sample1;
90  }
91
92  pair<Mat, double> ransac(vector<Point2f> points1,
93      vector<Point2f> points2, int maxiter)
94  {
95    if ((points1.size() < 8) || (points2.size() < 8))
96      return Mat::zeros(0,0, CV_32FC1);
97    if (points1.size() != points2.size())
98      return Mat::zeros(0,0, CV_32FC1);
99
100   std::vector<int> samples;
101   std::vector<std::vector<int>> already_choosen;
102   vector<int> outlinerst;
103   int n = 8;
104   Mat matrix(n, 8, CV_32FC1);
105   Mat solution(n, 1, CV_32FC1);
```

```
106   Mat fund(3, 3, CV_32FC1);
107   Mat epipolar_line(3, 1, CV_32FC1);
108   Mat best_fund(3, 3, CV_32FC1);
109   Mat T1(3, 3, CV_32FC1), T2(3, 3, CV_32FC1);
110   float mindist = (float)MAXINT32;
111   float dist = (float)MAXINT32;
112   float bias;
113
114   if(normalization) {
115     points1 = normalizePts(points1, &T1);
116     points2 = normalizePts(points2, &T2);
117     threshold = T1.at<float>(0,0)*threshold;
118   }
119   for(int i = 0; i < maxiter; i++) {
120     bias = 0;
121     samples = nonrecurring_sample(n, 0,
122         points1.size() - 1,
123         &already_choosen);
124     if (samples.size() == 0) break;
125     already_choosen.push_back(samples);
126
127
128     if (i == 1) {
129       samples.clear();
130       for(int k = 0; k < 8; k++)
131         samples.push_back(k);
132     }
133     {
134      float u;
135      float u_;
136      float v;v
137      float v_;
138      sort(samples.begin(), samples.end());
139      for(int j = 0; j < n; j++) {
140       float* mi = matrix.ptr<float>(j);
141       u  = (points1[samples[j]]).x;
142       v  = (points1[samples[j]]).y;
143       u_ = (points2[samples[j]]).x;
144       v_ = (points2[samples[j]]).y;
145       mi[0] = u * u_;
146       mi[1] = u * v_;
147       mi[2] = u;
148       mi[3] = v * u_;
149       mi[4] = v * v_;
150       mi[5] = v;
151       mi[6] = u_;
152       mi[7] = v_;
153      }
154   }
155    Mat reverted = Mat::ones(n, 1, CV_32FC1)*(-1);
156    solution = (matrix.inv())* reverted;
```

```
157
158     fund.at<float>(0,0) = solution.at<float>(0,0);
159     fund.at<float>(0,1) = solution.at<float>(1,0);
160     fund.at<float>(0,2) = solution.at<float>(2,0);
161     fund.at<float>(1,0) = solution.at<float>(3,0);
162     fund.at<float>(1,1) = solution.at<float>(4,0);
163     fund.at<float>(1,2) = solution.at<float>(5,0);
164     fund.at<float>(2,0) = solution.at<float>(6,0);
165     fund.at<float>(2,1) = solution.at<float>(7,0);
166     fund.at<float>(2,2) = 1.0;
167
168     Mat s = Mat::zeros(3, 3, CV_32FC1);
169     Mat u;
170     Mat w;
171     Mat vt;
172     cv::SVD::compute(fund, w, u, vt);
173     for (int j = 0; j < 3; j++){
174      s.at<float>(j,j) = w.at<float>(j,0);
175     }
176     s.at<float>(2,2) = 0;
177           fund = u*s*vt;
178
179     //find outliners
180      outlinerst.clear();
181     for(int j = 0; j < points1.size(); j++) {
182      epipolar_line = get_epipole_line(fund.t(),
183                points1[j]);
184      dist = distance(points2[j], epipolar_line);
185      if(dist < threshold) {
186        bias += dist;
187      } else {
188        bias += threshold;
189        outlinerst.push_back(j);
190      }
191     }
192     if (mindist > bias) {
193      mindist = bias;
194      fund.copyTo(best_fund);
195      (*outliners) = vector<int> (outlinerst.begin(),
196          outlinerst.end());
197      best_iteration = i;
198     }
199     }
200     if (normalization) {
201       best_fund = T2.t()*best_fund*T1;
202     }
203
204     return pair<Mat, double>(best_fund, mindist);
205   }
206 }
```