Пакеты со встроенной поддержкой GPU — Trilinos/PetSc

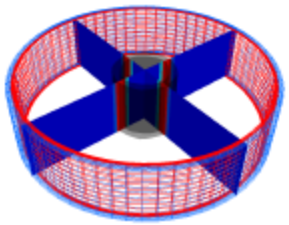
Trilinos

- Разработчик – SNL
- Домашний адрес http://trilinos.sandia.gov
- Исторически появился 1999 году, как продолжение AztecOO (Майкл Хэрокс)
- Задачи: изначально линейная алгебра, сейчас – множество различных численных методов (ОДУ, УРЧП, оптимизация и др.)

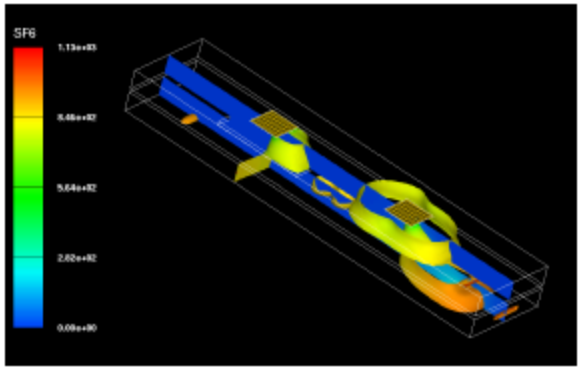
Особенности

- Состоит из большого количества пакетов (сейчас порядка 50), пакеты сравнительно независимы
- Язык – C++, хотя некоторые части – C, Fortran
- Активно используются средства ООП, темплейты
- Большое количество TPL (Blas, Lapack, Scalapack, CUDA, OpenCL, MPI, Boost, cublas, cusparse, cusp, Scotch, Metis/Parmetis, много других): в некотором смысле является не библиотекой, а попыткой “навести мосты” среди существующего ПО
- Лицензия – LGPL/BSD
- Много интерфейсов в другим языкам и системам (Fortran, Python, Matlab, etc)
- Сильно недоделан, мало документации

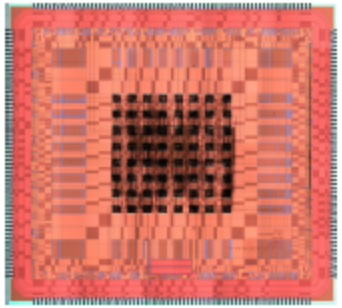
Applications

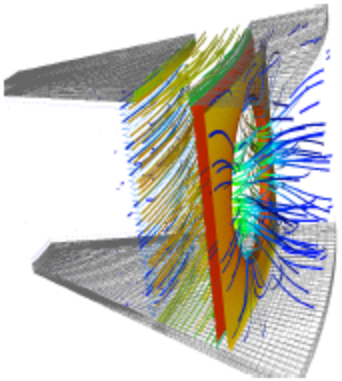


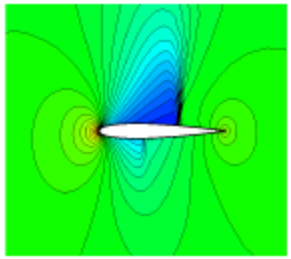
PDEs



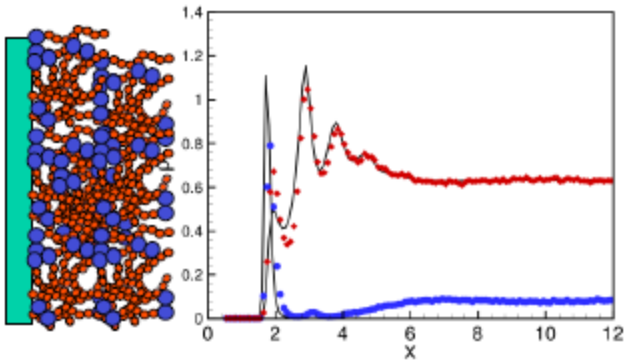
Circuits







Inhomogeneous Fluids



And More…

Пакеты

Сервисы	Объекты лин.алг.	**Epetra, Tpetra, Kokkos**
	Интерфейсы	**Thyra, Stratimikos, RTOp, FEI, Shards, Tpetra::RTI**
	Баланс графов	**Zoltan, Isorropia**
	Внешние интерфейсы	**PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos, Optika**
	Утилиты	**Teuchos, EpetraExt, Kokkos, Triutils, ThreadPool,**
Дискретизация	Работа с сеткой, пространственная дискр.	**STKMesh, Intrepid**
	Решение ОДУ	**Rythmos**
Другое	AD	**Sacado**

Пакеты

Солверы	Итерационные методы	**AztecOO, Belos, Komplex**
	Прямые методы (разреж.)	**Amesos, Amesos2**
	Прямые методы (плотн.)	**Epetra, Teuchos, Pliris**
	Собственные значения	**Anasazi**
	ML предобуславливатель	**ML**
	ILU предобуславливатели	**AztecOO, IFPACK, Ifpack2**
	Нелинейные системы	**NOX, LOCA**
	Оптимизация	**TriKota, Globipack, Optipack**
	Стохастические уравнения	**Stokhos**
	Блочные предобуславливатели	**Meros, Teko**

Примеры построения методов

Решение уравнения теплопроводности

$$\frac{\partial u}{\partial t} = \Delta u$$

Примеры построения методов

Решение уравнения гидродинамики

$$\nabla(u \otimes u) - \nabla p = 0$$

$$\nabla u = 0$$

Interpid, FEI, Shards — построение матрицы жесткости

Epetra::Operator — представление дискретногонелинейного уравнения

AztecOO — решение СЛАУ Якобиана

ML — предобуславливатель

NOX — решение

Globipack — оптимизация параметров течения

Утилиты

Указатели со счетчиком:

Teuchos::RCP<T> t = Teuchos::rcp<T>(new T(…));

Списки параметров:

Teuchos::ParameterList p;
p.set(“Solver”, “GMRES”);
p.set(“Tolerance”, 1.0e-4);
p.set(“Max Iterations”, 100);
…
Double tol = p.get<double>(“Tolerance”);

```
<ParameterList name="Ifpack">
  <Parameter name="Prec Type" type="string" value="ILU"/>
  <Parameter name="Overlap" type="int" value="0"/>
  <ParameterList name="Ifpack Settings">
  <Parameter name="fact: level-of-fill" type="int" value="0"/>
</ParameterList>
```

Утилиты

```
Teuchos::CommandLineProcessor cmdp(false,true);
cmdp.setOption("verbose","quiet",&verbose,"Print messages
and results.");
cmdp.setOption("printMatrix","noPrintMatrix",&printMatrix,"Print
the full matrix after reading it.");
if (cmdp.parse(argc,argv) !=
Teuchos::CommandLineProcessor::PARSE_SUCCESSFUL) {
   return -1;
}
```

Подробнее о лин. алгебре

Основные понятия:

Вектор (алгебраический, распеределнный, поддерживает операции лин. алг.) – напр. Tpetra::Vector

Мультивектор (несколько векторов)

Tpetra::MultiVector

Оператор (преобразование вектора в вектор)

Tpetra::Operator

Разреженная матрица

Tpetra::CrsMatrix

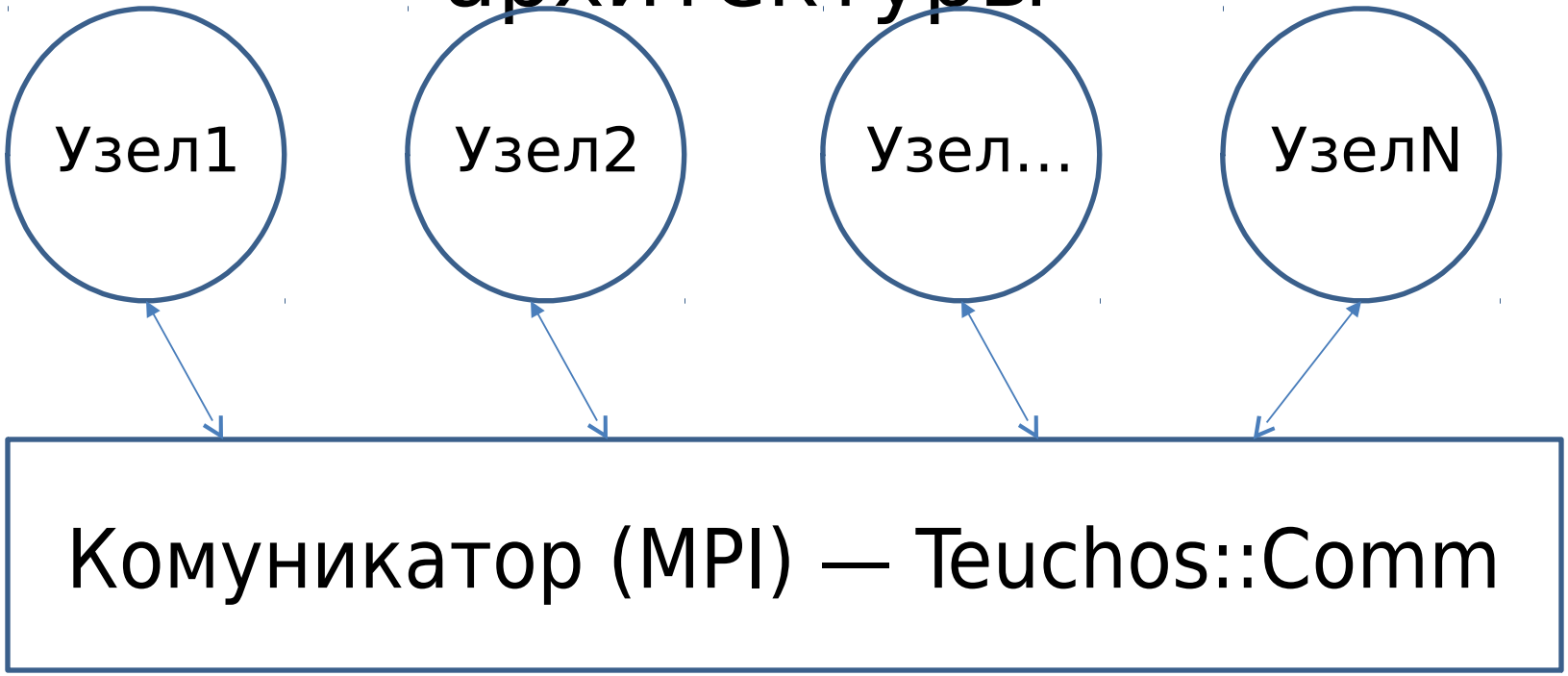
Подробнее о лин. алгебре

- Epetra (не поддерживает CUDA (пока))
- **Tpetra** (“templated”)
  - Темплейтится от многих вещей

```
Template<class Scalar, class LocalOrdinal,
class GlobalOrdinal, class Node >
class MultiVector
{
...
};
```

  - Пытается абстрагироваться от вычислительной платформы с помошью пакета Kokkos
- Thyra — абстрактные интерфейсы

Kokkos – представление архитектуры



Узел:

- Kokkos::SerialNode (последоват. узел)
- Kokkos::TBBNode (Thread Building Blocks)
- Kokkos::OpenMPNode
- Kokkos::ThrustGPUNode (CUDA)

Kokkos – представление архитектуры

Функции узла:

- Работа с памятью

ArrayRCP<T> allocBuffer(size_t size);

void copyFromBuffer(size_t size, const ArrayRCP<const T> &buffSrc, const ArrayView<T> &hostDest);

...

- parallel_for<functor>(int start, int end, functor f)
- Локальная линейная алгебра (локальные векторы, операторы, т.д.) например – CUBLAS, CUSP
- sync()

Пример создания вектора

Создание объекта узла, получение ранга узла:

```
Platform &platform = Tpetra::DefaultPlatform::getDefaultPlatform();
RCP<const Teuchos::Comm<int> > comm = platform.getComm();
RCP<Node>                      node = platform.getNode();
const int myRank = comm->getRank();
```

Создание объекта map:

```
RCP<const Map> map =
Tpetra::createUniformContigMap<Ordinal,Ordinal>(numGlobalElement
s, comm);
```

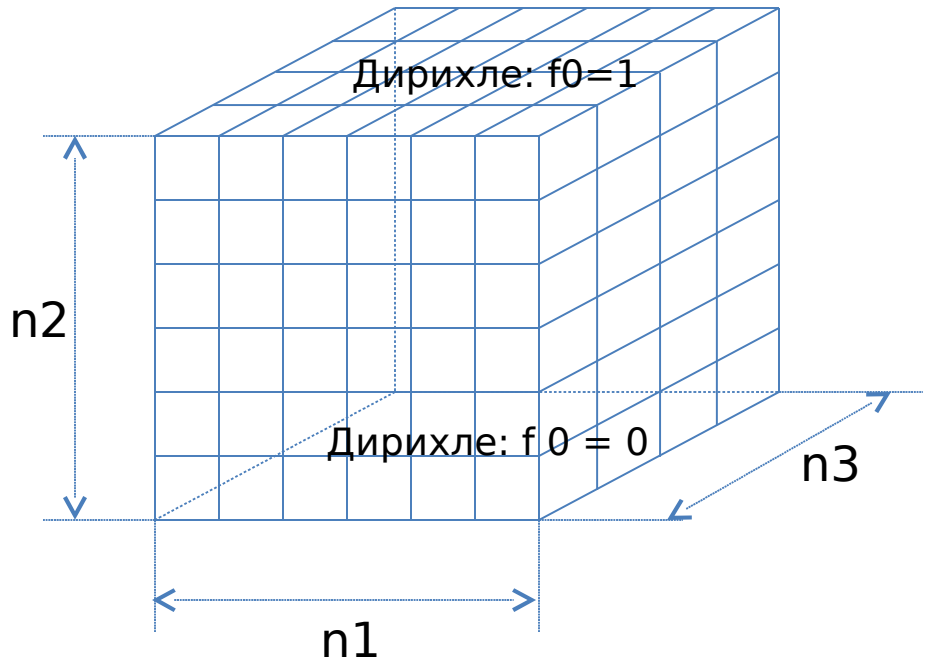
Создание вектора:

```
RCP<Tpetra::Vector> v = Tpetra::createVector<Scalar>(map);
```

Объект Map — распределение элементов (вектора) по узлам

Multi GPU: тестовая задача

Дискретное уравнение Пуассона (Лапласа) на прямоугольной сетке



$$\Delta f = 0 \quad \text{в} \quad \Omega \qquad f = f_0 \quad \text{в} \quad \Omega_0$$

$$\frac{\partial f}{\partial n} = 0 \quad \text{в} \quad \Omega_1$$

- Г.у. — с двух сторон Дирихле, остальные — Нейман
- Размер сетки n1×n2×n3
- Разбиение между gpu — вдоль n3
- Объем передаваемых данных ≈ n1×n2

С т.з. производительности рассматривались 2 задачи (примерно одинаковый объем вычислений) :

- 200 итераций на сетке 128×128×128
- 100 итераций на сетке 128×128×256 (меньше удельная доля передаваемых данных)

Multi GPU: тестовая задача

Реализации:

Пакет Trilinos (библиотеки Tpetra, Belos)

- Conjugate Gradients
- Реализация Пуассона (но мы брали правую часть = 0)
- Явно задается разреженная матрица
- Выполняется проверка сходимости

Реализация «вручную»

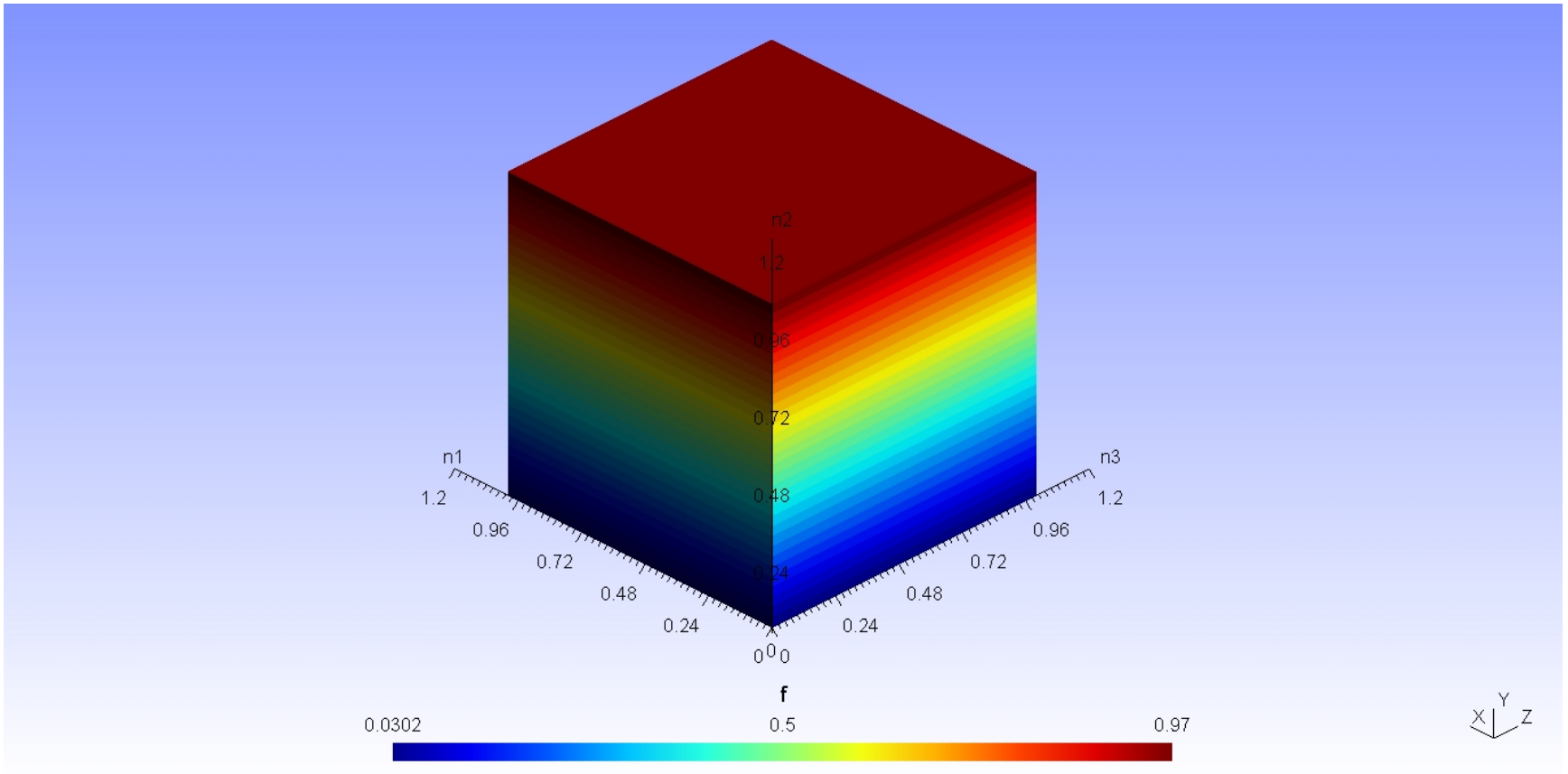
- Простая итерация
- Чистый Лаплас
- Matrix free (матрица «имеется в виду»)
- Нет проверки сходимости

В обоих случаях для реализации multiGPU используется комбинация mpi + CUDA

Поскольку методы не идентичны, то сравнение методов между собой (в пересчете на одинаковое число итераций) не является вполне корректным. Основная задача – показать и сравнить эффект от использования multiGPU.

Multi GPU: тестовая задача

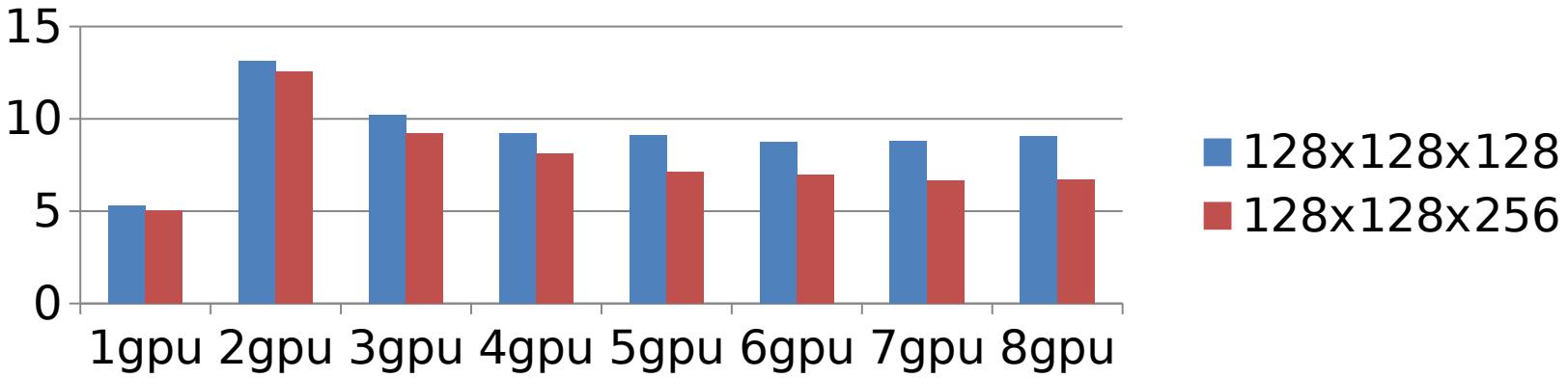
Пример решения:



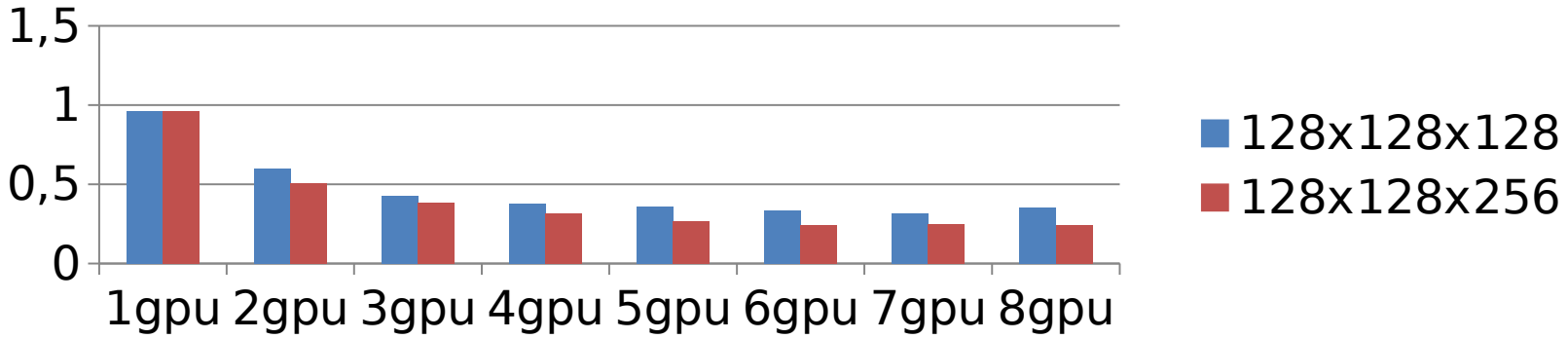
Multi GPU: тестовая задача

Производительность(время в сек):

Пакет Trilinos



Реализация «вручную»



Multi GPU: тестовая задача

Выводы:

- Текущая реализация Trilinos не дает **никакого** ускорения относительно одного GPU
- Это связано с тем, что в ней **все** данные копируются между cpu и gpu (cudaMemcpy)
- При реализации «вручную» можно добиться некоторого разумного ускорения
- Подход имеет очень сложный эффект масштаба:
  - С одной стороны, нужно увеличивать размер задачи, чтобы лучше загрузить gpu и уменьшить относительное время на передачу (как на обычных кластерах)
  - С другой стороны, увеличение размера данных ведет к увеличению объема передачи данных между cpu и gpu (на кластерах основная роль — latency)
- Реализация trilinos на одном gpu кажется приемлемой, однако нет смысла ее использовать, поскольку на данный момент отсутствует multigrid
- Сложно объяснить, почему в trilinos при уменьшении «сечения» произошло небольшое увеличение производительности (видимо, это связано просто с эффектом увеличения задачи)
- В собственной реализации это произошло из-за уменьшения относительного объема передаваемых данных (нужно стараться сделать его меньше)

Petsc

- Разработчик — Aragon
- Домашний адрес — http://www.mcs.anl.gov/petsc/
- Задачи: линейная алгебра, нелинейные солверы, интегрирование ОДУ
- В основном — язык C, но тем не менее, удалось добиться некоторой архитектурной независимости кода

Показать примеры