

Фреймворк для интерактивного моделирования
задач «реакция-диффузия» на
вычислительных кластерах

Дмитрий Глызин
glyzin@gmail.com

Даниил Фролов
frolovd94@gmail.com

Ярославский государственный университет им. П.Г. Демидова

Таруса, 24-26 мая 2016 г.

Задача

- Инструмент для моделирования уравнений в частных производных (в том числе уравнений с запаздыванием)
- Распределение уравнений в одномерных, двумерных и трехмерных областях
- Поддержка современного оборудования — неоднородные архитектуры с иерархической организацией памяти
- Повышение уровня абстракции — автоматическая генерация шаблонного кода для пользовательских функций
- Интерфейс для управления вычислениями

Задача

- Метод Эйлера
- Метод Рунге-Кутты (4)
- Метод Дормана-Принса (45)
- Неявные схемы (для решения жестких задач)
- Алгоритмы анализа данных

Общий вид допустимой системы

$$\partial u(t, x, y, z) / \partial t = f(t, x, y, z, u, \partial u / \partial x, \partial^2 u / \partial x^2, \dots, \partial^4 u / \partial z^4),$$

где f нелинейная, а $u = (u_1, \dots, u_n)$.

Возможные граничные условия

- Условие Неймана
- Условие Дирихле
- Периодические условия

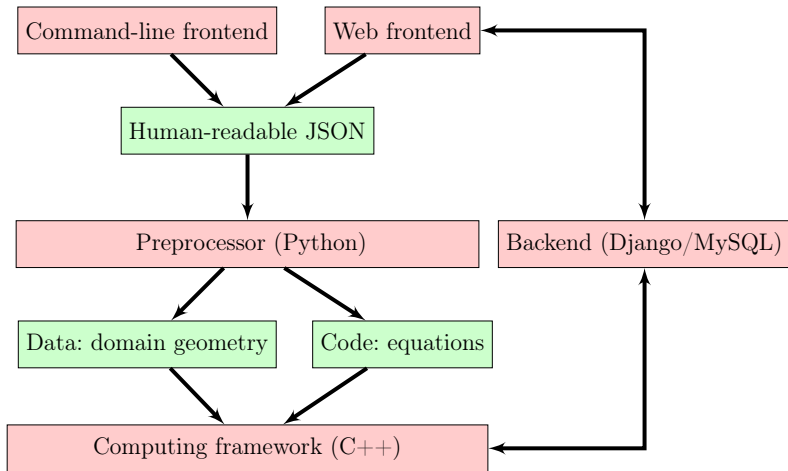
Структура пакета

- Пользовательский интерфейс
 - создание/модификации проекта
 - управление вычислениями
- Предварительная обработка
 - геометрия и свойства области
 - библиотека пользовательских функций
 - разбиение задачи на блоки и распределение блоков по вычислительным устройствам
- Вычислительное ядро
 - параллельный фреймворк
 - численные методы и алгоритмы анализа данных

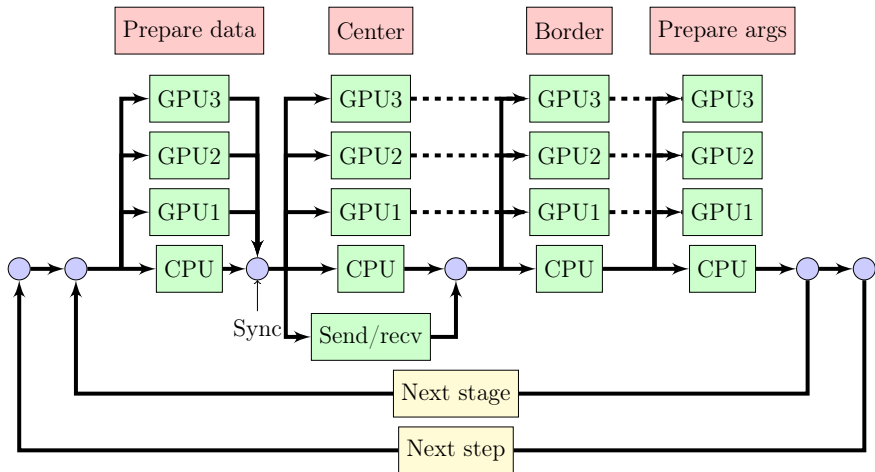
Вычислительное ядро

- Параллельный фреймворк
 - крупнозернистый параллелизм: разбиение задачи на блоки и распределение блоков по устройствам
 - пересылка необходимых данных между устройствами на разных узлах с помощью MPI
 - обмен данными через pinned-память между устройствами на одном узле
 - мелкозернистый параллелизм в рамках блока: работа с общей памятью, OpenMP на CPU и CUDA на GPU
 - использование арифметики двойной точности: CPU дает значительный вклад в общую производительность

Схема работы



Параллельная обработка



Web-интерфейс

Tracer Web UI Tasks Trash Cluster

Info Edit

Domain

ProjectName
Like in Hairer and Wanner
Name of the project

Equations ▾ + Equation

0 - Flat Brusselator

0 - Flat Brusselator ▾

Name
Flat Brusselator

Independent variables ▾

Variable
x
+ Variable

List of equations ▾


Equation		
$U' = a + U^2 V - (b+1)U + cD[U, [$	✕	↓
$V' = b - U - UV + cD[V, [x, 2]]$	✕	↑





+ Equation ✕ Last Equation ✕ All

Parameters >


Web-интерфейс

Tracer Web UI Tasks Trash Cluster

Create 

Task title	Job started	Job finished	Job state	Finished	Actions
newformat			New		   

Tracer Web UI Tasks Trash Cluster



Cluster info

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
debug	up	10:00	3	alloc	cnode[3-5]
debug	up	10:00	4	idle	cnode[1-2,6-7]
main*	up	infinite	3	alloc	cnode[3-5]
main*	up	infinite	2	idle	cnode[6-7]

Cluster queue info

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
18232	main	mpirun	ea_guzov	R	3:21:20:43	1	cnode3
18922	main	mpirun	ea_guzov	R	1:10:52:03	1	cnode5
18991	main	mpirun	ea_guzov	R	1:30:16	1	cnode4

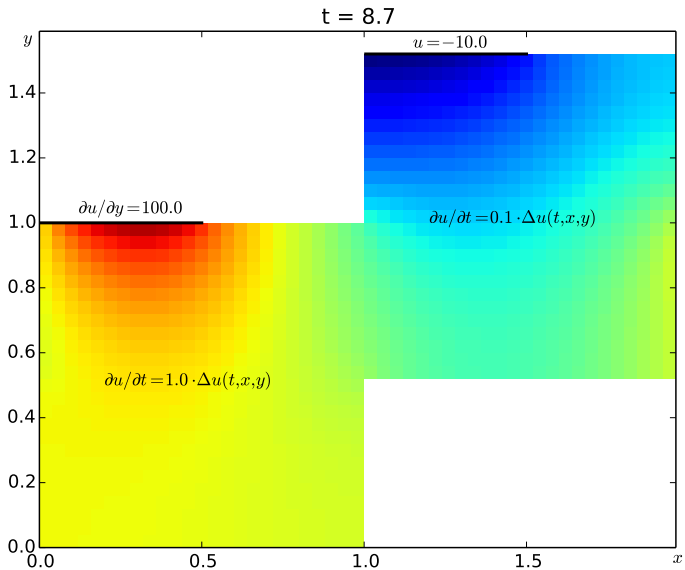
main*

CNODE	ENABLE
cnode5	●
cnode4	●
cnode7	●
cnode6	●
cnode3	●

debug

CNODE	ENABLE
cnode5	●
cnode4	●
cnode7	●
cnode6	●
cnode1	●
cnode3	●
cnode2	●

Пример



Генератор

```
"Equations": [
  {
    "Name": "ID heat 1",
    "Vars": [
      "x",
      "y"
    ],
    "System": [
      "U' = a + (0[U,{x,2}] + 0[U,{y,2}])"
    ]
  },
  {
    "Name": "ID heat 1",
    "Vars": [
      "x",
      "y"
    ],
    "System": [
      "U' = b + (0[U,{x,2}] + 0[U,{y,2}])"
    ]
  }
],
```

```
"Bounds": [
  {
    "Name": "N 1",
    "Type": 1,
    "Values": [
      "100.0"
    ]
  },
  {
    "Name": "D 1",
    "Type": 1,
    "Values": [
      "-50.0"
    ],
    "Derivative": [
      "0.0"
    ]
  }
],
```

```
"Interconnects": [
  {
    "Name": "connection 1",
    "Block1": 0,
    "Block2": 1,
    "Block1Side": 1,
    "Block2Side": 0
  },
  {
    "Name": "connection 1",
    "Block1": 0,
    "Block2": 1,
    "Block1Side": 0,
    "Block2Side": 1
  }
],
```

Генератор

```
//=====CENTRAL FUNCTIONS FOR BLOCK WITH NUMBER 0=====//

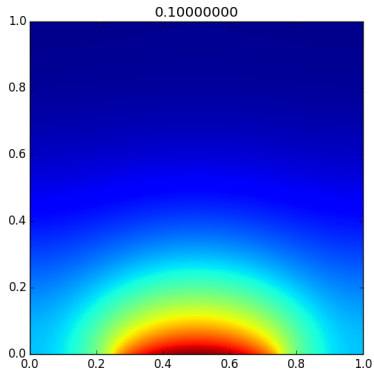
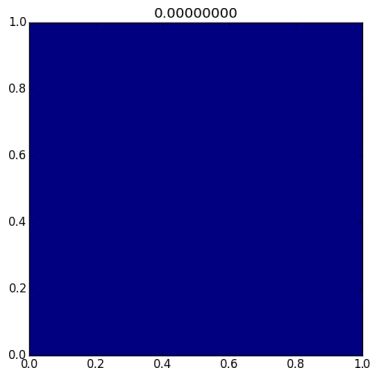
//0 central function for 2d model for block with number 0
void Block0CentralFunction0(double* result, double* source, double t, int idxX, int idxY, int idxZ, double* params, double** ic){
    int idx = ( idxX + idxY * Block0StrideY + idxZ * Block0StrideZ ) * Block0CELLSIZE;
    result[idx + 0] = params[0] + ((DXM2 * (1 + source[idx + 1 * Block0StrideX + Block0CELLSIZE + 0]) - 2 * source[idx + 0 * Block0StrideX + Block0CELLSI
})

//=====BOUNDARY CONDITIONS FOR BLOCK WITH NUMBER 0=====//

//Default boundary condition for Vertex between boundaries y = 0 and x = 0
void Block0DefaultNeumann_Vertex2_0_Eqn0(double* result, double* source, double t, int idxX, int idxY, int idxZ, double* params, double** ic){
    int idx = ( idxX + idxY * Block0StrideY + idxZ * Block0StrideZ ) * Block0CELLSIZE;
    result[idx + 0] = params[0] + ((2.0 * DXM2 * (source[idx + Block0StrideX + Block0CELLSIZE + 0] - source[idx + 0] - (0.0) * DX)) + (2.0 * DYM2 * (so
})
//Default boundary condition for Vertex between boundaries y = 0 and x = x_max
void Block0DefaultNeumann_Vertex2_1_Eqn0(double* result, double* source, double t, int idxX, int idxY, int idxZ, double* params, double** ic){
    int idx = ( idxX + idxY * Block0StrideY + idxZ * Block0StrideZ ) * Block0CELLSIZE;
    result[idx + 0] = params[0] + ((2.0 * DXM2 * (source[idx - Block0StrideX - Block0CELLSIZE + 0] - source[idx + 0] + (0.0) * DX)) + (2.0 * DYM2 * (so
})
//Non-default boundary condition and interconnect for Vertex between boundaries y = y_max and x = 0
void Block0NeumannAndInterconnect_Vertex3_0_Eqn0(double* result, double* source, double t, int idxX, int idxY, int idxZ, double* params, double** ic){
    int idx = ( idxX + idxY * Block0StrideY + idxZ * Block0StrideZ ) * Block0CELLSIZE;
    result[idx + 0] = params[0] + ((DXM2 * (source[idx + Block0StrideX + Block0CELLSIZE + 0] - 2.0 * source[idx + 0] + ic[1][(idxY - 50) + Block0CELLSI
})
//Default boundary condition and interconnect for Vertex between boundaries y = y_max and x = x_max
void Block0DefaultNeumannAndInterconnect_Vertex3_1_Eqn0(double* result, double* source, double t, int idxX, int idxY, int idxZ, double* params, double** ic){
    int idx = ( idxX + idxY * Block0StrideY + idxZ * Block0StrideZ ) * Block0CELLSIZE;
    result[idx + 0] = params[0] + ((DXM2 * (ic[0][(idxY - 50) * Block0CELLSIZE + 0] - 2.0 * source[idx + 0] + source[idx - Block0StrideX + Block0CELLSI
})
//Boundary condition for boundary y = 0
void Block0DefaultNeumann_Bound2_Eqn0(double* result, double* source, double t, int idxX, int idxY, int idxZ, double* params, double** ic){
    int idx = ( idxX + idxY * Block0StrideY + idxZ * Block0StrideZ ) * Block0CELLSIZE;
    result[idx + 0] = params[0] + ((DXM2 * (1 + source[idx + 1 * Block0StrideX + Block0CELLSIZE + 0]) - 2 * source[idx + 0 * Block0StrideX + Block0CELLSI
})
//Boundary condition for boundary y = y_max
void Block0Neumann_Bound3_0_Eqn0(double* result, double* source, double t, int idxX, int idxY, int idxZ, double* params, double** ic){
    int idx = ( idxX + idxY * Block0StrideY + idxZ * Block0StrideZ ) * Block0CELLSIZE;
    result[idx + 0] = params[0] + ((DXM2 * (1 + source[idx + 1 * Block0StrideX + Block0CELLSIZE + 0]) - 2 * source[idx + 0 * Block0StrideX + Block0CELLSI
})
//Boundary condition for boundary y = y_max
void Block0DefaultNeumann_Bound3_Eqn0(double* result, double* source, double t, int idxX, int idxY, int idxZ, double* params, double** ic){
    int idx = ( idxX + idxY * Block0StrideY + idxZ * Block0StrideZ ) * Block0CELLSIZE;
    result[idx + 0] = params[0] + ((DXM2 * (1 + source[idx + 1 * Block0StrideX + Block0CELLSIZE + 0]) - 2 * source[idx + 0 * Block0StrideX + Block0CELLSI
})
//Boundary condition for boundary x = 0
void Block0DefaultNeumann_Bound0_Eqn0(double* result, double* source, double t, int idxX, int idxY, int idxZ, double* params, double** ic){
    int idx = ( idxX + idxY * Block0StrideY + idxZ * Block0StrideZ ) * Block0CELLSIZE;
    result[idx + 0] = params[0] + ((2.0 * DXM2 * (source[idx + Block0StrideX + Block0CELLSIZE + 0] - source[idx + 0] - (0.0) * DX)) + (DYM2 * (1 + sour
})
```

Тестовая задача

- Уравнение теплопроводности
- Метод Эйлера

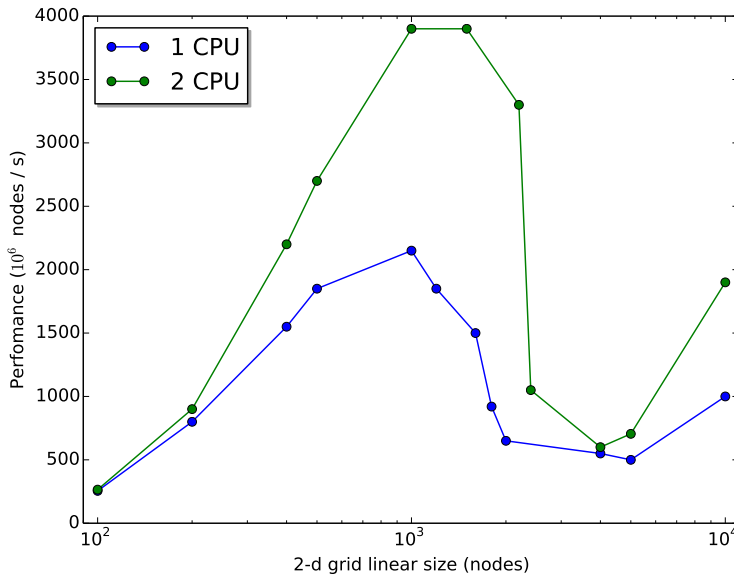


Технические характеристики

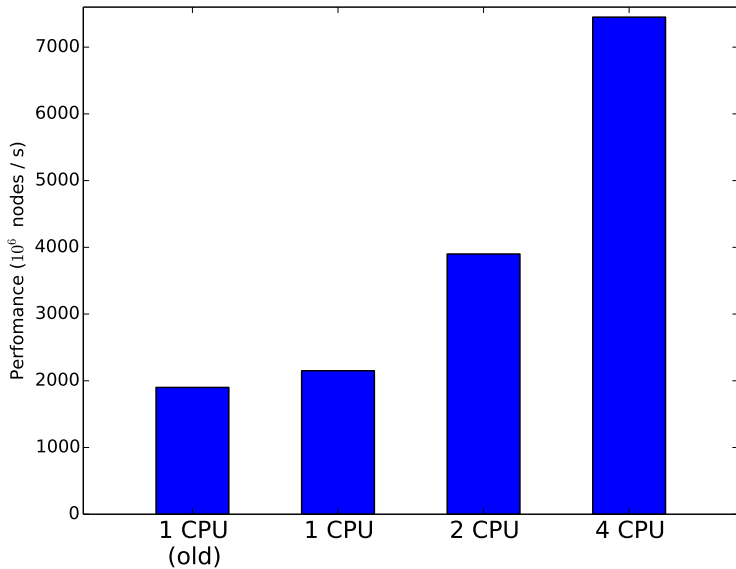
- Оборудование: 2 узла: 2xCPU E5-2690 (8 ядер, 2.9ГГц), 3xGPU Tesla M2090, Infiniband QDR 40Gbps
- ПО: SLES 12, gcc 4.8, Mellanox OFED 2.4, OpenMPI 1.8, CUDA Toolkit 7.0, SLURM 14.11

В дальнейшем 2 центральных процессора (16 ядер) в рамках одного узла используются как единое вычислительное устройство, обозначенное CPU.

CPU, 1 и 2 узла



Итоговое сравнение



Дальнейшие планы

- реализация генератора для GPU
- решение уравнений с запаздыванием
- алгоритмы анализа данных
- доработка веб-интерфейса
- добавление неявных схем