Распределение вычислительной нагрузки между GPU и CPU в системе LuNA

Докладчик: Беляев Н. А.

Введение

- При решении больших задач численного моделирования на суперкомпьютерах существенный прирост производительности можно получить, задействовав в вычислениях спец. Вычислители (GPU, Xeon Phi, FPGA,...)
- Использование спец. вычислителей требует от разработчика параллельной программы (ПП) знаний в области системного программирования

Обзор существующих решений

OpenCL

- Открытый стандарт параллельного программирования
- Программа представляется в виде множества задач, записанных на встроенном си-подобном языке (kernel)
- Взаимодействие между задачами, а также, управление памятью ложится на плечи разработчика ПП
- Программирование коммуникаций в распределенной памяти ложится на плечи разработчика ПП

OpenACC

- Предоставляет набор директив компилятора для работы с GPU (по аналогии с OpenMP)
- Автоматически генерирует код для GPU
- Не поддерживается работа в распределенной памяти
- Не поддерживается Xeon Phi/ FPGA

Charm++

- ПП представляется в виде множества взаимодействующих объектов (чаров)
- Чары взаимодействуют друг с другом с помощью механизма RPC
- Поддерживается GPU
- Поддерживается работа в распределенной памяти, обеспечивается динамическая балансировка нагрузки между узлами мультикомпьютера

Charm++

- Представление численного алгоритма в виде множества взаимодействующих объектов не всегда удобно и очевидно
- Не поддерживается работа с FPGA

DVMH

- Предоставляет набор директив компилятора для распараллеливания
- Работает в распределенной памяти
- Поддерживает работу с GPU, Xeon Phi
- Не поддерживает FPGA

LuNA

Общие сведения

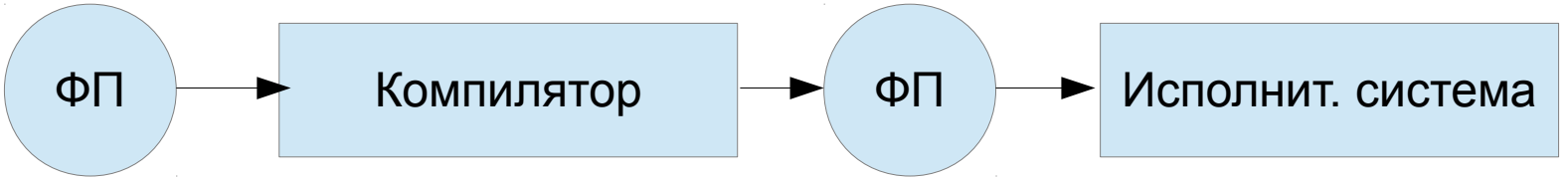
- Система LuNA – это система параллельного программирования, ориентированная на решение задач численного моделирования на суперкомпьютерах

Параллельная программа в системе LuNA

- ПП в системе LuNA представляется я виде множества фрагментов вычисления (ФВ) и фрагментов данных (ФД)
- Также ПП в системе LuNA может быть представлена в виде двудольного ориентированного графа,

Архитектура системы LuNA

- Система состоит из компилятора языка LuNA и исполнительной системы



Пример ФП

```
import cf_init(name #val) as init;
import cf_cuda1(value #in, name #out) as cuda1;
import cf_cuda2(value #in, name #out) as cuda2;
import cf_print(int, value #in) as print;

##const SZ 50

sub main()
{
	df data, data1, data2, data3;

	for i = 0 .. $SZ -1
		cf ini[i] : init(data[i]);

	for j = 0 .. $SZ - 1  {
		cf cu1[j] : cuda1(data[j], data1[j])-->(data[j]);

		cf cu2[j] : cuda2(data1[j], data2[j])-->(data1[j]);

	}

	for l = 0 .. $SZ - 1
		cf prt[l] : print(l, data2[l]);//-->(data1[l]);
}
```

Интерфейс программиста для использования GPU

```
import cf_init(name #val) as init;
import cf_cuda1(value #in, name #out) as cuda1:CUDA;
import cf_cuda2(value #in, name #out) as cuda2:CUDA;
import cf_print(int, value #in) as print;

##const SZ 50

sub main()
{
	df data, data1, data2, data3;

	for i = 0 .. $SZ -1
		cf ini[i] : init(data[i]);

	for j = 0 .. $SZ - 1 {
		cf cu1[j] : cuda1(data[j], data1[j])-->(data[j]);

		cf cu2[j] : cuda2(data1[j], data2[j])-->(data1[j]);

	}

	for l = 0 .. $SZ - 1
		cf prt[l] : print(l, data2[l]);//-->(data1[l]);
}
```

Алгоритмы распределения нагрузки

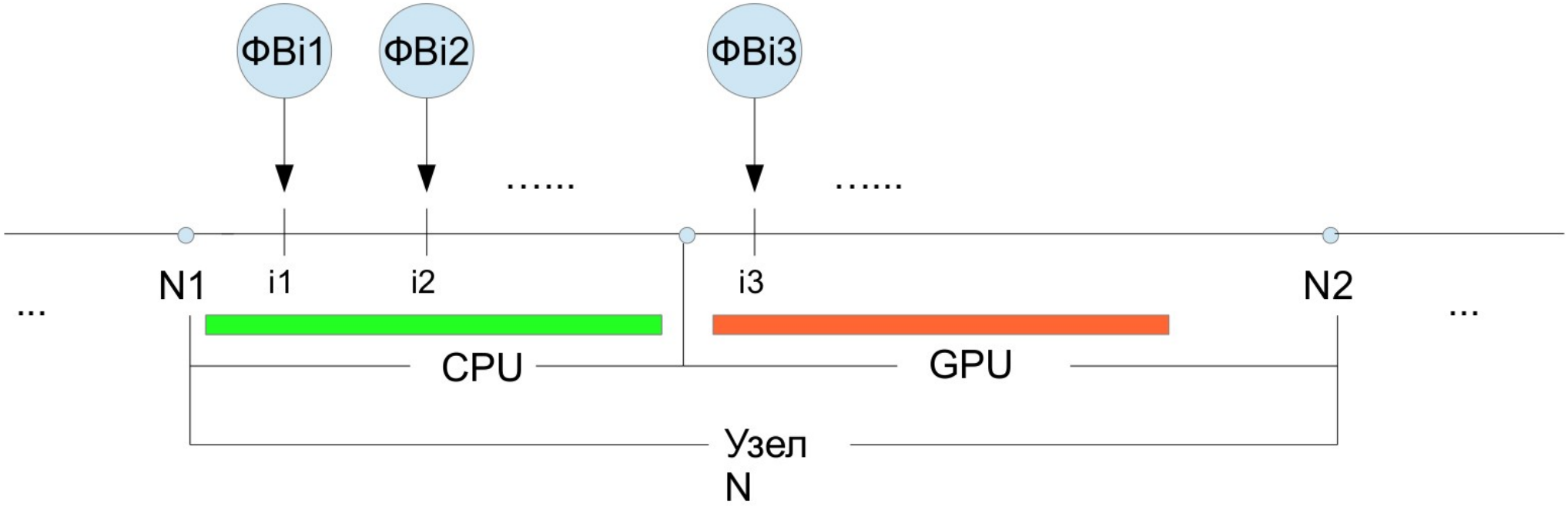
Задача распределения нагрузки

- Для каждого атомарного ФВ во время исполнения фрагментированной программы необходимо принять решение о выборе вычислителя (CPU или GPU) для его исполнения

Наивный алгоритм

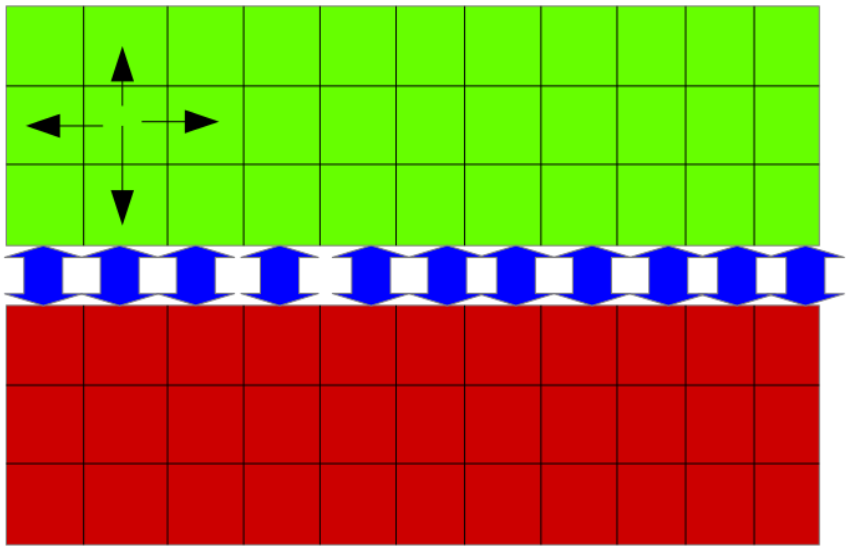
- Для каждого вновь поступающего на исполнение ФВ устройство выбирается случайно с вероятностью 0.5
- Перед исполнением ФВ на GPU значения входных ФД копируются на GPU
- После исполнения ФВ на GPU значения выходных ФВ копируются в оперативную память

RoB

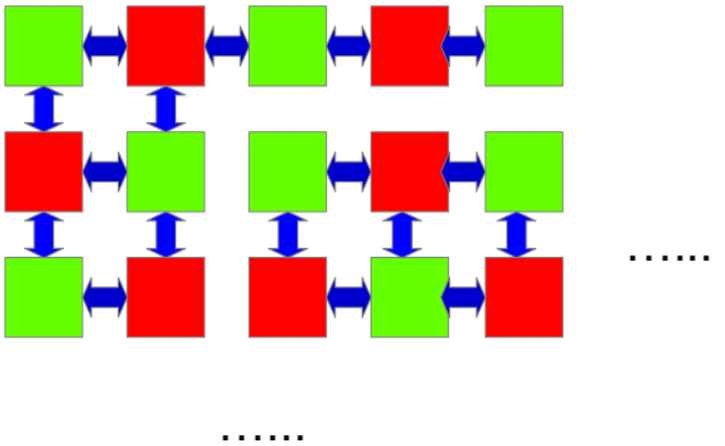


RoB

Алгоритм RoB



Простой алгоритм

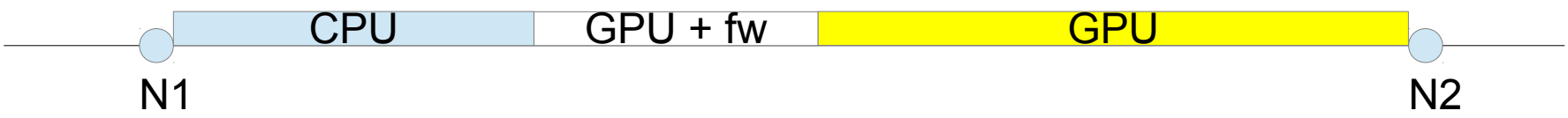


ФВ, выполняющийся на GPU

ФВ, выполняющийся на CPU

Передача данных между GPU и CPU

RoB

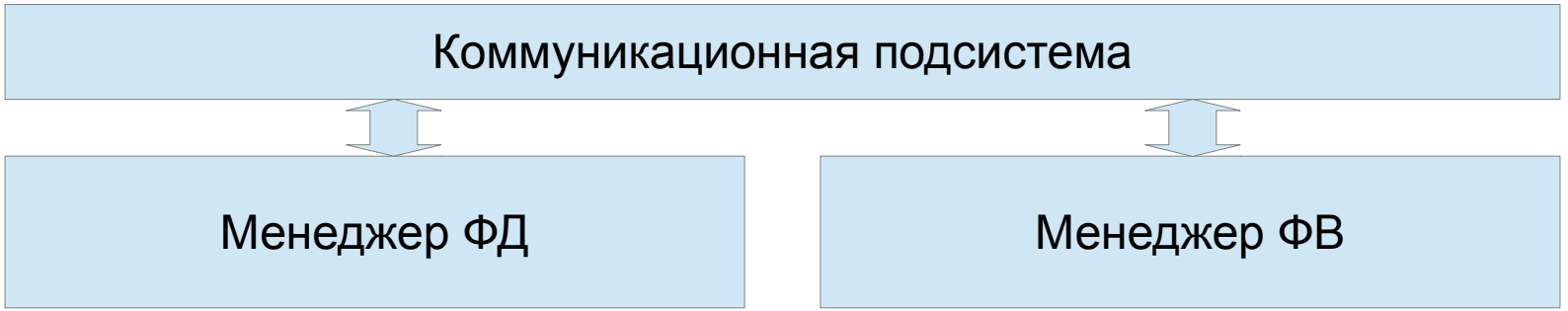


Выбор фрагментации (рефрагментация)

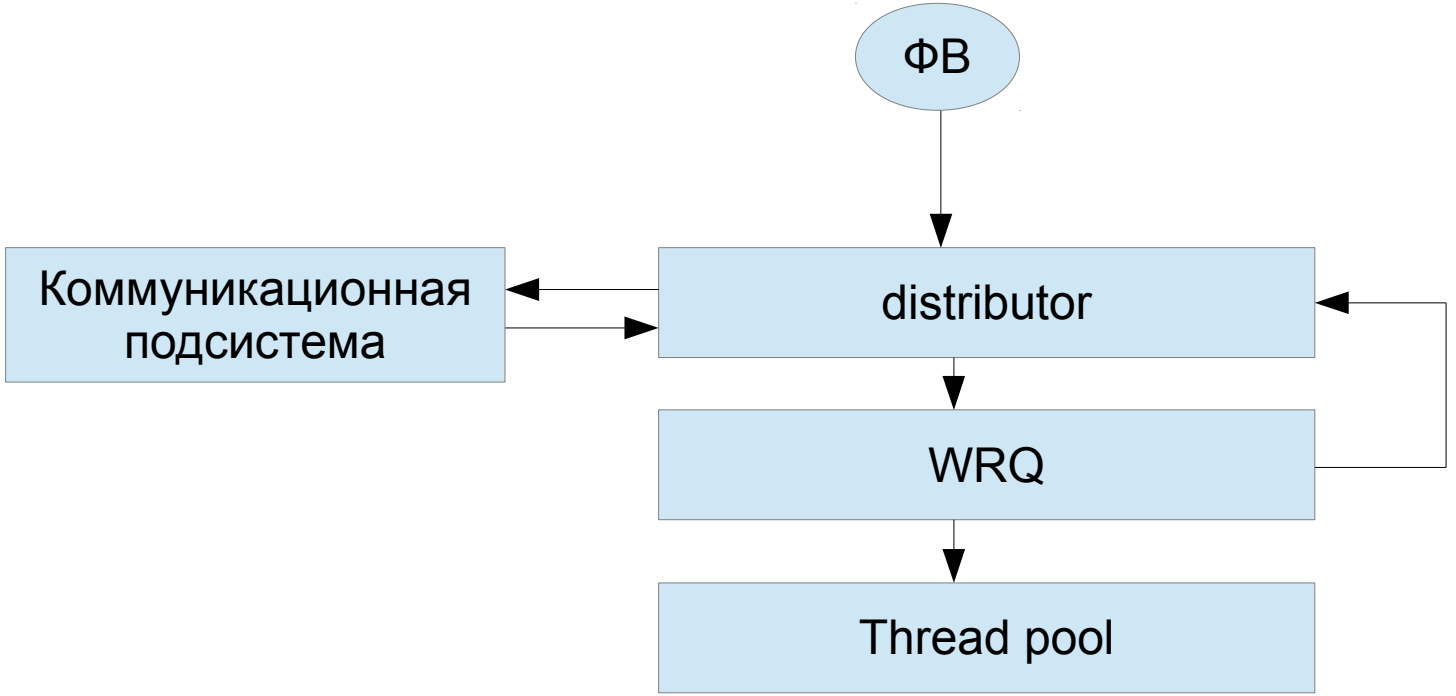
- Исходная фрагментация задачи изменяется системой путем изменения границ циклов и размеров ФД, на основании рекомендаций пользователя

Архитектура

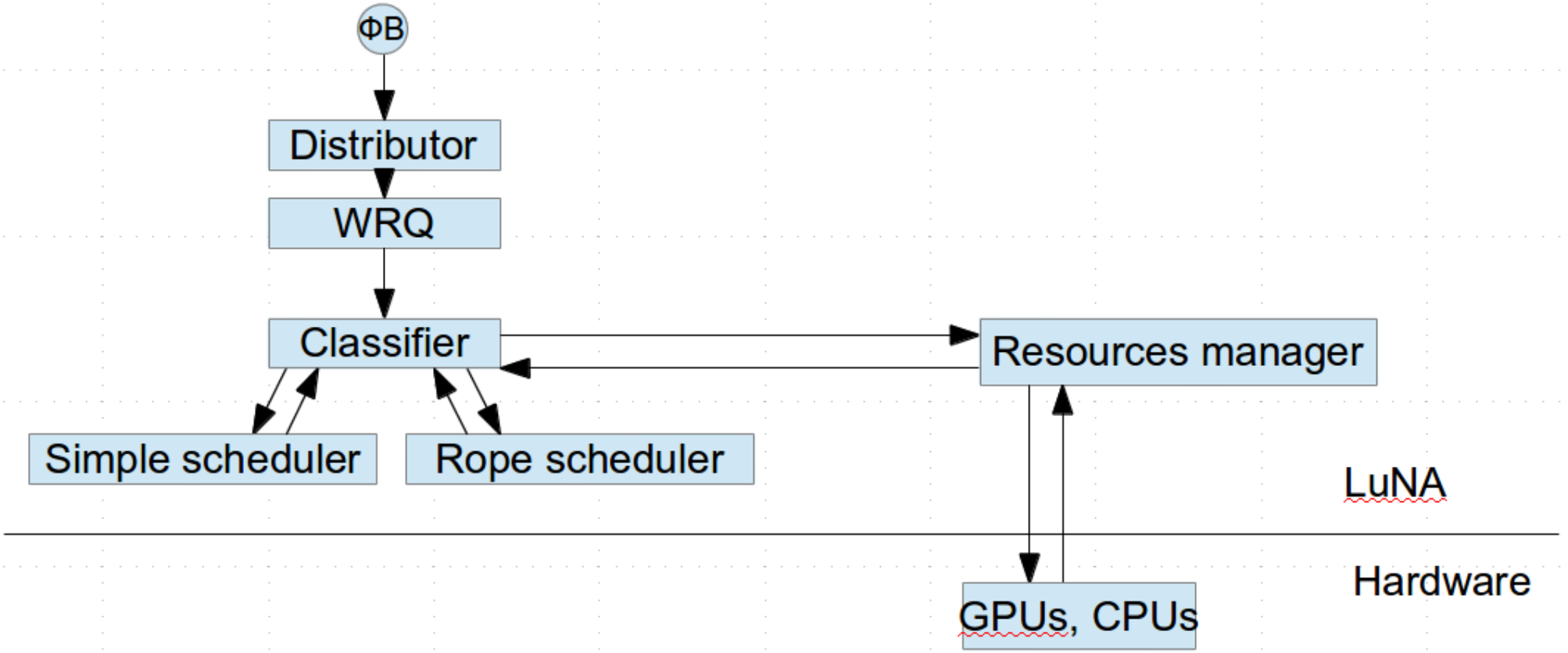
Архитектура исполнительной системы



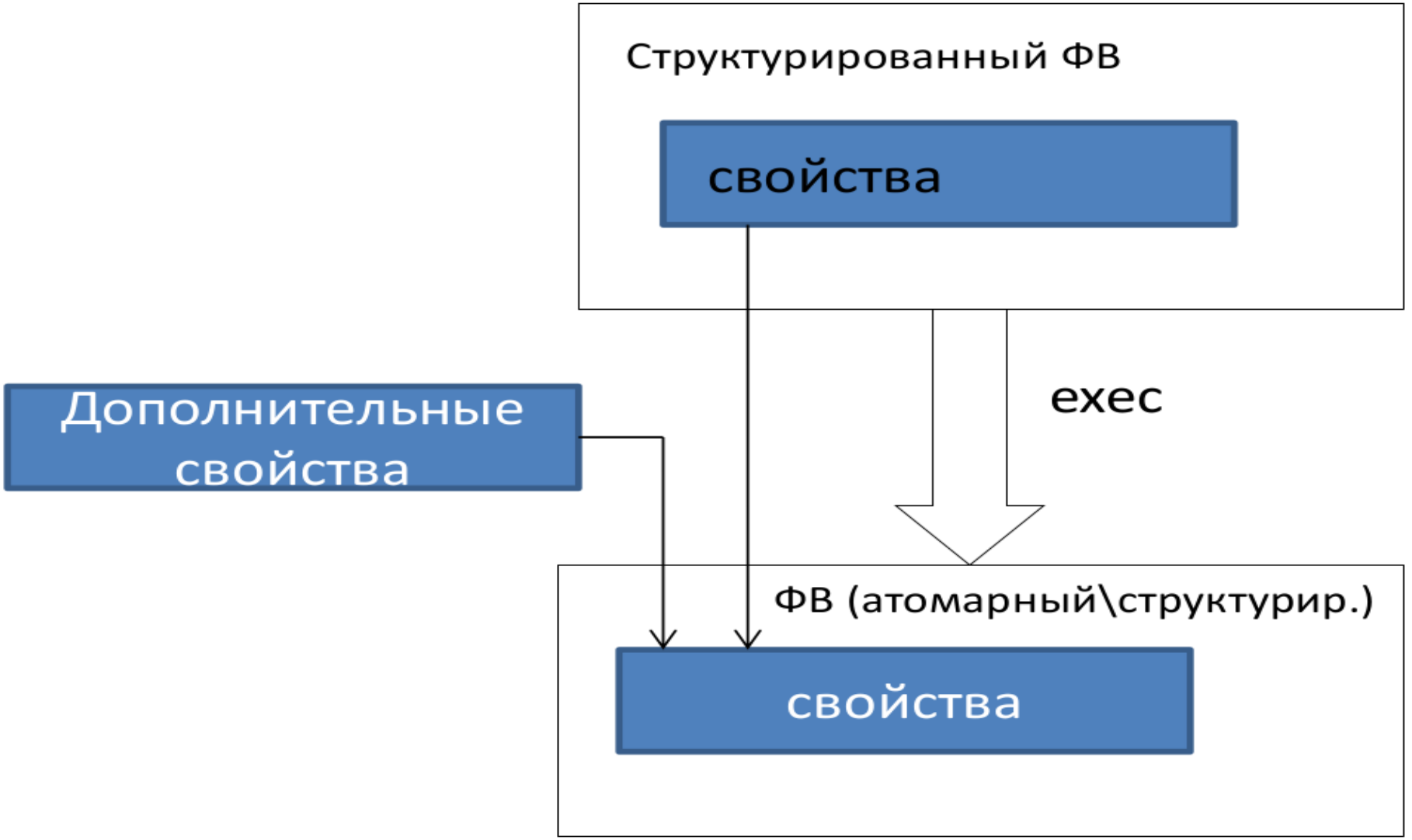
Архитектура менеджера ФВ



Менеджер ФВ с поддержкой GPU



MDOI container

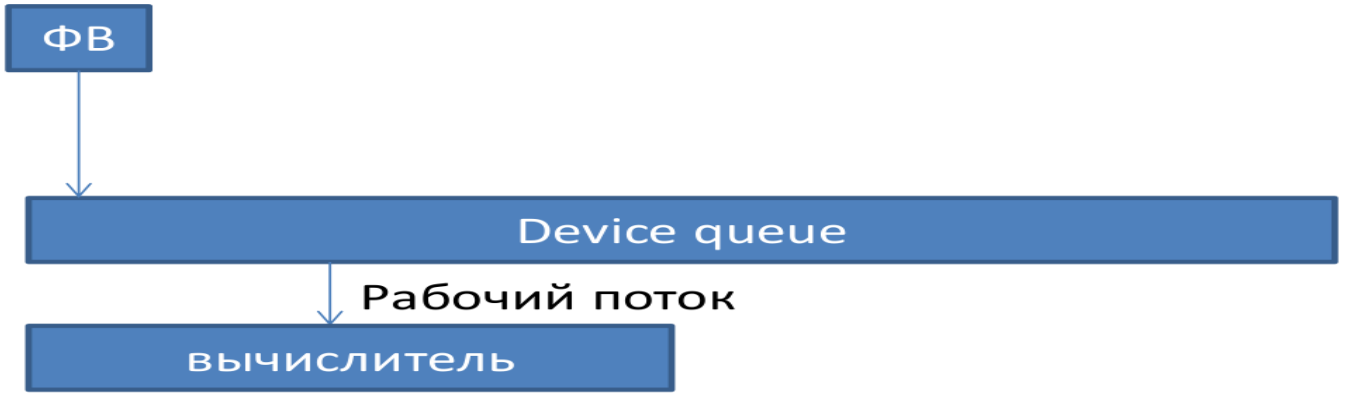


Classifier

- Осуществляет выбор алгоритма распределения вычислительной нагрузки для заданного подмножества ФВ
- Выбор осуществляется на основании рекомендаций пользователя

Resources manager

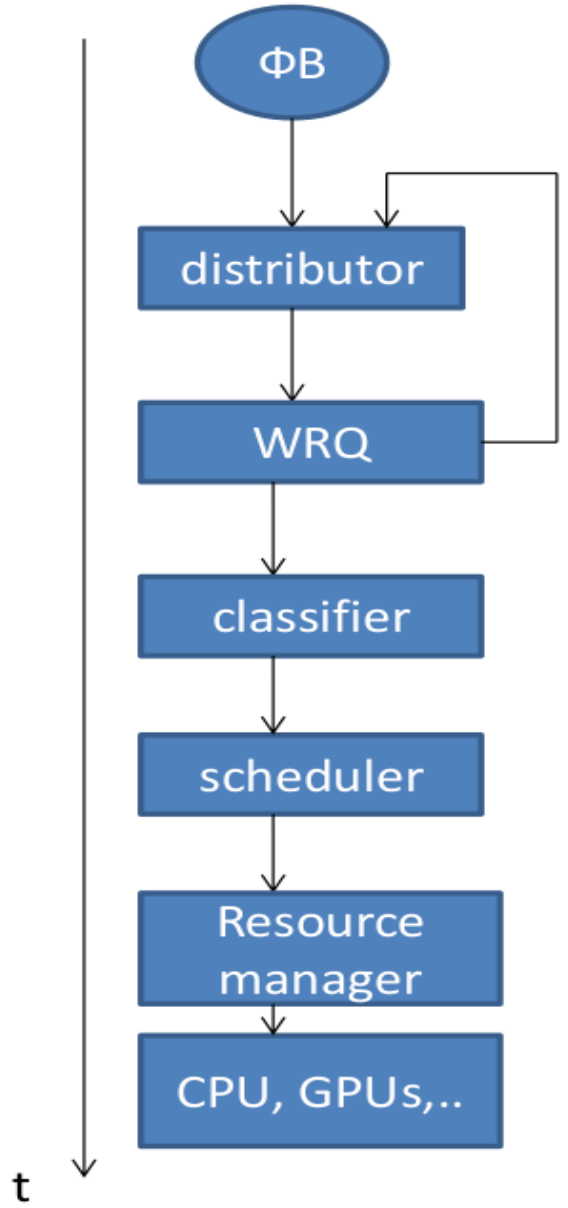
- Отвечает за непосредственно исполнение ФВ, управление памятью GPU и передачей данных между GPU и CPU



Планировщики (schedulers)

- Планировщики для заданного непустого подмножества ФВ ставят в соответствие целое неотрицательное число (номер устройства, на котором ФВ будет исполнен)

Исполенние ФВ



Рекомендации

Рекомендации

- Выбор алгоритма распределения нагрузки управляется рекомендациями пользователя
- Механизм полуавтоматического выбора фрагментации управляется рекомендациями
- Рекомендации реализованы в виде “пометок”, которые разработчик ФП оставляет в коде ФП

Использование алгоритма Rob

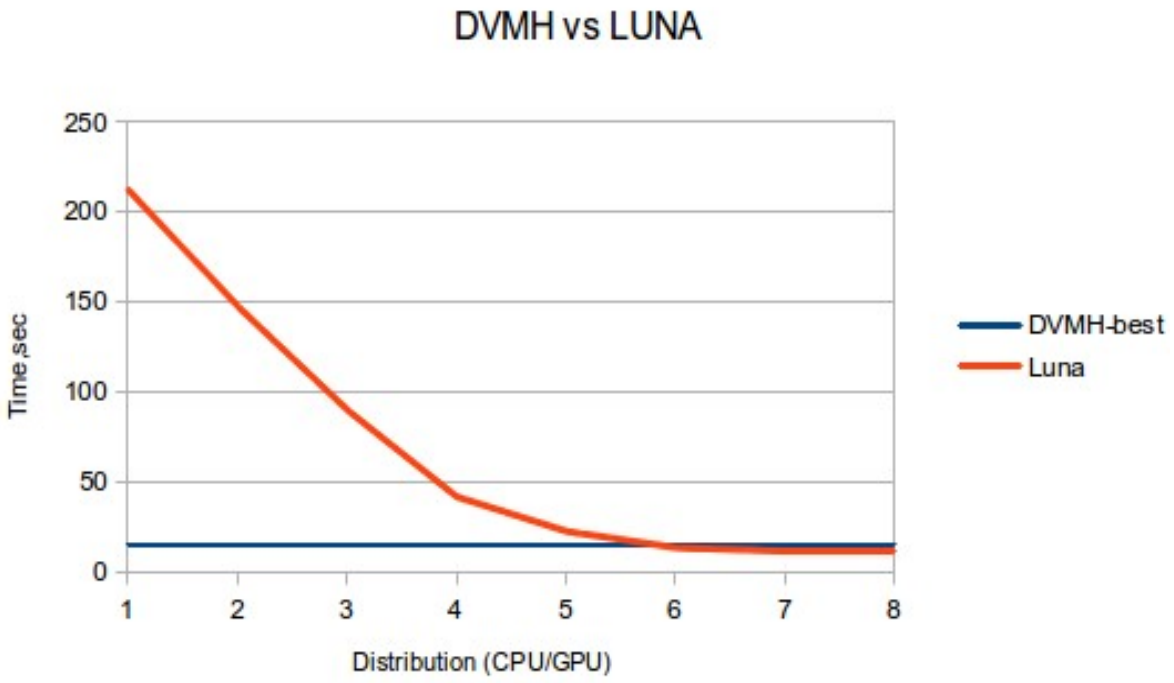
```
import cf_init(name #val) as init;
import cf_cuda1(value #in, name #out) as cuda1:CUDA;
import cf_cuda2(value #in, name #out) as cuda2:CUDA;
import cf_print(int, value #in) as print;

##const SZ 50

sub main()
{
        df data, data1, data2, data3;

        for i = 0 .. $SZ -1 <SPACEITERS>
                cf ini[i] : init(data[i]);

        for j = 0 .. $SZ – 1 <SPACEITERS> {
                cf cu1[j] : cuda1(data[j], data1[j])-->(data[j]);

                cf cu2[j] : cuda2(data1[j], data2[j])-->(data1[j]);

        }

        for l = 0 .. $SZ – 1 <SPACEITERS>
                cf prt[l] : print(l, data2[l]);//-->(data1[l]);
}
```

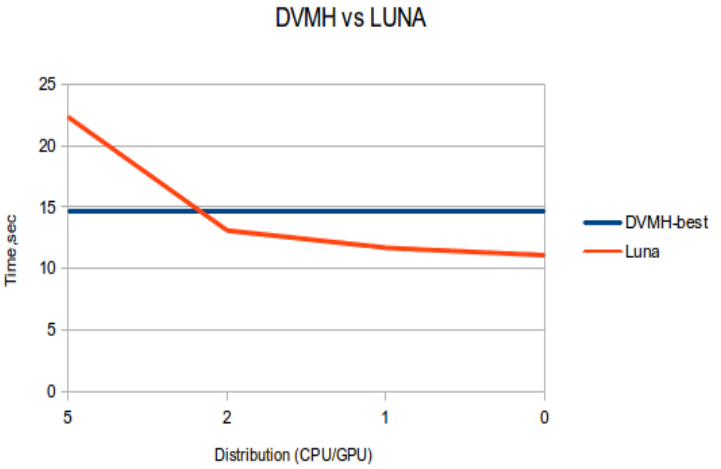
Использование механизма рефрагментации

```
…….
sub main()
{
        df params, data, iter, max, n , sz;

        init_p(sz,n);<1DREFRAGINIT=0,1>
……

 …….
 for i = 0 .. n - 1
         cf psn[i] : cf_calc_puasson(data_prev[i], params[i], data[i], maxdiff[i], sz) -->
 (data_prev[i]);<REFRAGSZIDX=4>
 ……..
```

Тестирование

LuNA vs DVMH

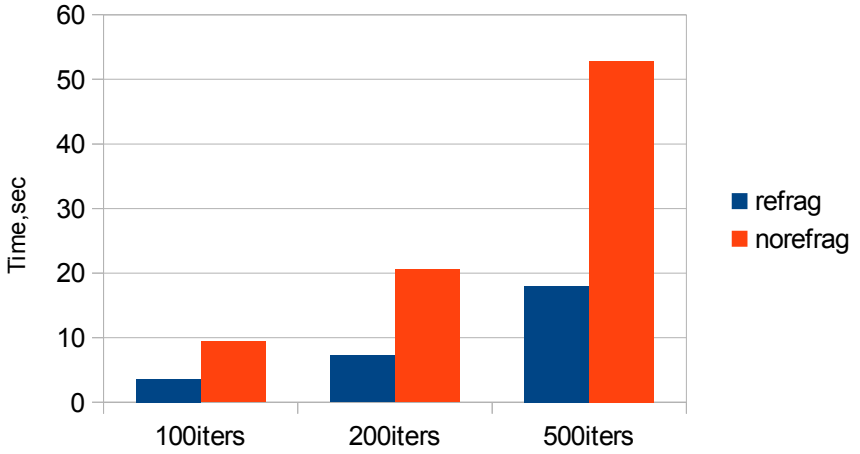




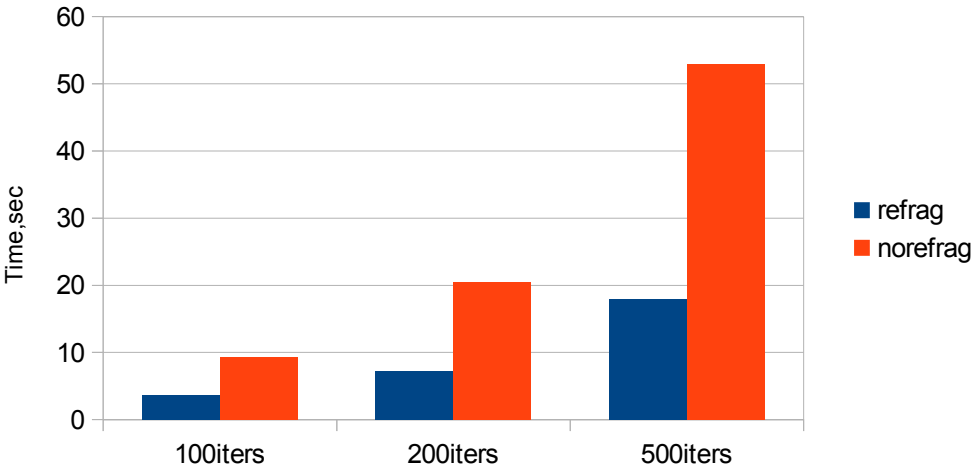
Задача: решение уравнения Пуассона, 1Д фрагментация, 2000x2000x40frags(double)

Оборудование: CPU Intel core i7-3820 3.5GHz, GPU Nvidia GeForce GTX 650

Прирост производительности при полуавтоматическом выборе фрагментации

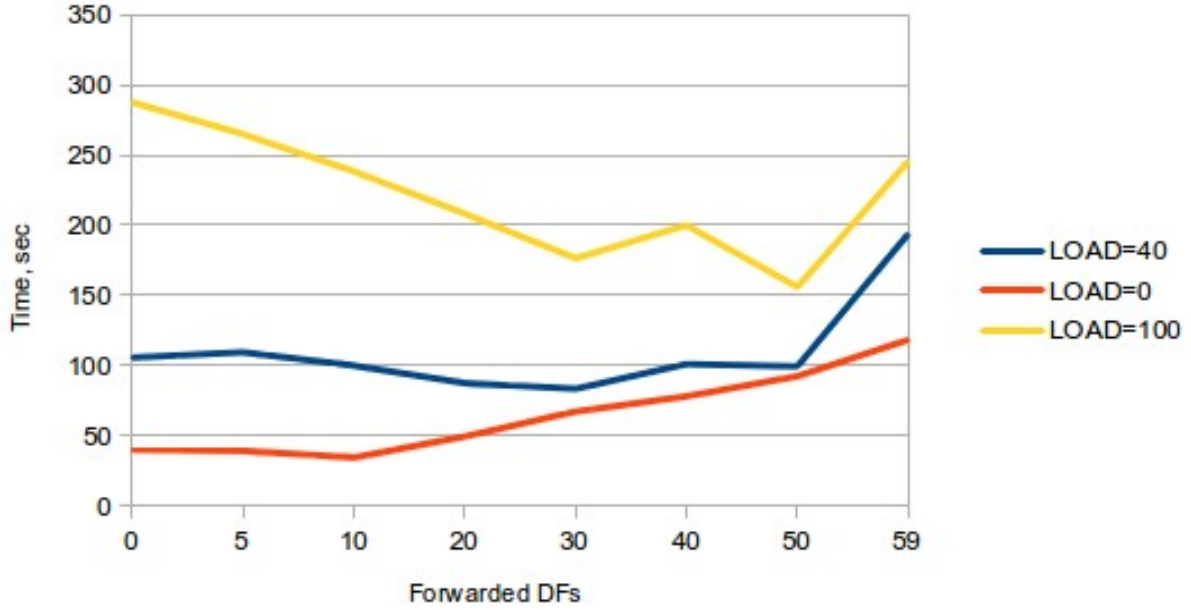


Исходная фрагментация: 100 фрагментов 800x800



Исходная фрагментация: 50 фрагментов 1600x800

GPU: Quadro FX 4800



Синтетический тест: “утяжелённый” решатель уравнения Пуассона, 1Д декомпозиция

Размер области: 2000x2000 110 фрагментов

Оборудование: CPU Intel core i7-3820 3.5GHz, GPU Nvidia GeForce GTX 650