МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
”Новосибирский национальный исследовательский государственный университет”
(Новосибирский государственный университет)
Структурное подразделение Новосибирского государственного университета –
Высший колледж информатики НГУ
КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Создания модуля для параллельного решения бигармонического уравнения методом Монте-Карло.

Дипломный проект
на квалификацию техник

Студент Семенов С.А.
гр. 903а2 «___»_________2013

Научный руководитель Лукинов В.Л.
к.ф-м.н., н.с ИВМиМГ СО РАН «___»_________2013

Новосибирск 2013

Оглавление

Введение

Дипломная работа посвящена созданию эффективной библиотеки для численного решения первой краевой задачи для бигармонического уравнения методами Монте-Карло.

Официальной датой рождения метода Монте-Карло принято считать 1949 год, когда была опубликована статья С. Улама и Н. Метрополиса [1] . Сам термин был предложен еще во время Второй мировой войны выдающимися учеными XX века математиком Дж. фон Нейманом и физиком Энрико Ферми в Лос-Аламосе (США) в процессе работ по ядерной тематике. Хотя методы Монте-Карло были известны и до 40-х годов, интенсивное развитие статистическое моделирование получило несколько позже в связи с появлением компьютеров, что позволило проводить вычисления больших объемов. С другой стороны, более широкое распространение получает статистическое описание тех или иных сложных физических процессов в связи с чем методы Монте-Карло все более активно используются во многих научных областях (теория переноса, теория массового обслуживания, теория надежности, статистическая физика и др.).

Основными преимуществами данных методов являются:

– физическая наглядность и простота реализации,
– малая зависимость трудоемкости задачи от размерности,
– возможность решения задач со сложной геометрией,
– оценивание отдельных функционалов от решения.... ДОПИСАТЬ КАК У МЕНЯ В ДИССЕРТАЦИИ

Бигармонические уравнения используются при решении задач теории упругости. Например, уравнение изгиба тонких пластин имеет вид $\triangle\triangle u = g$, где u – нормальный прогиб пластины. Если пластина лежит на упругом основании, то u удовлетворяет уравнению $\triangle\triangle u + cu = g$.

В настоящей работе использовались алгоритмы, основанные на двух принципиально разных подходах к решению краевых задач методом Монте-Карло.

Первый подход заключается в сведении исходной дифферениальной задачи к некоторому интегральному уравнению, что дает возможность использовать развитой аппарат методов Монте-Карло для решения интегральных уравнений второго рода. На этой основе строятся алгоритмы “блуждания по сферам”.

Во втором подходе дифференциальная задача заменяется соответствующей разностной, которую после приведения её к специальному виду возможно решить методом Монте-Карло. В рамках этого подхода получаются простые и универсальные алгоритмы “блуждания по решетке”

Несмотря на то, что рассматриваемые в дипломе алгоритмы хорошо изучены, новой и неисследованной является задача изучения данных алгоритмов при вычисления на кластерах. Основная задача дипломной работы состоит в построении шкалируемой вычислительной библиотеки для кластерных вычислений.

При создании требуемой библиотеки были использованы следующие программные средства и технологии.

Message Passing Interface (MPI, интерфейс передачи сообщений) – программный интерфейс (API)[1] для передачи информации, который позволяет обмениваться сообщениями

[1] см. сокращение

между процессами, выполняющими одну задачу. Разработан Уильямом Гроуппом, Эвином Ласком и другими.

MPI является наиболее распространённым стандартом интерфейса обмена данными в параллельном программировании. Существуют его реализации для большого числа компьютерных платформ. MPI используется при разработке программ для кластеров и суперкомпьютеров. Основным средством коммуникации между процессами в MPI является передача сообщений друг другу. Стандартизацией MPI занимается MPI Forum. В стандарте MPI описан интерфейс передачи сообщений, который должен поддерживаться как на платформе, так и в приложениях пользователя. В настоящее время существует большое количество бесплатных и коммерческих реализаций MPI. Существуют реализации для языков Фортран 77/90, Java, Си и Си++.

В первую очередь MPI ориентирован на системы с распределенной памятью, то есть когда затраты на передачу данных велики, в то время как OpenMP[2] ориентирован на системы с общей памятью (многоядерные с общим кэшем). Обе технологии могут использоваться совместно, дабы оптимально использовать в кластере многоядерные системы. Более подробно об этом [3].

Git — распределённая система управления версиями файлов. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день поддерживается Джунио Хамано.

Система управления версиями (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).

Программа является свободной и выпущена под лицензией GNU GPL версии 2.

ДОПИСАТЬ про систему контроля версий GIT, C++, Dll, можно одну строчку про то, что диплом написан в LATEX и что это такое - как форма библиотеки

[2]http://openmp.org/wp/

Глава 1

Постановка задачи

1.1. Конкретизации требований и задачи

Входными условиями вычисления (пользовательскими функциями) является определение:

- функции ϕ;
- функций u, g;
- границ области.

Функции ϕ, u, g соответствуют функциям в уравнении: $(\Delta + c)^{p+1}u = -g, (\Delta + c)^k u|_\Gamma = \phi_k$ Функция границ области возвращает единицу если точка с некоторой погрешностью находится на границе. Входными данными является:

- количество путей;
- начальная точка.

Для задания пользовательских функций мы можем использовать программный код, прессинг функций или скрип. Первый наиболее скор в разработки, но заставляет компилировать программу каждый раз когда мы меняем вычисляемое уравнение. Для борьбы с этим недостатком сделаем вычисление в классе, который вынесем в отдельный модуль. Получаемый модуль параллельного вычисления скомпилируем как статическую библиотеку. Определение пользовательских функций проходит как задания функций обратного вызова. Так-же сделаем шаблон программы для облегчения определения пользователем своих функций. В комплект необходимо вести реализацию под конкретные условия.

С учетом того, что конечный программный продукт будет запускается как с изменением предыдущих параметров так и для частного конкретного случая ввод данных следует сделать с помощью аргументов и(или) файлов данных.

Вывод осуществляется в tex, html файлы и на экран.

Конкретизируем задачу:

а) Создание статической библиотеки класса с функциями обратного вызова .
б) Создание приложение под конкретные условия.
в) Создание файла данных под программу созданную по предыдущим условиям.
г) Создание справки.

Интерфейс программы смотреть приложение ”Справка”.

1.2. Формулировка задачи

1.3. Аналоги

Главным аналогом на основе которого и разрабатывается приложение является программа Biharmon2, чей код приведен в приложении. Недостатком данной реализации алгоритмов является:

– необходимость изменять алгоритм и функции основной программы(малая степень защиты от дурака);
– последовательность вычислений;
– при изменении алгоритма вычисления меняется и часть программы.

1.4. блуждание по сферам

Рассмотрим задачу Дирихле для уравнения Гельмгольца

$$\Delta u + cu = g, u|_\Gamma = \phi$$

в области $D \subset R^n$ с границей Γ , причем $c < c^*$, где c^* первое собственное число оператора Лапласа для области $D, r = (x1; :::; xn) \in D$. Предполагаются выполненными сформулированные условия регулярности функций g, ' и границы Γ, обеспечивающие существование и единственность решения задачи , а также его вероятностное представление и интегральное представление с помощью шаровой функции Грина.

Введем следующие обозначения:
– $\bar{D}$ - замыкание области D;
– $d(P)$ - расстояние от точки P до границы Γ;
– $\epsilon > 0$ - числовой параметр;
– Γ_ϵ - ϵ - окрестность границы Γ, т. е. $\Gamma_\epsilon = \{P \in \bar{D} : d(P) < \epsilon\}$;
– $S(P)$ максимальная из сфер (точнее - из гиперсфер) с центром в точке P, целиком лежащих в $\bar{D}, S(P) = \{Q \in \bar{D} : |Q - P| = d(P)\}$.

В процессе блуждания по сферам очередная точка P_{k+1} выбирается равномерно по поверхности сферы $S(P_k)$; процесс обрывается, если точка попадает в Γ_ϵ. Дадим точное определение процесса блуждания по сферам. Зададим цепь Маркова $\{R_m\}_{m=1,2,...,N}$ следующими характеристиками:
– $\pi(r) = \delta(r - r_0)$ - плотность начального распределения (т.е. цепь выходит из точки r_0);
– $p(r, r') = \delta_r(r')$ плотность перехода из r в r', представляющая собой обобщенную плотность равномерного распределения вероятностейна сфере $S(r)$;
– $p_0(r)$ вероятность обрыва цепи, определяемая выражением

$$p_0(r) = \begin{cases} 0, r \notin \Gamma_\epsilon \\ 1, r \notin \Gamma_\epsilon \end{cases}$$

– N - номер последнего состояния.

Как уже указывалось, данная цепь называется процессом блуждания по сферам. Ее можно, очевидно, записать в виде $r_m = r_{m-1} + \omega_m d(r_{m-1}); m = 1; 2; ....; \omega_m$ – последовательность независимых изотропных векторов единич- длины.

1.4.1. Оценки решения метагармонического уравнения$(\delta + c)^p u = g$

Для случая $L = \delta, \lambda = 0, c = const < c*$ вероятносное представление задачи имеет вид

$$u(r_0) = E \int_0^\gamma e^{ct} g(\xi(t))dt + E[e^{c\tau} \Phi(\xi(\tau))]$$

, где $\xi(t)$ начинающийся в точке r_0 соответствующий оператору Лапласа диффузионный процесс, τ момент первого выхода процесса из области D. На основе строго марковского свойства процесса отсюда имеем

$$u(r_0) = \sum_{i=0}^{\infty} E[e^{c\tau_i} \int_0^{\tau_{i+1}-\tau_i} e^{ct} g(\xi(t + \tau_i))dt + E[\Phi(\xi(\tau)) \prod_{i=0}^{\infty} e^{c(\tau_{i+1}-\tau_i)}]$$

, где τ_i - момент первого выхода процесса $\xi(t)$ на поверхность i-й сферы соответствующего блуждание по сферам.

1.5. Численные результаты

Metot	N	t_{cp}	ΔN
Par1	100003	1 sec	$+/-200$
Par1	1000030	10 sec	$+/-250$
Par2	100003	1 sec	$+/-10$
Par2	1000030	12 sec	$+/-10$
Pos	100003	2 sec	$-$
Pos	1000030	25 sec	$-$

Таблица 1.1: Время выполнения и расхождения

1.6. Заключение

Сформируем основные результаты работы:

а) Создана статической библиотеки класса с функциями обратного вызова .

б) Создана приложение под конкретные условия.

в) Оценино оптимизация для двух алгоритмов и двух методов решения

1.7. Руководство пользователя

1.7.1. Установка

Для сборки приложеия под Windows необходимо MPICH2, набор унтилит для компиляции: компилятор GNU GCC и GNU Make данные унтилиты представленны в пакете MinGW. Для Linux GNU GCC не обязателен, компиляция происходит силами пакета MPICH2.

а) Скачайте необходимую версию библиотеки с тестовым примером.

б) Запустите консоль в папке проэкта или перейдите в нее с помощью команды cd.
   - нажмите Пуск -> Выполнить -> cmd - Это откроет консоль Windows;
   - в консоли наберите имя диска на котором распологается проэкт с двоиточеем на конце (C:);
   - там же напечатайте cd <путь к проэкту > (cd C:
     project).

в) Запустите make.

Результатом станет скомпилированный тестовый пример и статическая библиотека находящиеся в папке с проэктом.

1.7.2. Запуск приложения

Запуск приложения описан в файле README и документации к MPICH2

Литература

1. S. Ulam and N. Metropolis. The Monte Carlo method, Journal of American Statistical Association, 44, 335, 1949.

2. Лукинов Виталий Леонидович *Скалярные Алгоритмы метода Монте-Карло для решения мета-гармонических уравнений* 2005

3. http://www.mpich.org/

Приложение А

Исходные коды и диаграммы программ

А.1. Основной аналог - Biharmon2 измененная

```
#include <iostream>
#include <stdlib.h>
#include <cmath>                        // for trigonometry functions

using namespace std;

 const double epsilon=0.001e-00;
 const double xfirst=0.5e-00;
 const double yfirst=0.5e-00;
 const double PI=3.141592e-00;
 double x;
 double y;



 double u(double x, double y)
 {
         return double(exp(x)*exp(y));
         //return double(sin(x)*sin(y)*sin(z));
 }

 double g(double x, double y)
 {
         return double(-4.0*exp(x)*exp(y));
        // return double(-4.0*sin(x)*sin(y)*sin(z));
 }
 double phi_0(double x, double y)
 {
     return double(exp(x)*exp(y));
        // return double(sin(x)*sin(y)*sin(z));
 }
 double phi_1(double x, double y)
 {
         return double(2.0*exp(x)*exp(y));
        // return double(-2.0*sin(x)*sin(y)*sin(z));
 }
```

```
int boundary(double x1, double y1) // являетсялиточкаграницей
{
        if (x1<epsilon) {x=0.0; return 0;};
        if (x1>(1-epsilon)) {x=1.0; return 0;};
        if (y1<epsilon) {y=0.0; return 0;};
        if (y1> (1-epsilon)) {y=1.0; return 0;};
        return 1;

}

double min(double a, double b)
{
        if ( b>=a)  return a;
           else return b;

}
double diam(double x, double y) // диаметрматрицы ?
{
        if (x> fabs(1-x)) x= fabs(1-x); // fabs - абсолютноезначениедля

//аргумента сплавоющейточкой
        if (y>fabs(1-y)) y=fabs(1-y); // пишемвглобальныепеременные
        return min(x,y); // возвращаемминимальное R ?

}
double stat2(double a, double b, double c)
{
     if ( c>=b) if (b>=a) return b;
                else if (a>=c) return c;
                                     else return a;
    else if (c>=a) return c;
                  else if (b>=a)return a;
                    else return b;

}



int main()

{
  double N; // количествопутей

  cout << "\nInput␣a␣number␣of␣ways\n";
  cin >> N;
  double U=0;
  double Disp=0;
  for (int k= 0; k<N; k++ )
  {


                     x=xfirst;
```

```
                y=yfirst;

                int SN=0;
                double S=0;
                double S1=0;

                while (boundary(x,y)) // поканеграница
            {


                  double d=diam(x,y);
                  double alpha= double (rand())/ double(RAND_MAX
?
                  double omega1=cos(2*PI*alpha);
                  double omega2=sin(2*PI*alpha);
                  alpha= double (rand())/ double(RAND_MAX);
                  double om1=cos(2*PI*alpha);
                  double om2=sin(2*PI*alpha);
                  int indeks=1;
                  double alpha1=0;
                  double alpha2=0;
                  while(indeks)
                  {
                        alpha1=double (rand())/ double(RAND_MA
                        alpha2=double (rand())/ double(RAND_MA
                        alpha2=4*alpha2/exp(1);
                        if (alpha2<(-4*alpha1*log(alpha1))) in

                  }
                     //cout<< "\n  "<<alpha1 <<" "<< alpha2 <<

                  double nu=alpha1*d;
                  S=S+(S1-((d*d-nu*nu-nu*nu*log(d/nu))/log(d/nu)
                  SN=SN+1;
                  S1=S1-d*d;
                  x=x+omega1*d;
                  y=y+omega2*d;
                  //cout << "\n  "<<x <<" "<< y<<  ;
            };
            //cout << "\n boundary  "<<x <<" "<< y ;
            S=S+(S1/4)*phi_1(x,y)+phi_0(x,y);
            //S=S+phi_0(x,y,z);
            cout <<"\n␣␣␣␣"<< SN;
            U=U+S/N;
            Disp=Disp+(S*S)/N;


    };
    Disp=Disp-U*U;
    Disp=sqrt(fabs(Disp)/N);

    cout << "\nPresise␣solution␣␣␣"<< u(xfirst,yfirst) ;
```

```
    cout << "\nNumerical␣solution␣␣␣␣"<< U ;
    cout << "\nDelta␣␣␣"<< fabs(u(xfirst,yfirst)-U) <<"\n" ;
    cout << "\nDisp␣␣␣"<< Disp <<"\n" ;

        return 0;

}
```

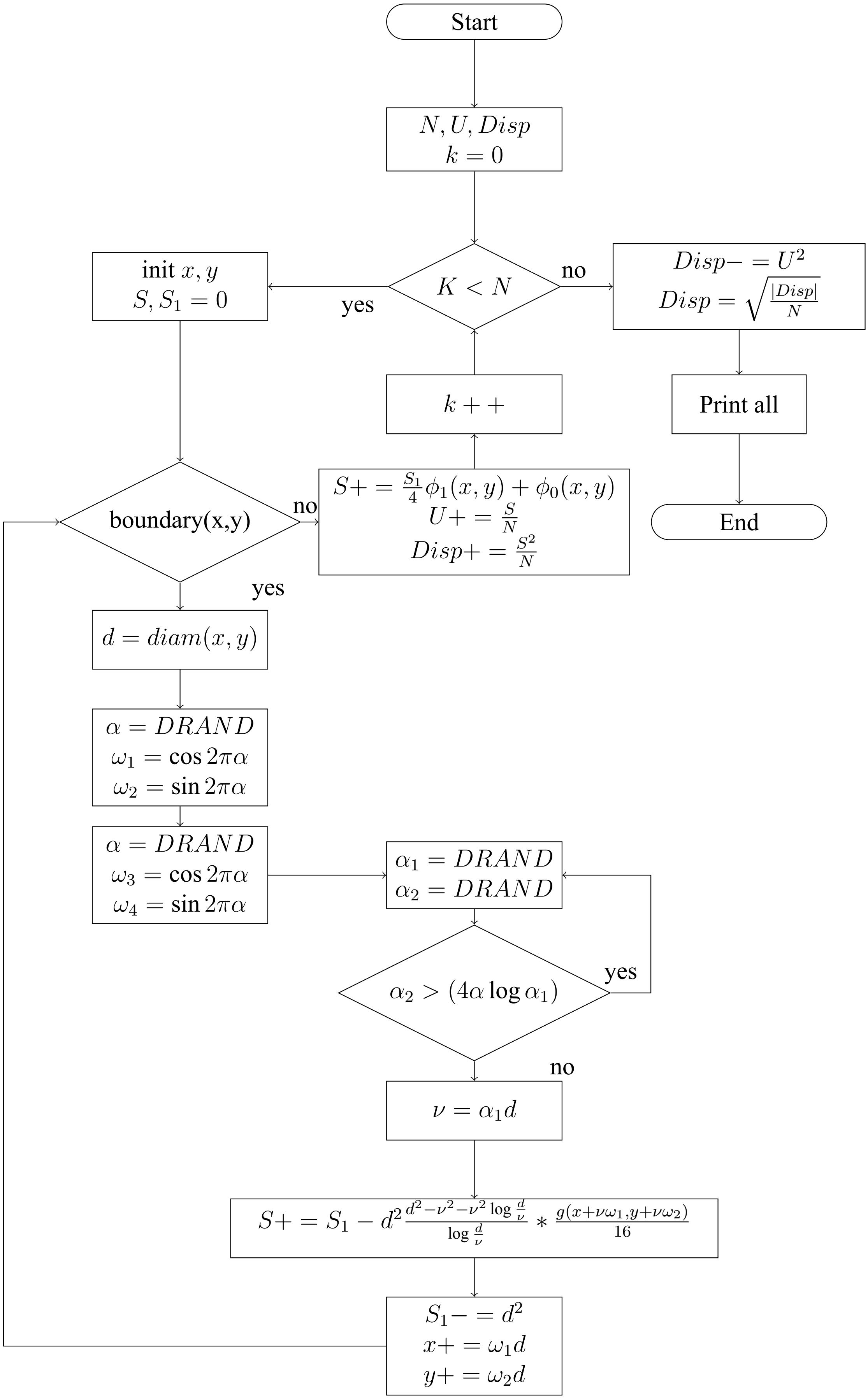

Рис. А.1: Принцип действия программы

Приложение B

Справка

B.1. Исполняемая программа

biharmon -l |[-d]| -h
biharmon -l |[-d]| -h [filename]
- l Latex-file - происходит создание файла report.tex
- d Display - вывод данных на дисплей
- h html-file - происходит создание файла report.html
Если файл существует к имени добавляется индекс.

filename - файл данных;
Приоритет -l, -h, -d

B.2. Библиотека

Открытые методы класса
init(int argc,char *args[])
setFunPhi();

B.3. Файл данных

Построчный.
X Y [SectionName]

Приложение C

Выходные данные

C.1. tex,html

Отчет

N=10^4

x, y	h	$u(r)$	$u\ (r)$	$\lvert u(r)\rvert$
$0.5, 0.5$	0.1	0.22	0.22	0.0004

Приложение D

Тезаурус

API - Интерфейс программирования приложений (иногда интерфейс прикладного программирования) (англ. application programming interface) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах.

MPI - Message Passing Interface (интерфейс передачи сообщений) — API для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу.