ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 1(1), 2010, с. 3–20

УДК 517.977

Ю. Л. Сачков, А. А. Ардентов, А. П. Маштаков

Параллельный алгоритм и программа восстановления изофот для поврежденных изображений

Аннотация. Описан опыт распараллеливания решения задачи восстановления кривых на изображениях с помощью вариационного подхода. Приведены показатели эффективности разработанной C++ программы в библиотеке параллельного программирования TSim.

Ключевые слова и фразы: оптимальное управление, обработка изображений, параллельные вычисления.

Введение

Задача восстановления поврежденных или скрытых изображений является одной из актуальных проблем компьютерной графики, реставрации фотографии, фильмов и живописи. Для решения этой задачи разработан ряд методов, многие из которых основаны на современных математических методах, в частности, на применении вариационного исчисления и оптимального управления [1, 2]. Данная работа основана на положениях одного из новых направлений нейрофизиологии зрения — нейрогеометрии [3, 4], а также на недавних результатах по субримановой геометрии [5]. На основе результатов этих исследований разработан комплекс параллельных программ в системе TSim [6] для восстановления серии изофот на поврежденных штриховых и полутоновых изображений.

Работа выполнена в рамках научно–технической программы Союзного государства «СКИФ-ГРИД», а также проекта Российского Фонда Фундаментальных Исследований No. 09-01-00246.

1. Восстановление изображений: Нейрогеометрия зрения и оптимальное управление

1.1. Нейрогеометрия зрения

Важным открытием нейрофизиологии зрения начала нынешнего века является геометрическая структура, соответствующая первичной зрительной коре головного мозга человека. Первичная зрительная кора осуществляет первичное (предшествующее всякой обработке) восприятие зрительной информации мозгом человека. Установлено, что для сохранения изображений первичная кора моделирует контактную структуру $\{(x, y, p)\} = D \times \mathbb{R}P^1$ на поверхности сетчатки глаза $D \subset \mathbb{R}^2$. Здесь касательный элемент p соответствует наклону кривой $y(x)$ в данной точке. Оказалось, что для эффективной обработки изображений мозгу человека выгоднее сохранять контур не в виде набора последовательных точек (x_i, y_i), а в виде набора штрихов (x_i, y_i, p_i), в пределе — в виде непрерывной кривой $(x(t), y(t), p(t))$, $p = dy/dx$. Если часть кривой повреждена или скрыта от наблюдения, то недостающая дуга восстанавливается на основе следующего вариационного принципа: восстанавливаемая дуга должна иметь минимальную евклидову длину в пространстве (x, y, θ), $\theta = \operatorname{arctg} p$:

$$\int \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{\theta}^2}\, dt \to \min .$$

Этот принцип положен далее в основу метода восстановления скрытой дуги. Описанная внутренняя геометрия зрительной коры является одним из основных объектов исследования нейрогеометрии зрения — направления нейрофизиологии зрения, изучающего геометрические структуры человеческого мозга, моделирующие пространственные образы внешнего мира.

1.2. Постановка задачи восстановления изображения и метод решения

Рассматривается задача восстановления черно-белого (штрихового или полутонового) изображения, некоторые фрагменты которого испорчены или скрыты от наблюдения. Требуется восстановить испорченные фрагменты антропоморфным (естественным для человека) способом. Математически задача может быть формализована

следующим образом. Дана прямоугольная область $D \subset \mathbb{R}^2$, взаимно непересекающиеся подобласти $O_1, ..., O_N \subset D$, и гладкая функция $f : D \backslash (\bigcup_{i=1}^{N} O_i) \to [0, 1]$. Требуется восстановить функцию f в областях $O_1, ..., O_N$. Здесь D — область исходного изображения, O_i — подобласти с испорченным изображением; функция f задает доступное наблюдателю изображение (для полутонового изображения яркость, а для штрихового изображения это функция, линии уровня которой совпадают с кривыми, составляющими изображение). Предлагается восстановление изображения в подобластях O_i на основе дополнения изофот — линий уровня функции f в этих подобластях (области между восстановленными кривыми в случае полутонового изображения раскрашиваются согласно значениям яркости на этих кривых). Кривые вычисляются на основе вариационного подхода: построенная линия $(x(t), y(t))$ должна минимизировать расстояние в пространстве (x, y, θ), где (x, y) — координаты на плоскости $\mathbb{R}^2$, а $\theta(t) = \operatorname{arctg} p(t) = \operatorname{arctg}(\dot{y}/\dot{x})$ — угол наклона касательной к кривой $(x(t), y(t))$. Алгоритм решения соответствующей задачи оптимального управления основан на результатах работы [5] и реализован в библиотеке параллельного программирования TSim [6]. Предлагаемый метод восстановления изофот может использоваться в сочетании с другими методами.

1.3. Восстановление кривой на основе вариационного подхода

Рассмотрим гладкую плоскую кривую $AB = \{(x(t), y(t)) \ | t \in [a, b]\}$. Предположим, что часть этой кривой $CD = \{(x(t), y(t)) | t \in [c, d]\}$ скрыта от наблюдения или повреждена. Для восстановления кривой CD, построим касательную T_C к кривой AC в точке C и касательную T_D в точке D, см. Рис. 1. Обозначим через θ_c, θ_d углы наклона касательных T_C, T_D. Искомая кривая $\overline{CD} = \{(\overline{x}(t), \overline{y}(t)) | t \in [c, d]\}$ должна выходить из точки C с углом наклона θ_c, приходить в точку D с углом наклона θ_d, и иметь кратчайшую длину в пространстве (x, y, θ):

$$\int_c^d \sqrt{(dx/dt)^2 + (dy/dt)^2 + (d\theta/dt)^2} dt \to \min .$$

Граничные условия означают гладкое сопряжение новой кривой CD с известными участками AC и DB исходной кривой. Исходная и новая кривая изображены на Рис. 2. Условие минимума формализует

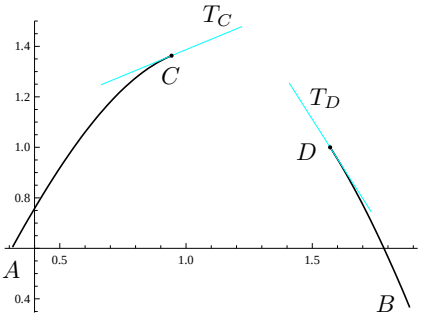

Рис. 1. Граничные условия для восстановления дуги CD

условие естественности новой кривой CD: при ее поиске штрафуются большие отклонения как по координатам (x, y), так и по углу наклона θ. Таким образом, минимизируется некоторый интегральный компромисс между линейной и угловой скоростями движения кривой.

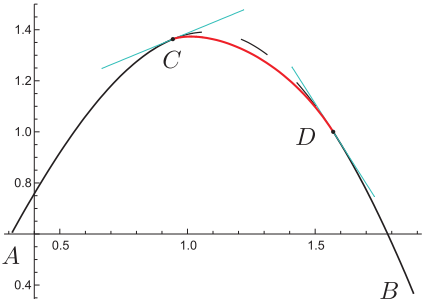

Рис. 2. Исходная кривая AB с поврежденной и новой дугами CD

Описанная задача формализуется как следующая задача оптимального управления:

$$\dot{x} = \pm u\cos\theta, \quad \dot{y} = \pm u\sin\theta, \quad \dot{\theta} = v,$$
$$(x, y) \in \mathbb{R}^2, \quad \theta \in [0, \pi], \quad (u, v) \in \mathbb{R}^2,$$
$$(x(c), y(c)) = C, \quad \theta(c) = \theta_c, \quad (x(d), y(d)) = D, \quad \theta(d) = \theta_d,$$
$$\int_c^d \sqrt{u^2 + v^2}dt \to \min.$$

Далее эта задача называется задачей для пар штрихов (в точках C и D). В работе [5] эта задача была сведена к решению систем алгебраических уравнений в эллиптических функциях. Описанный способ восстановления кривой обладает важным свойством инвариантности относительно движений плоскости: при поворотах и параллельных переносах плоскости решения задачи оптимального управления переходят в решения, поэтому форма восстанавливаемой кривой не зависит от положения и ориентации изображения на плоскости.

2. Программный комплекс OptimalInpainting

На языке C++, с использованием библиотеки TSim, создан программный комплекс OptimalInpainting («Оптимальное Восстановление Изображений»), моделирующий работу с бинарными и полутоновыми изображениями:

(1) создание изображения по аналитическим данным, определенным пользователем,
(2) порождение повреждений с параметрами, определяемыми пользователем,
(3) восстановление изофот на изображениях в последовательном или параллельном режимах.

Пример исходного, поврежденного и восстановленного изображения приведен на Рис. 3, 4.

Структура программного комплекса OptimalInpainting изображена на Рис. 5.

Основным модулем комплекса является программа GlobalSolver, вычисляющая восстанавливающую изофоту изображения как решение задачи для пар штрихов, описанной в п. 1.3. Основная цель данной работы — описание алгоритма распараллеливания этой программы, и демонстрация эффективности этого алгоритма.

3. Параллельные алгоритм и программа GlobalSolver решения серии задач для пар штрихов

3.1. Гранулы параллелизма

Для эффективного распараллеливания необходимо определить такие гранулы параллелизма, которые имеют достаточно большую вычислительную сложность, но при этом их количества должно быть

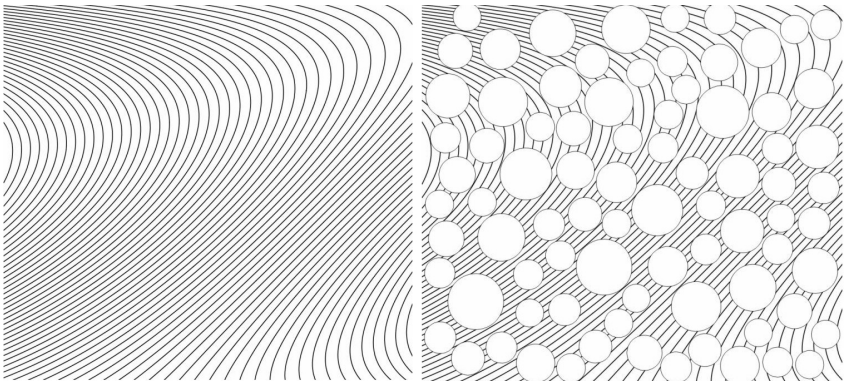

Рис. 3. Исходное и поврежденное изображение

Рис. 4. Восстановленное изображение

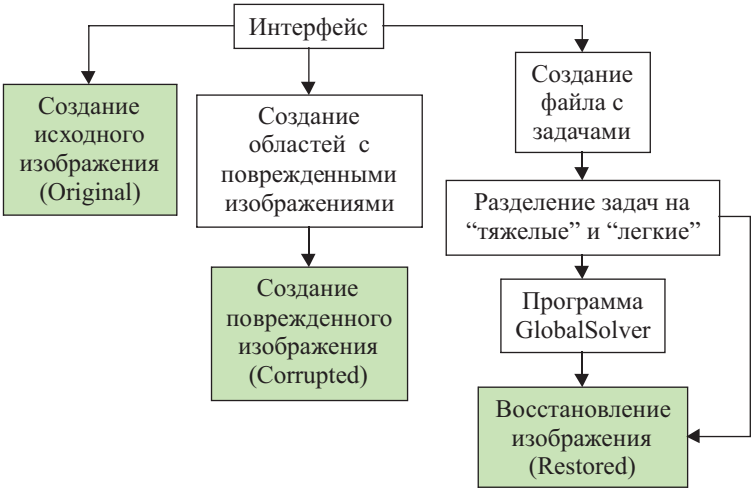

Рис. 5. Структура программного комплекса OptimalInpainting

достаточно для равномерного распределения по узлам. В последовательной реализации среднее время решения задачи для пар штрихов примерно равно 1/3 секунды, (время посчитано при тестировании на

узле кластера blade.botik.ru). Но при этом существуют задачи, которые могут решаться более часа. Общее время вычисления серии задач не может быть меньше времени вычисления одной такой задачи. Единственный способ ускорить вычисление — решать такую задачу на нескольких узлах. Но так как невозможно определить, будет ли та или иная задача считаться дольше среднего времени, то при этом придется вычислять все задачи таким образом. Поэтому необходимо найти компромисс между легкими и тяжелыми задачами.

В качестве гранулы параллелизма принимается решение одной задачи для пары штрихов. Заметим, что размер такой гранулы можно искусственно увеличивать за счет склеивания нескольких задач в одну гранулу. Но такой подход оказывается неоправданным, так как существует вероятность склеивания нескольких тяжелых задач в одну, тем более для его реализации необходимо отдельно заботиться о передаче большого числа входных параметров для T-функции.

3.2. Работа с библиотекой T-Sim

Для распараллеливания использовалась библиотека параллельного вычисления T-Sim (http://wiki.botik.ru/TSim/) для языка C++. Название происходит от сокращения «T-Simplified» — «упрощённая» реализация подхода T-системы, среды автоматического и динамического распараллеливания вычислений: бесконфликтная модель вычислений на основе чистых (не имеющих побочных эффектов) функций и «неготовых значений» как средства синхронизации доступа к результатам вычислений (при попытке доступа к данным поток–потребитель приостанавливается до тех пор, пока значение не будет вычислено потоком–производителем). Основой T-Sim послужили идеи этого подхода, при этом, в отличие от T++, T-Sim позволяет обойтись без введения новых ключевых слов для выражения понятий Tсистемы. При этом TSim характеризуется сравнительно легкой переносимостью, так, например, для нее не нужен компилятор языка, требуется лишь компилятор C++. Также имеется возможность быстро конструировать стратегии выравнивания нагрузки, повторно использовать уже существующие.

Алгоритм распараллеливания в библиотеке TSim основан на использовании неготовых значений в сочетании с Tфункциями.

3.2.1. *Пример параллельной программы на основе библиотеки TSim*

Ниже приведен пример кода параллельной программы на языке C++ с использованием библиотеки T-Sim: функция funVec применяется к списку значений, которые пробегает итератор i, результат записывается в вектор T-переменных, которые потом выводятся в файл «output_vector.txt».

```
#include <vector>
#include <fstream>
#include "tsim.h" // подключение библиотеки T-Sim
using namespace std;

typedef TVal<double> TDouble; // тип неготового значения
TSIM_TFUNDEF_2 (funVec,double,TDouble,TFunVec)
// объявление  T-функции с двумя переменными;
// обычный вызов происходит с помощью funVec;
// вызов в качестве T-функции производится с помощью TFunVec.

void funVec (double d, TDouble td) { // описание T-функции
        td = d; // выполняемые операции
        return;
}

int main() {
        TSimRuntime rt; // инициализация библиотеки T-Sim
        fstream out("output_vector.txt", ios::out);
        // выходной файл с результатами вычислений

        int n = 10; // количество элементов,
                    //для которых вычисляется T-функция
        vector <TDouble> a; // вектор неготовых значений
        for (int i=0; i<n; ++i) {
                TDouble ai; // создание неготового значения
                TFunVec((double)i,ai); // вызов T-функции
                a.push_back(ai); // добавление временной переменной
                                 // в вектор неготовых значений
        }
        for (int i=0; i<n; ++i) out <<a[i] <<endl;
        // ожидаем результата вычисления и выводим его в файл

        while (!a.empty()) {
                TDouble& p = a.back(); // создаем ссылку на
                                       //последний элемент
                if(p.isReady()) {
                        p.release(); // удаляем занимаемую
```

```
                                        // TDouble память
                                a.pop_back(); // удаляем соответствующий
                                        //элемент вектора
                        }
                }
                out.close();
                return 0;
}
```

В данном примере был использован тип double для создания неготовых значений. Для параметризации можно использовать любой тип, имеющий конструктор по умолчанию и поддерживающий сериализацию. Сериализация необходима для обеспечения передачи объектов по сети.

3.2.2. *Типы переменных, используемые для создания неготовых значений*

Для создания неготовых значений в программе решения серии задач для пар штрихов использовался тип bool, для которого пришлось определить специализацию шаблонного класса Serializer в файле ≪tsim_serialize.h≫:

```
template <typename TF>
class Serializer<bool,TF> {
 public:
        Serializer(bool& t,TF& tf) {
                tf&t;
        }
};
```

3.2.3. *Планировщик RoundRobin*

При использовании стандартного планировщика библиотеки T-Sim для решения серии задач авторами работы наблюдалась необоснованная неравномерность распределения гранул между узлами, поэтому для задачи данного проекта использовался другой планировщик, RoundRobin [7]. Это алгоритм распределения нагрузки распределённой вычислительной системы методом перебора её элементов по циклу.

Пусть имеется N узлов, способных выполнить заданное действие, и M задач, которые должны быть выполнены этими узлами. Подразумевается, что узлы равны по своим свойствам между собой, а задачи имеют равный приоритет. Тогда планировщик RoundRobin назначает первую задачу для выполнения первому узлу, вторую —

второму и т. д., до достижения последнего узла. После этого следующая задача будет назначена снова первому узлу и т. п. Иными словами, происходит перебор выполняющих задания узлов по циклу, или по кругу (round), и по достижении последнего узла следующая задача будет опять назначена первому узлу. Решение задач может быть дополнительно разбито на кванты времени, причем для продолжения решения во времени нумерация узлов (и, соответственно, назначенные задачи) сдвигается по кругу на 1, то есть задача первого узла отдается второму, второго — третьему, и т. д., а первый узел получает задачу последнего, либо освобождается для приема новой задачи. Таким образом, алгоритм RoundRobin является алгоритмом распределения времени или балансировки нагрузки.

3.3. Порядок обработки каждой гранулы: отправление на вычисление, получение значения готовых переменных

При запуске в T-Sim программе T-функции, для ее выполнения будет создан отдельный поток (нить, тред), T-Sim использует для этих целей ставшую компонентой glibc библиотеку NPTL. Эта библиотека позволяет создавать ограниченное количество потоков, не больше 500. Если запустить сначала на вычисление максимально возможное количество вызовов T-функций, и затем вызывать следующие вычисления по мере упорядоченной готовности T-переменных, то может возникнуть ситуация, при которой все запущенные задачи посчитаются быстрее, чем первая, результат которой ожидает программа. После этого все узлы кроме одного будут простаивать, что существенно скажется на эффективности распараллеливания. Поэтому для улучшения эффективности распараллеливания необходим специальный инструмент для задания порядка запуска T-функций.

Опишем сначала переменные, которые будут использоваться далее:

```
const int tsk = 100; // количество одновременно запущенных задач
// любая константа меньше 500
int runt; // текущее количество запущенных задач
int complt; // количество посчитанных, но не обработанных задач
int i; // индекс текущей задачи
vector <bool> ready; // вектор, показывающий, какие
                     // задачи считаются готовыми
int j; // индекс для подсчета готовых задач
int lastt; // первая задача среди незапущенных на вычисление
```

Функция Ready показывает готовность входной переменной. Функции in, out выполняют ввод входных параметров и соответственно вывод результатов в соответствующие файлы. TFindar — вызов T-функции, которая производит вычисление задающей оптимальную кривую тройки {u, v, k}=$\lambda[i]$ по тройке граничных условий {x, y, th}=$q[i]$.

Опишем схему, проиллюстрированную на Рис. 6. Сначала на вычисление рассылаются tsk задач, после чего количество запущенных задач runt равно tsk. Если первая среди запущенных задача к тому времени посчиталась ($Ready(\lambda[i]) == true$, где $\lambda[i]$ — результат i-ой задачи), то результат вычисления выводится в файл (функция out) и посылается очередная задача на вычисление (T-функция TFindar). Если же первая задача еще не посчиталась, то все запущенные задачи проверяются на готовность (функция Ready) и вычисляется количество задач, которые стали готовыми с момента последней проверки. Таким образом, уменьшается значение переменной runt, и на счет запускаются задачи до тех пор, пока runt не будет равно tsk. Процесс повторяется до тех пор, пока все задачи не посчитаются.

Описанная схема была применена в программе GlobalSolver для параллельного решения серии задач для пар штрихов.

3.4. Отсеивание тяжелых гранул (задач для пар штрихов, решение которых близко к прямой линии)

При тестировании выяснилось, что решение всех «тяжелых» задач — кривая, визуально похожая на прямую линию.

На основе этого наблюдения был определен критерий, с помощью которого можно отсеять большинство «тяжелых» задач: если $(|y/x| < \varepsilon_1) \wedge (|\theta(mod\ \pi)| < \varepsilon_2)$, то решением этой задачи считается прямая линия. Константы ε_1 и ε_2 определяются таким образом, чтобы прямая, которая принимается в качестве решения, была визуально похожа на кривую, которую она заменяет. Пример такой тяжелой задачи приведен на Рис. 7 (непосредственное решение этой задачи занимает примерно 137 секунд). На основе этой задачи в текущей версии ПК OptimalInpainting были выбраны значения параметров $\varepsilon_1 = 0.009$, $\varepsilon_2 = 0.028$.

Чем больше значение этих констант, тем больше задач отсеиваются, поэтому необходимо задать такие ε_1 и ε_2, чтобы при этом отсеивались все «тяжелые» и наименьшее количество «легких».

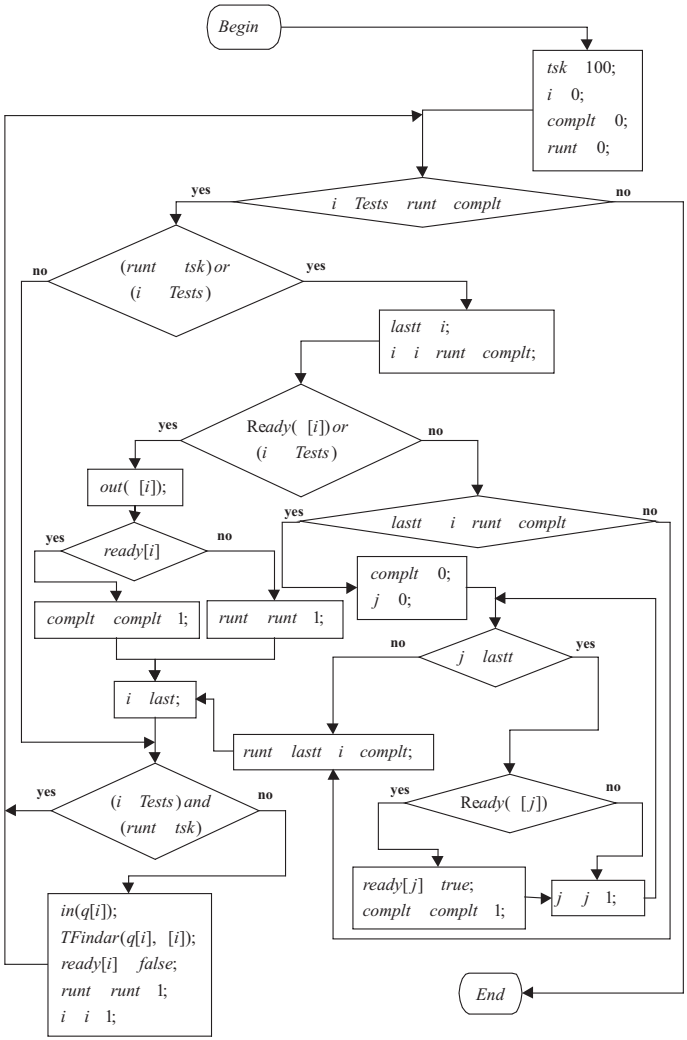

Рис. 6. Блок-схема обработки гранул в программе GlobalSolver

На Рис. 8 изображен график зависимости времени на вычисление задачи от номера задачи для пар штрихов, задачи упорядочены

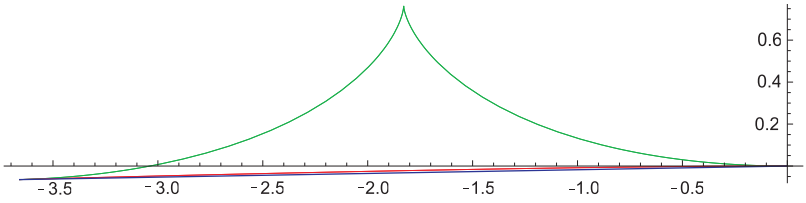

Рис. 7. Пример тяжелой задачи для пары штрихов

по времени вычисления. В этой серии присутствует лишь несколько задач, которые по времени вычисляются на порядок дольше, чем остальные. Таким образом, 1472 задачи были посчитаны за 1568.58 секунд.

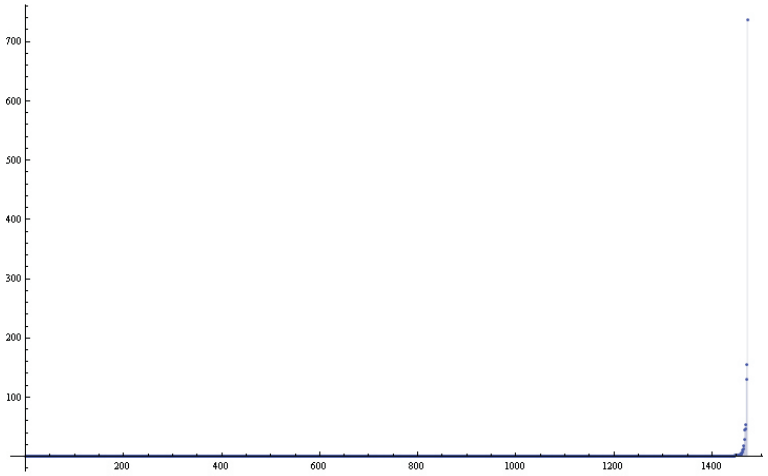

Рис. 8. Распределение времени до отсева ≪тяжелых≫ задач.

На Рис. 9 изображен график зависимости времени на вычисление задачи от номера задачи для пар штрихов после отсева ≪тяжелых≫ задач с помощью описанного критерия. Вместе с тяжелыми задачами в примере отсеивается примерно треть от всех задач, для вычисления остаются 979 задач. Время, требуемое для их вычисления, равно 132.9 секунды, это больше чем в десять раз меньше чем вместе с тяжелыми задачами. Обработка отсеянных задач занимает

меньше десятой доли секунды, отсюда имеем аналогичное решение серии задач, для которого требуется в 10 раз меньше времени, чем раньше. Заметим, что для каждой серии ускорение будет разным и зависит от рода и количества «тяжелых» задач, которые в ней содержатся.

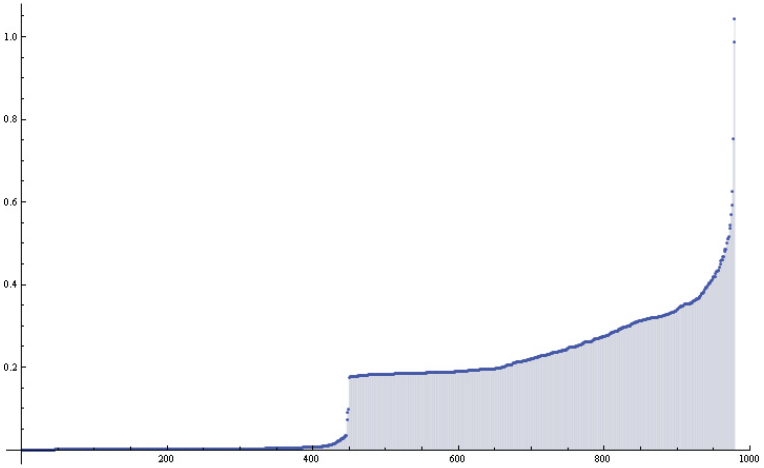

Рис. 9. Распределение времени после отсева «тяжелых» задач.

Как видно из рисунка 9, в этом примере после отсева каждая задача считается меньше секунды.

3.5. Тестирование программы GlobalSolver

Была написана программа GlobalSolver, поддерживающая как последовательный так и параллельный режимы работы, которая по списку входных значений «x0, y0, th0, x1, y1, th1, col» (где {x0, y0, th0} — начальная точка искомой кривой; {x1, y1,th1} — конечная точка искомой кривой; col — идентификатор цвета кривой) выдает список строк типа «u, v, k, nC, s1, col» (где {u, v, k, s1} — искомый корень соответствующего уравнения для кривой; nC — номер области C_i; col — идентификатор цвета кривой), с помощью которого можно построить по формулам работы [5] соответствующие входным данным оптимальные кривые — изофоты, восстанавливающие поврежденное изображение.

Таблица 1. Тестирование программы GlobalSolver (979 тестов)

Число узлов: i	**Время работы программы:** T**, с**	**Ускорение:** $a = \frac{T_1}{T_i}$	**Коэффициент эффективности:** $CoE = \frac{a}{i} \times 100$**, %**
1	58.519	1.000	100.00
2	30.490	1.919	95.95
3	20.836	2.808	93.36
4	16.474	3.552	88.81

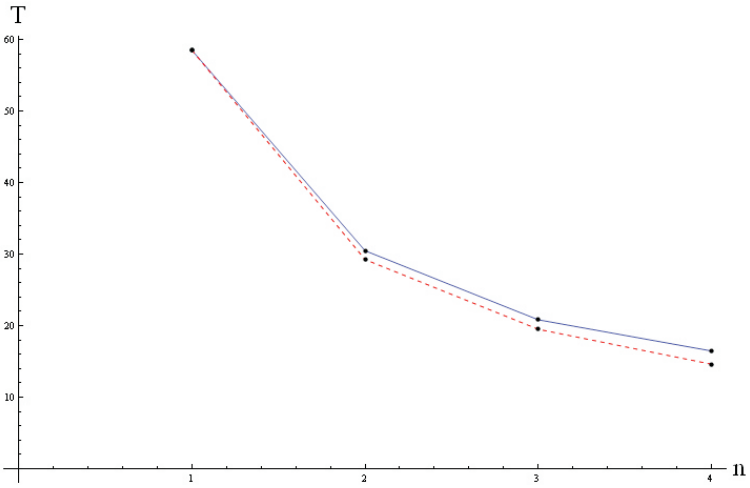

Рис. 10. Зависимость времени выполнения программы GlobalSolver от количества узлов

Параллельная версия программы GlobalSolver была протестирована на кластере blade.botik.ru. Тестирование проводилось на 1, 2, 3 и 4 узлах кластера blade, каждый узел которого состоит из двух процессоров, а каждый процессор — из 4 ядер. Посчитано время выполнения программы (T_i, сек.), ускорение ($a = \frac{T_1}{T_i}$) и коэффициент эффективности ($CoE = \frac{T_1}{i \times T_i} \times 100$, %). На Рис. 10–12 изображен пример зависимости этих величин от количества узлов, на котором

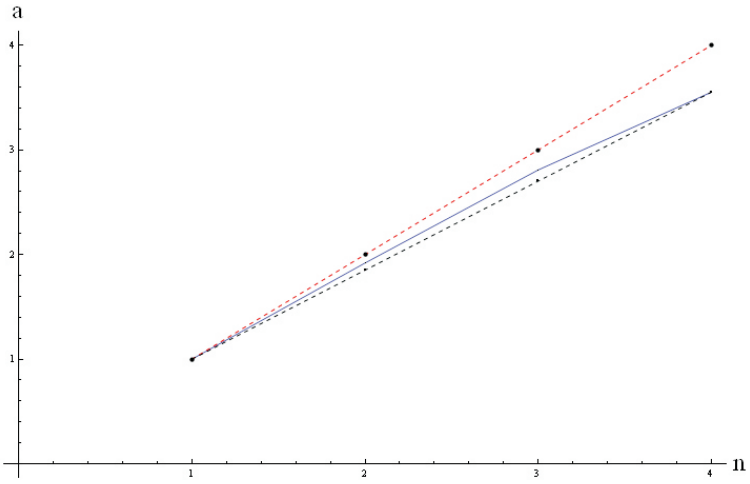

Рис. 11. Ускорение времени работы программы GlobalSolver

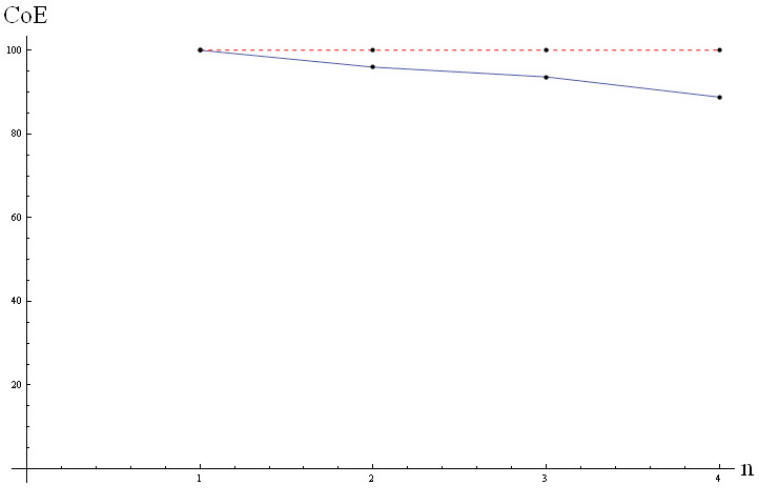

Рис. 12. Коэффициент эффективности GlobalSolver

производится расчет, красным пунктиром показан идеальный случай распараллеливания, синей сплошной линией показаны реальные данные.

Приведенные в данном разделе таблицы и графики демонстрируют эффективность описанного подхода к распараллеливанию процесса вычисления семейства изофот при восстановлении поврежденного изображения на основе вариационного метода.

4. Заключение

В работе описан алгоритм распараллеливания решения задачи восстановления изофот изображения на основе вариационного подхода. Приведены показатели эффективности разработанной C++ программы в библиотеке TSim.

Список литературы

[1] Chan T., Kang S., Shen J. *Euler's elastica and curvature based inpainting* // SIAM Journal of Applied Math., 2002. **63**, no. 2, p. 564–592. ↑[]

[2] Citti G., Sarti A. *A cortical based model of perceptual completion in the roto-translation space* // J. Math. Imaging Vision, 2006. **24**, no. 3, p. 307–326. ↑[]

[3] Petitot J. *The neurogeometry of pinwheels as a sub-Riemannian contact structure* // J. Physiology, 2003. **97**, p. 265–309. ↑[]

[4] Petitot J. Neorogeometrie de la vision. Modeles mathematiques et physiques des architectures fonctionelles : Editions de l'Ecole Polytechnique, 2008. ↑[]

[5] Sachkov Yu. L. Cut locus and optimal synthesis in the sub-Riemannian problem on the group of motions of a plane, 2010 (принята к публикации), http://arxiv.org/abs/0903.0727v1. ↑[], 1.2, 1.3, 3.5

[6] Московский А. А. *T-Sim — библиотека для параллельных вычислений на основе подхода T-системы* // Международная конференция ≪Программные системы: теория и приложения≫ (Переславль-Залесский, октябрь 2006). — М. : Наука. Физматлит, 2006. Т. **1**, с. 183–193. ↑[], 1.2

[7] Round-robin scheduling : Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Round-robin_scheduling. ↑3.2.3

Yu. L. Sachkov, A. A. Ardentov, A. P. Mashtakov. *Parallel algorithm and software for recovery of isophotes for corrupted images.*

Abstract. An experience of parallel solution to the problem of recovery of curves at corrupted images via variational approach is presented. Efficiency indices for C++ software in TSim parallel programming library are provided.

Key Words and Phrases: optimal control, recovery of images, parallel programming.

Образец ссылки на статью:

Ю. Л. Сачков, А. А. Ардентов, А. П. Маштаков. *Параллельный алгоритм и программа восстановления изофот для поврежденных изображений* // Программные системы: теория и приложения : электрон.

научн. журн. 2010. № 1(1), с. 3–20. URL: http://psta.psiras.ru/read/psta2010_1_3-20.pdf