Московский государственный университет им. М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра оптимального управления

Курсовая работа

Построение множеств достижимости для плоских управляемых систем с использованием технологии CUDA для параллельных вычислений

студентки 313 группы
Крашенинниковой А.О.

Научный руководитель, старший преподаватель
Аввакумов С.Н.

Москва
2010

Содержание

1 Введение

Оптимальное управление охватывает широкий круг задач, в которых при определенных ограничениях на ресурсы требуется минимизировать (максимизировать) заданный критерий качества. Задачи оптимального управления встречаются в различных областях науки, техники, медицины, экономики. Например: задачи ядерной энергетики (управление охлаждением реактора), робототехники (движение роботов, управление всевозможными станками и автоматами), механики полета (самонаводящиеся ракеты, автопилоты, автоматическая стыковка на орбите, управление самолетом), экономики (задачи долговременного планирования), экологии (расчет допустимого воздействия на экосистему), биофизики и т.д.

В задачах оптимального управления важнейшую роль играет множество достижимости. Оно характеризует все возможные положения управляемой системы в каждый момент времени.

В курсовой работе изучены различные алгоритмы построения множеств достижимости для плоских управляемых систем. Так как эти алгоритмы позволяют эффективно разделить процедуру вычислений на множество независимых параллельных процессов, то для программирования используется технология CUDA (Compute Unified Device Architecture) с Си-подобным языком программирования. Использование такой технологии помогает уменьшить время вычислений в несколько десятков раз.

2 Постановка задачи оптимального управления

В рассматриваемой задаче оптимального управления требуется перевести объект из множества начальных состояний M_0 на множество конечных состояний M_1 за счет выбора допустимого управления $u = u(t)$ из класса допустимых управлений Y_U так, чтобы функционал J принимал минимальное значение. Управление $u(t)$, решающее поставленную задачу, оптимальное в смысле интегрального функционала

$$J = \int_{t_0}^{t_1} f^0(t, x(t), u(t))dt.$$

Траектория $x(t)$, отвечающая оптимальному управлению $u(t)$, называется оптимальной траекторией. Таким образом, постановка задачи оптимального управления в краткой форме имеет следующий вид:

$$\begin{cases} \dot{x} = f(t, x, u), & u(t) \in U, \\ x(t_0) \in M_0, & x(t_1) \in M_1, \\ J \to \min\limits_{u(\cdot) \in Y}. \end{cases}$$

Рассматриваемая задача требует задания следующего набора исходных данных: $\{\ f_0,\ f,\ Y = Y_U;\ M_0,\ M_1,\ t_0\ \}$, где f_0, f - непрерывные функции, множества M_0, M_1, U выпуклы и компактны.

3 Множество достижимости

3.1 Определение

При изучении задачи оптимального управления большое значение имеет множество достижимости $X(t_0, t, M_0)$. Это множество зависит от следующего набора параметров:

- M_0 – множество начальных значений
- t_0 – начальный момент времени
- $t \geq t_0$ – рассматриваемый момент времени
- f – функция, в линейном случае зависящая от матрицы A
- $Y = Y_U$ – класс допустимых управлений

Множество достижимости отвечает на вопрос: куда можно перейти к моменту времени t по траекториям дифференциального уравнения ($\dot{x} = f(t, x, u)$), исходящим в начальный момент времени t_0 из различных точек множества начальных значений M_0 при использовании всевозможных управлений $u(\cdot) \in Y$? Множество в момент времени t концов всех таких траекторий образует множество достижимости – $X(t_0, t, M_0)$.

Получим формулу для аналитического представления множества достижимости в случае линейной управляемой системы:

$$\begin{cases} \dot{x} = Ax + u, & u(t) \in U, \\ x(t_0) \in M_0, & x(t_1) \in M_1, \\ J \to \min\limits_{u(\cdot) \in Y}. \end{cases}$$

Рассмотрим задачу Коши

$$\begin{cases} \dot{x} = Ax + u, & t_0 \leq t, \\ x(t_0) = x_0. \end{cases}$$

Формула Коши для её решения выглядит следующим образом:

$$x(t) = e^{(t-t_0)A}x_0 + \int\limits_{t_0}^{t} e^{(t-s)A}u(s)ds.$$

Тогда формула для множества достижимости имеет вид:

$$X(t_0, t, M_0) = \bigcup_{\substack{x_0 \in M_0 \\ u(\cdot) \in Y}} \left\{ e^{(t-t_0)A}x_0 + \int\limits_{t_0}^{t} e^{(t-s)A}u(s)ds \right\},$$

или

$$X(t) = X(t_0, t, M_0) = e^{(t-t_0)A} M_0 + \int_{t_0}^{t} e^{(t-s)A} Y_U(s) ds.$$

3.2 Основные свойства

- Компактность и выпуклость
  - если множество начальных значений $M_0 \in \Omega(E^n)$ компактно, то множество достижимости тоже является компактным $X(t) \in \Omega(E^n)$ [2]
  - если множество начальных значений есть выпуклый компакт $M_0 \in \operatorname{conv}\Omega(E^n)$, то множество достижимости тоже является выпуклым компактом $X(t) \in \operatorname{conv}\Omega(E^n)$, здесь $\Omega(E^n)$ – множество всех непустых компактов E^n
- Непрерывность
  - множество достижимости $X(t)$ непрерывно зависит от времени t: $h(X(t'), X(t)) \to 0$ при $t' \to t$, здесь h – метрика Хаусдорфа

Замечание 1 *Множество достижимости $X(t_0, t, M_0)$ никак не зависит от множества конечных состояний M_1.*

4 Алгоритмы построения множеств достижимости

4.1 Первый метод построения множеств достижимости

Определение 1 *Гиперплоскость $G_{\psi^0} = \{x \in E^n : (x, \psi^0) = c(\psi^0)\}$ называется опорной гиперплоскостью компакта в направлении ψ^0.*

[2]

Множество $H_{\psi^0} = U \bigcap G_{\psi^0}$ – опорное множество компакта U в направлении ψ^0. [2]

$H_{\psi^0} = \{u \in U : (u, \psi^0) = c(\psi^0)\} \neq 0$ – опорное множество H_{ψ^0} состоит из тех точек $u \in U$, на которых при $\psi = \psi^0$ достигается максимум $c(\psi)$.

Сформулируем теорему о градиенте опорной функции:

Теорема 1 *Пусть $U \in \operatorname{conv}\Omega(E^n)$, $c(\psi)$ – опорная функция выпуклого компакта U, H_{ψ^0} – опорное множество компакта U в направлении $\psi^0 \neq 0, \psi \in E^n$*

- *Если в точке ψ^0 существует градиент $c'(\psi^0)$ опорной функции $c(\psi)$, то опорное множество H_{ψ^0} состоит из единственной точки h_0, причем $h_0 = c'(\psi^0)$, т.е. опорная точка h_0 совпадает с градиентом опорной функции в точке ψ^0.*
- *Если опорное множество H_{ψ^0} выпуклого компакта U состоит из единственной точки h_0, то опорная функция $c(\psi)$ имеет в точке ψ^0 градиент $c'(\psi^0)$, причем $c'(\psi^0) = h_0$.*

Следствие 1 *Применение теоремы о градиенте опорной функции для построения границы строго выпуклого компакта по его опорной функции :*

параметрическое уравнение границы выглядит следующим образом:

$$x = c'(\psi)\big|_{\psi=\binom{\cos\alpha}{\sin\alpha}\equiv q(\alpha)},$$

$$\begin{cases} x_1 = c'_{\psi_1}(\psi)\Big|_{\psi=q(\alpha)}, \\ x_2 = c'_{\psi_2}(\psi)\Big|_{\psi=q(\alpha)}, \end{cases} \quad \alpha \in [0, 2\pi),$$

или, в другой форме:

$$c_0(\alpha) = c(\psi)\big|_{\psi=q(\alpha)},$$

$$\partial U : x = c_0(\alpha)q(\alpha) + c'_0(\alpha)q(\alpha)$$
$$\Updownarrow$$
$$\begin{cases} x_1 = c_0(\alpha)\cos\alpha - c'_0(\alpha)\sin\alpha, \\ x_2 = c_0(\alpha)\sin\alpha + c'_0(\alpha)\cos\alpha, \end{cases} \quad \alpha \in [0, 2\pi].$$

Таким образом, можно построить множество достижимости плоской линейной управляемой системы, зная его опорную функцию.

Параллельные вычисления для данного метода уместны на стадии вычисления координат границы множества (зависящих от параметра α): отрезок $[0; 2\pi]$ разбиваем на n равных отрезков и параллельно строим участки границы для каждого их них.

4.2 Второй метод построения множеств достижимости

Сформулируем принцип максимума Понтрягина для линейной задачи быстродействия:

Теорема 2 *Рассмотрим пару:*

$$(x(t), u(t)), \quad t_0 \le t \le t_1,$$

где

- $u(\cdot) \in Y_u$, *т. е.* $u(t)$ *- допустимое управление, определенное на отрезке* $t_0 \le t \le t_1$, *причем в каждый момент времени* $t \in [t_0; t_1]$ *значение* $u(t) \in U$
- $x(t)$ *- траектория, отвечающая управлению* $u(t)$, *т.е.* $\dot{x} = Ax(t) + u(t)$ *для почти всех* $t \in [t_0; t_1]$, *и удовлетворяющая краевым условиям:* $x(t_0) \in M_0$ *и* $x(t_1) \in M_1$

Будем говорить, что эта пара $(x(t), u(t))$ *удовлетворяет принципу максимума Понтрягина на отрезке* $[t_0, t_1]$, *если существует такая сопряженная переменная (ненулевое решение сопряженного уравнения), что выполнены следующие три условия:*

- *условие максимума:* $(u(t), \psi(t)) = c(U, \psi(t))$ *для почти всех* $t \in [t_0, t_1]$
- *условие трансверсальности на множестве* M_0*:*

$$(x(t_0), \psi(t_0)) = c(M_0, \psi(t_0))$$

- *условие трансверсальности на множестве* M_1*:*

$$(x(t_1), -\psi(t_1)) = c(M_1, -\psi(t_0))$$

Следствие 2 *Рассмотрим задачу Коши:*

$$\begin{cases} \dot{x} = Ax + c'(\psi), & x(t_0) = x_0, \\ \dot{\psi} = -A^T\psi, & \psi(t_0) = p, \quad p \in S. \end{cases}$$

Здесь $c'(\psi)$ *- градиент опорной функции области управления. Применим теорему о градиенте опорной функции для* $c(U, \psi) = c(\psi)$*. Вектор* p *пробегает единичную сферу* S*. Решение задачи Коши для сопряженной переменной имеет вид:*

$$\psi(t, p) = e^{-tA^T}p.$$

Подставив его для каждого значения p *в первое дифференциальное уравнение для* x*, получим точки, лежащие на границе множества достижимости.*

Параллелизацию вычислений удобно проводить, подставляя различные значения параметра p. На этой стадии вычисления производятся независимо друг от друга. Для большей точности необходимо рассматривать много точек, что потребовало бы большего времени при последовательных вычислениях. Поэтому такой способ распараллеливания является эффективным.

5 Реализация методов построения множеств достижимости для плоских управляемых систем с использованием математической среды Maple и Matlab

5.1 Алгоритмы

Для первого метода: исходными данными являются следующие параметры: $\{f_0, A, U, M_0, t_0\}$ Множество управления и множество начальных значений задаются через их опорные функции. Выражаем через опорные функции этих множеств опорную функцию множества достижимости.

```
ex := s -> exponential((t - s) * transpose(A));
CX := psi -> CM0(ex(t0) . psi) + integral(s -> CU(ex(s) . psi), t0, t, 20);
```

Строим границу множества достижимости, используя параметрические уравнения.

```
 getX := proc(Cv) local C:
         C := (psi1, psi2) -> Cv(Vector([psi1, psi2])):
         subs([psi1 = cos(phi), psi2 = sin(phi)], diff(C(psi1, psi2), psi1));
```

```
 end proc:

 getY := proc(Cv) local C:
         C := (psi1, psi2) -> Cv(Vector([psi1, psi2])):
         subs([psi1 = cos(phi), psi2 = sin(phi)], diff(C(psi1, psi2), psi2));
 end proc:

 DrawSet := proc(C)
         plot([getX(C)(phi), getY(C)(phi), phi = 0..2*Pi], color = COLORR);
 end proc:
```

Для второго метода: исходными данными являются те же параметры, что и в первом методе: $\{f_0, A, U, M_0, t_0, \psi_0\}$ Множество управления также задается через его опорную функцию. Множество начальных значений состоит из одной точки.

```
gradC = matlabFunction([diff(C, psi1); diff(C, psi2)], 'vars', {[psi1; psi2]});
ansx = zeros(length(t_levels), nalpha);
ansy = zeros(length(t_levels), nalpha);
i = 0;
for alpha = linspace(0, 2*pi, nalpha)
    dx = @(t, x) [A*x(1:2) + gradC(x(3:4)); -A'*x(3:4)];
    sol = ode45(dx, [0 T], [x0; cos(alpha); sin(alpha)]);
    i = i + 1;
    ansx(:, i) = deval(sol, t_levels, 1);
    ansy(:, i) = deval(sol, t_levels, 2);
end
```

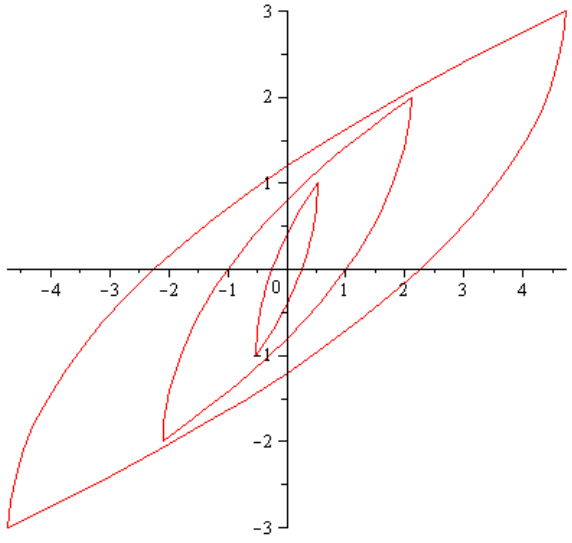
5.2 Пример: задача о тележке

5.2.1 Задача №1

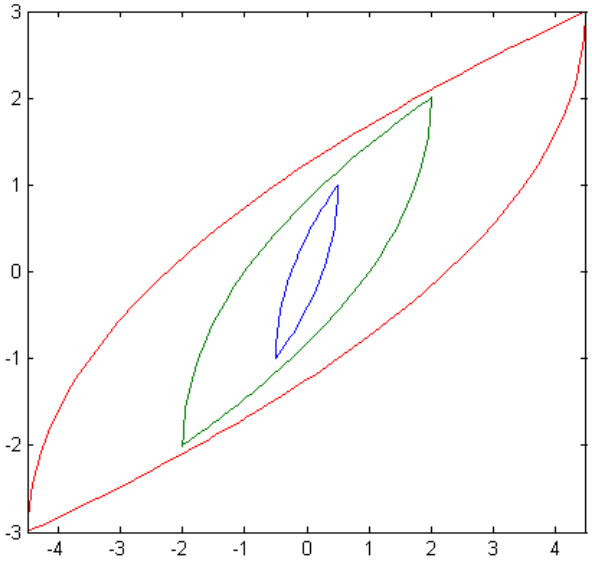
Постановка задачи:

- $T \geq t_0$ – рассматриваемый момент времени
- $f_0 = 1$
- $A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ – матрица системы
- $c(U) = |\psi_2|$ – опорная функция области управления
- $t_0 = 0$ – начальный момент времени
- $M_0 = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}$ – множество начальных значений.

Полученные в результате применения двух методов множества:



применение 1 метода при T = 1, T = 2, T = 3



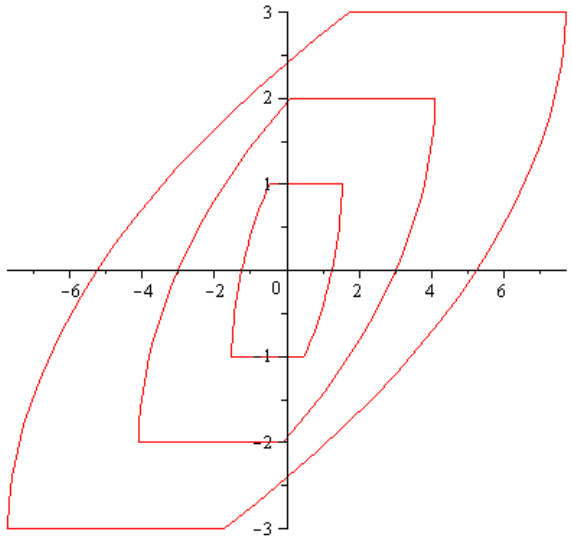
применение 2 метода при T = 1, T = 2, T = 3

5.2.2 Задача №2

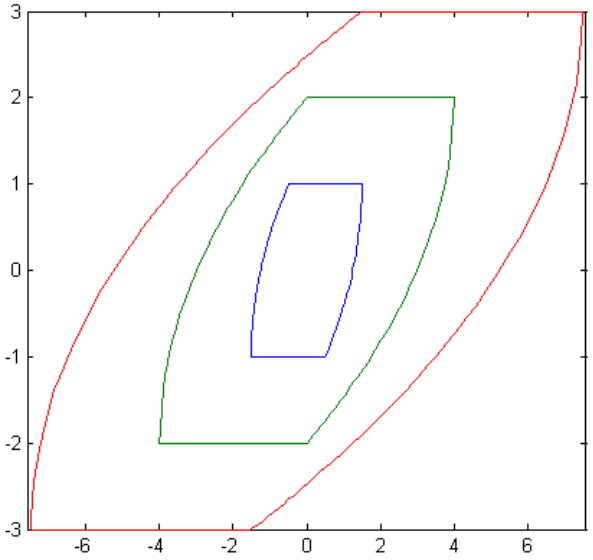
Постановка задачи:

- $T \geq t_0$ – рассматриваемый момент времени
- $f_0 = 1$
- $A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ – матрица системы
- $c(U) = |\psi_1| + |\psi_2|$ – опорная функция области управления
- $t_0 = 0$ – начальный момент времени
- $M_0 = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}$ – множество начальных значений.

Полученные в результате применения двух методов множества:



применение 1 метода при T = 1, T = 2, T = 3



применение 2 метода при T = 1, T = 2, T = 3

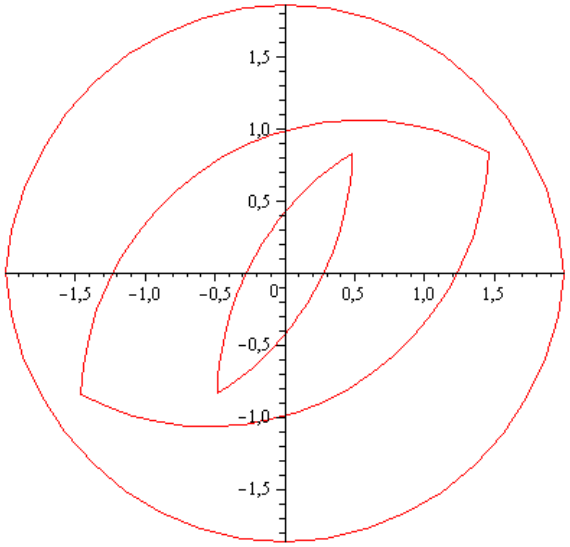
5.3 Пример: задача о математическом маятнике

5.3.1 Задача №3

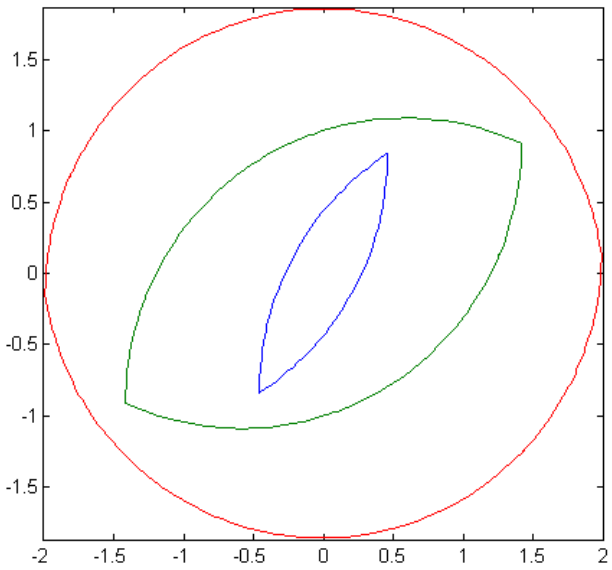
Постановка задачи:

- $T \geq t_0$ – рассматриваемый момент времени
- $f_0 = 1$
- $A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ – матрица системы
- $c(U) = |\psi_2|$ – опорная функция множества управления
- $t_0 = 0$ – начальный момент времени
- $M_0 = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}$ – множество начальных значений

Полученные в результате применения двух методов множества:



применение 1 метода при T = 1, T = 2, T = 3



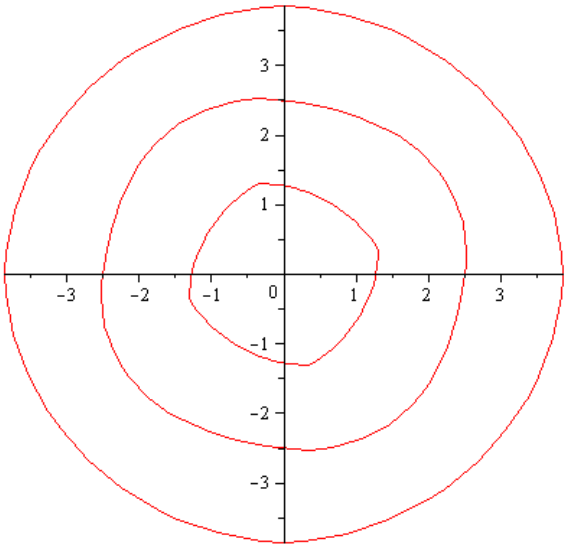
применение 2 метода при T = 1, T = 2, T = 3

5.3.2 Задача №4

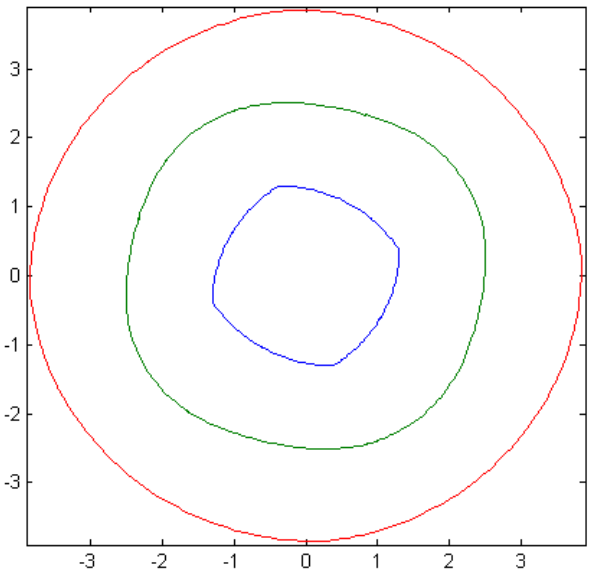
Постановка задачи:

- $T \geq t_0$ – рассматриваемый момент времени
- $f_0 = 1$
- $A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ – матрица системы
- $c(U) = |\psi_1| + |\psi_2|$ – опорная функция множества управления
- $t_0 = 0$ – начальный момент времени
- $M_0 = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}$ – множество начальных значений

Полученные в результате применения двух методов множества:



применение 1 метода при T = 1, T = 2, T = 3



применение 2 метода при T = 1, T = 2, T = 3

6 CUDA

6.1 Определение

CUDA (англ. Compute Unified Device Architecture) — технология GPGPU (англ. General-Purpose computing on Graphics Processing Units), позволяющая программистам реализовывать на упрощённом языке программирования Си алгоритмы, выполнимые на графических процессорах ускорителей GeForce восьмого поколения и старше (GeForce 8 Series, GeForce 9 Series, GeForce 200 Series), Nvidia Quadro и Tesla компании Nvidia. Технология CUDA разработана компанией nVidia.

Фактически, CUDA позволяет включать в текст программы на C специальные функции. Эти функции пишутся на особом диалекте C, и выполняются на графическом процессоре.

CUDA даёт разработчику возможность по своему усмотрению организовывать доступ к набору инструкций графического ускорителя и управлять его памятью, организовывать на нём сложные параллельные вычисления. Графический ускоритель с поддержкой CUDA становится мощной программируемой открытой архитектурой, приближаясь к сегодняшним центральным процессорам.

Использует grid-модель памяти, кластерное моделирование потоков и SIMD инструкции. Применим в основном для высокопроизводительных графических вычислений и разработок NVIDIA-совместимого графического API. Включена возможность подключения к приложениям, использующим Microsoft Direct3D 9 и OpenGL. Создан в версиях для Linux, Mac OS X и Windows.

Первоначальная версия CUDA SDK была представлена 15 февраля 2007 года. В основе CUDA API лежит язык Си с некоторыми ограничениями. Для успешной трансляции кода на этом языке, в состав CUDA SDK входит собственный Си-компилятор командной строки nvcc компании Nvidia. Компилятор nvcc создан на основе открытого компилятора Open64 и предназначен для трансляции host-кода (главного, управляющего кода) и device-кода (аппаратного кода) (файлов с расширением .cu) в объектные файлы, пригодные в процессе сборки конечной программы или библиотеки в любой среде программирования, например в Microsoft Visual Studio.

6.2 Преимущества

По сравнению с традиционным подходом к организации вычислений общего назначения посредством возможностей графических API, у архитектуры CUDA отмечают следующие преимущества в этой области:

- Интерфейс программирования приложений CUDA (CUDA API) основан на стандартном языке программирования Си с некоторыми ограничениями. По мнению разработчиков, это должно упростить и сгладить процесс изучения архитектуры CUDA
- Разделяемая между потоками память (shared memory) размером в 16 Кб может быть использована под организованный пользователем кэш с более широкой полосой пропускания, чем при выборке из обычных текстур
- Более эффективные транзакции между памятью центрального процессора и видеопамятью
- Полная аппаратная поддержка целочисленных и побитовых операций

6.3 Ограничения

- Все функции, выполнимые на устройстве, не поддерживают рекурсии и имеют некоторые другие ограничения
- Архитектуру CUDA поддерживает и развивает только производитель NVidia

6.4 Пример

```
__global__ void kernel ( float * data )
{
   // номер текущей нити
   int   idx = blockIdx.x * blockDim.x + threadIdx.x;

   // значение аргумента
   float x   = 2.0f * 3.1415926f * (float) idx / (float) N;

   // найти значение и записать его в массив
   data [idx] = sinf ( sqrtf ( x ) );
}

float   a [N];
float * dev = NULL;

// выделить память на GPU под N элементов
cudaMalloc ( (void**)&dev, N * sizeof ( float ) );

// запустить N нитей блоками по 512 нитей
// выполняемая на нити функция - kernel
// массив данных - dev
kernel<<<dim3((N/512),1), dim3(512,1)>>> ( dev );

// скопировать результаты из памяти GPU (DRAM) в
// память CPU (N элементов)
cudaMemcpy ( a, dev, N * sizeof ( float ), cudaMemcpyDeviceToHost );

// освободить память GPU
cudaFree   ( dev );
```

6.5 Построение множества достижимости

```
#include <stdio.h>
#include <stdlib.h>
#include <cutil_inline.h>

#define SET_MATRIX(A, B, C, D) \
    __constant__ const float A00 = A; \
    __constant__ const float A01 = B; \
    __constant__ const float A10 = C; \
    __constant__ const float A11 = D;
#define SET_INITIAL(A, B) \
    __constant__ const float x0_0 = A; \
    __constant__ const float x0_1 = B;

__constant__ const float pi = 3.141592653589793;

const int tsteps = 100;
const int nalpha = 1024;
const int gridSize = 64;
__constant__ const float T = 3;
SET_MATRIX(0, 1, 0, 0);
SET_INITIAL(0, 0);

inline __device__ __host__ float2 gradC(float2 psi)
{
    return make_float2(0, psi.y > 0 ? 1 : -1);
```

```
}

inline __device__ __host__ float4 operator* (float a, float4 b)
{ return make_float4(a*b.x, a*b.y, a*b.z, a*b.w); }

inline __device__ __host__ float4 operator* (float4 b, float a)
{ return make_float4(a*b.x, a*b.y, a*b.z, a*b.w); }

inline __device__ __host__ float4 operator/ (float4 a, float b)
{ return make_float4(a.x/b, a.y/b, a.z/b, a.w/b); }

inline __device__ __host__ float4 operator+ (float4 a, float4 b)
{ return make_float4(a.x+b.x, a.y+b.y, a.z+b.z, a.w+b.w); }

inline __device__ __host__ float4 operator- (float4 a, float4 b)
{ return make_float4(a.x-b.x, a.y-b.y, a.z-b.z, a.w-b.w); }

inline __device__ __host__ float4 operator- (float4 a)
{ return make_float4(-a.x, -a.y, -a.z, -a.w); }

inline __device__ __host__ float4& operator*= (float4& a, float b)
{ a.x*=b; a.y*=b; a.z*=b; a.w*=b; return a; }

inline __device__ __host__ float4& operator/= (float4& a, float b)
{ a.x/=b; a.y/=b; a.z/=b; a.w/=b; return a; }

inline __device__ __host__ float4& operator+= (float4& a, float4 b)
{ a.x+=b.x; a.y+=b.y; a.z+=b.z; a.w+=b.w; return a; }

inline __device__ __host__ float4& operator-= (float4& a, float4 b)
{ a.x-=b.x; a.y-=b.y; a.z-=b.z; a.w-=b.w; return a; }

__device__ __host__ float4 f(float t, float4 y)
{
    float2 x = make_float2(y.x, y.y);
    float2 psi = make_float2(y.z, y.w);
    float2 grad = gradC(psi);
    return make_float4(
        A00*x.x + A01*x.y + grad.x,
        A10*x.x + A11*x.y + grad.y,
        -A00*psi.x - A10*psi.y,
        -A01*psi.x - A11*psi.y);
}

template <class Tx, class Ty>
__device__ __host__ Ty rk4(Tx x, Tx x1, Ty y, int steps)
{
    int i;
    Tx h = (x1-x)/steps;
    Tx h2 = h/2;
    Ty k1, k2, k3, k4;

    for (int i = 1; i <= steps; ++i)
    {
        k1 = h*f(x, y);
        x += h2;
        k2 = f(x, y+k1/2)*h;
        k3 = f(x, y+k2/2)*h;
        x += h2;
```

```
85          k4 = f(x, y+k3)*h;
86          y += (k1+2*(k2+k3)+k4)/6;
87      }
88      return y;
89  }
90
91  __global__ void kernel(float2* result)
92  {
93      int i = blockDim.x * blockIdx.x + threadIdx.x;
94      float alpha = (float)i / (float)nalpha * 2 * pi;
95      float4 y0 = make_float4(x0_0, x0_1, cos(alpha), sin(alpha));
96      float4 res = rk4(0.0f, T, y0, 100);
97      result[i] = make_float2(res.x, res.y);
98  }
99
100 int main(int argc, char* argv[])
101 {
102     if(!InitCUDA()) {
103         return 0;
104     }
105
106     float2* device_result = 0;
107     float2 host_result[nalpha];
108
109     cudaMalloc((void**)&device_result, sizeof(float2) * nalpha);
110
111     unsigned int timer = 0;
112     cutilCheckError( cutCreateTimer( &timer));
113     cutilCheckError( cutStartTimer( timer));
114
115     kernel<<<gridSize, nalpha/gridSize>>>(device_result);
116     cutilCheckMsg("Kernel execution failed\n");
117
118     cudaThreadSynchronize();
119     cutilCheckError( cutStopTimer( timer));
120     printf("Processing time: %f (ms)\n", cutGetTimerValue( timer));
121     cutilCheckError( cutDeleteTimer( timer));
122
123     cudaMemcpy(host_result, device_result, sizeof(float2) * nalpha, cudaMemcpyDeviceToHost);
124     cudaFree(device_result);
125
126     FILE* fp = fopen("C:\\alina\\output.txt", "w");
127     for (int i = 0; i < nalpha; ++i)
128         fprintf(fp, "%f %f\n", host_result[i].x, host_result[i].y);
129
130     return 0;
131 }
```

7 Результаты тестирования программ

Программы протестированы на четырех рассмотренных выше примерах на компьютере AMD Turion^{TM}X2 Dual-Core Mobile RM-70 2.00 GHz с 3.00 Gb RAM. Для программы на CUDA использовалась видеокарта NVidia GeForce 9800 GT. Результаты приведены в таблице.

Разница времени вычислений обусловлена эффективностью применения техно-

Таблица 1: Результаты тестирования

	Maple	MatLab	CUDA
Задача 1	1.0 с	3.8 с	$\ll 0.1$ с
Задача 2	1.7 с	4.3 с	$\ll 0.1$ с
Задача 3	1.2 с	3.2 с	$\ll 0.1$ с
Задача 4	1.9 с	4.5 с	$\ll 0.1$ с

логии CUDA для построения множеств достижимости.

8 Список литературы

[1] *Понтрягин Л.С., Болтянский В.Г., Гамкрелидзе Р.В., Мищенко Е.Ф.* Математическая теория оптимальных процессов: Учебное пособие. – М.: Наука, 1983.

[2] *Киселев Ю.Н., Аввакумов С.Н., Орлов М.В.* Оптимальное управление. Линейная теория и приложения: Учебное пособие. – М.: МАКС Пресс, 2007.

[3] NVIDIA CUDA - неграфические вычисления на графических процессорах http://www.ixbt.com/video3/cuda-1.shtml