iSCALARE

Лаборатория суперкомпьютерных технологий для биомедицины, фармакологии и малоразмерных структур

Полноплатформенная симуляция DES Многопроцессорные системы

Григорий Речистов

grigory.rechistov@phystech.edu

25.03.2013

- Discrete event simulation
- Многопроцессорные модели
- Гиперсимуляция

На предыдущих лекциях:

- Рассмотрены различные способы моделирования ЦПУ
- Модели работали «по шагам» / «пробежками»
- Модель работала в «вакууме»
  - Единственное внешнее устройство — ОЗУ
- Как поступать, если мы хотим иметь не только ЦПУ, но и периферийные устройства?
  - Или несколько ЦПУ?

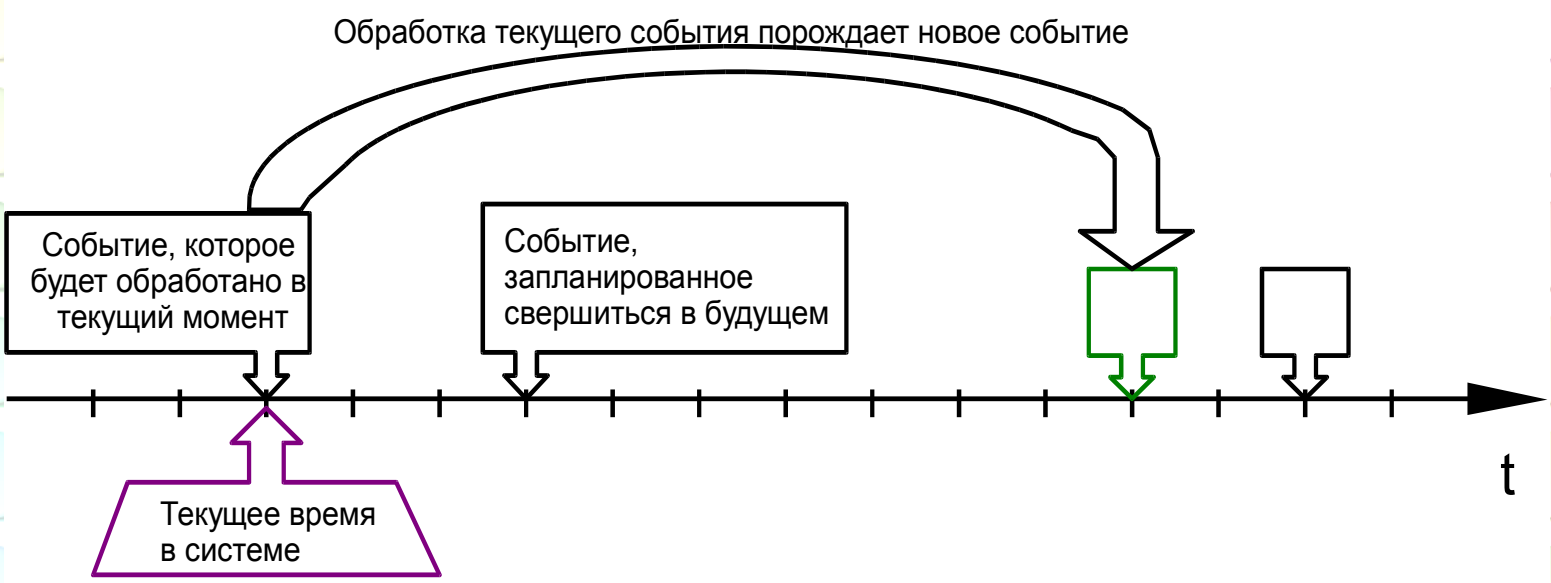
Дискретные события

- Состояние любого объекта в любой момент времени может быть описано полностью
- Изменения состояния происходят мгновенно
- Объекты не содержат в себе переходных состояний

Взаимодействие внутри полной модели

- Модель состоит из объектов
- Объекты могут слать друг другу (и сами себе) сообщения
- Реакции на сообщения могут быть запланированы к доставке не мгновенно
- Состояние полной системы = сумма состояний всех объектов + очередь ещё не доставленных событий

Очередь* событий



* Правильнее было бы назвать списком.

Алгоритм DES

```
struct evt_t { fn(); *obj;}
uint sim_time = 0;
while (! empty(queue)) {
   sim_time += get_delta(queue);
   evt_t evt = pop(queue);
   evt.fn(evt.obj, queue);
}
```

Симуляция продвигается исключительно обработкой событий

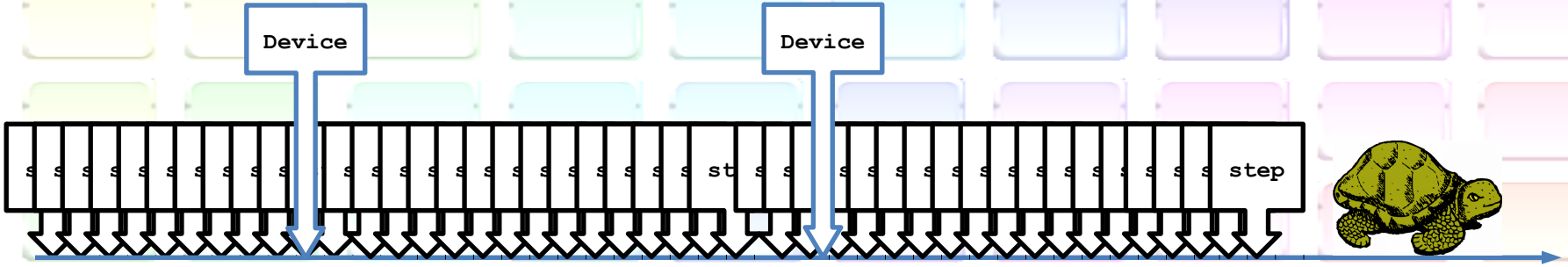
Простой пример: периодический таймер

```
struct {
  uint64 val;
  uint threshold;
} timer;
void handle(obj, queue) {
  obj->val += obj->threshold;
  post(queue, threshold, obj, handle);
}
```

События

- Порождаемые события не могут попасть в прошлое, т.е. иметь метку времени меньше, чем текущее время
- Обработка событий может не только порождать события в будущем, но и отменять некоторые из них (ещё не обработанные)
- Несколько событий могут иметь одинаковую метку времени
- В модели может существовать больше одной очереди
  - Первая очередь измеряется в инструкциях процессора, а другая — в тактах
  - Многопроцессорные системы, в которых с каждым ЦПУ связана своя очередь

Моделирование ЦПУ в DES

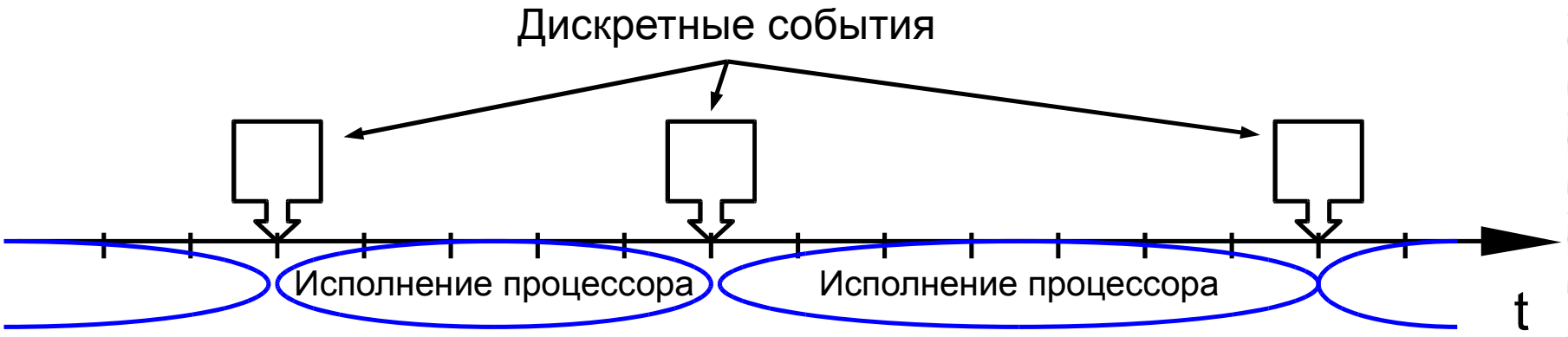


Лучшее, что можно будет использовать — это интерпретация

Два класса моделей

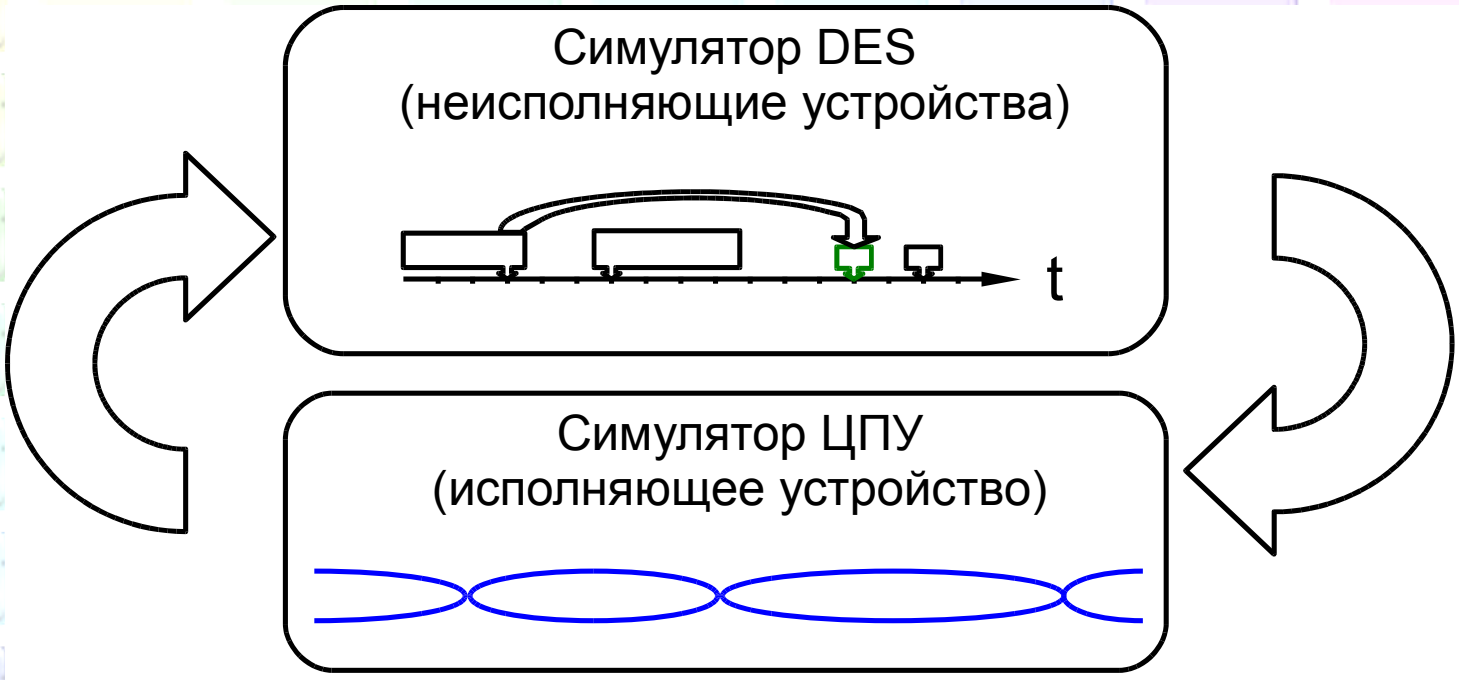
- Исполняющая (executing)
  - Меняет своё состояние каждый шаг (такт, инструкция)
  - Активное устройство (execution driven)
- Неисполняющая (non-executing)
  - Изменение состояния устройства асинхронно
  - Пассивное устройство (запрос-отклик)

Схема симуляции системы с ЦПУ



1. Определяется длительность интервала, в течение которого в моделируемой системе не произойдёт никаких событий. Эта величина равна времени ближайшего не обработанного события в очереди.
2. Управление передаётся в модель процессора, которая исполняется некоторое время, не превышающее найденное в первом пункте значение. Затем она останавливается и возвращает управление симулятору.
3. Симулируемое время продвигается на число тактов, потраченных процессором. События обрабатываются по модели DES. Затем мы переходим к первому шагу.

Ко-симуляция



Многопроцессорные системы

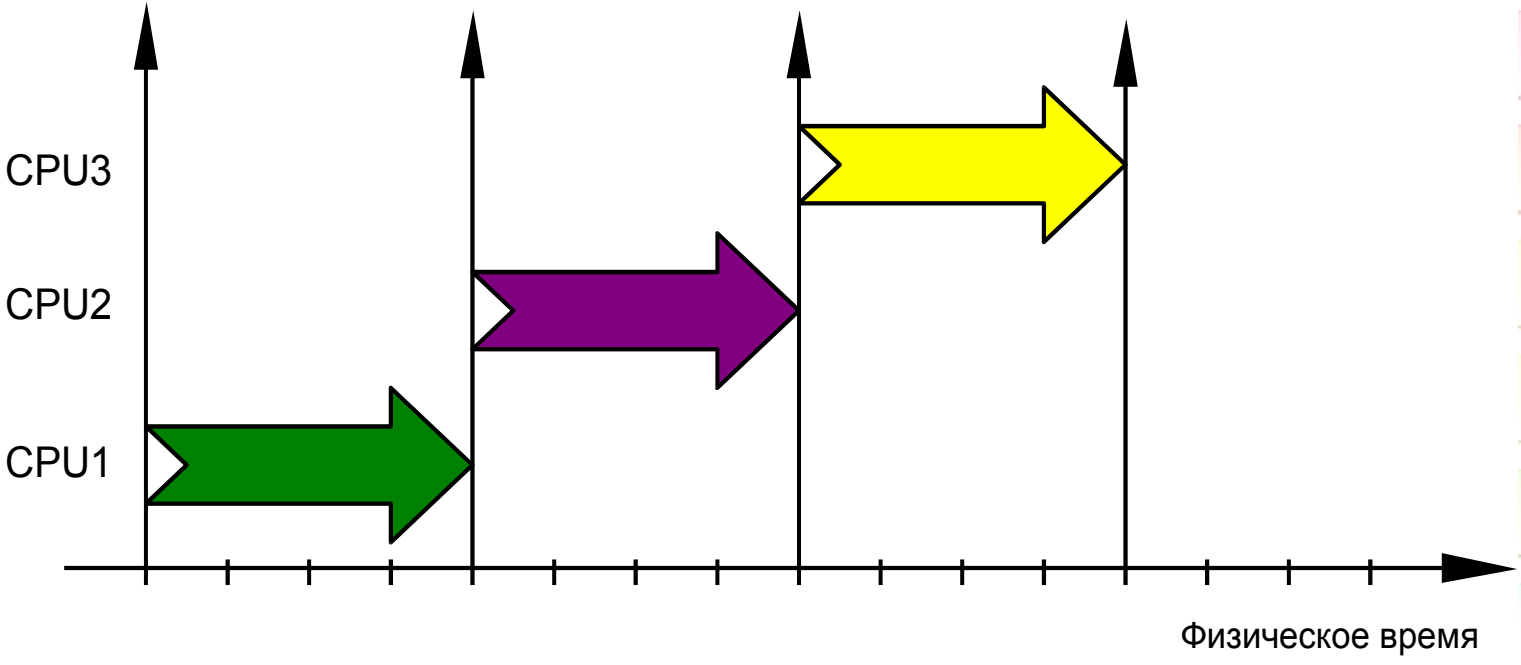
- Гостевые ЦПУ исполняются независимо друг от друга, иногда обмениваясь сообщениями (доступ к памяти)
- Имеется только один хозяйский процессор
- Очевидное решение – поочерёдное пошаговое исполнение каждой модели
  - Локальное время будет отличаться не более чем на 1 такт
- Недостаток — частое переключение контекста, медленность симуляции

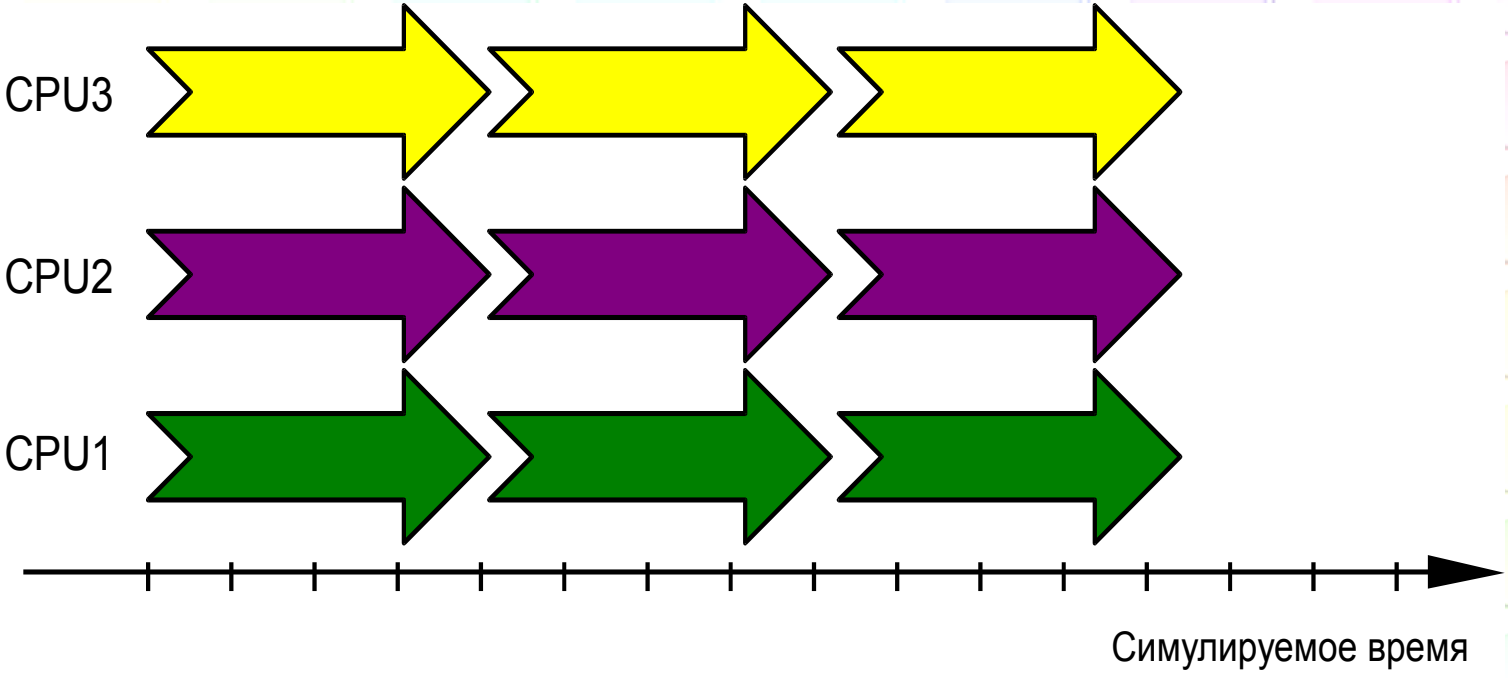
Оптимизация

- В реальности полная синхронность потоков приложению не гарантируется — оно должно быть написано таким образом, чтобы работать на разных системах
- Можно моделировать отдельные процессоры большими кусками симулируемого времени
- Максимальный размер такого интервала — *квота* исполнения

  == quota == quantum == switch time

Переключение текущего устройства




CPU3
CPU2
CPU1
Симулируемое время

Замечания

- Процессор может исполнить меньше инструкций, чем содержится в выданной ему квоте.
- Не следует увлекаться излишне большими квотами, пытаясь ускорить исполнение — это может негативно повлиять на точность модели
  - Каждое устройство бежит в полной изоляции, остальные заморожены и не могут посылать ему никакие сигналы.
- В симуляторе могут быть реализованы псевдо-события, обработка которых вызывает переключение текущего устройства.

Алгоритм

```
while(true) {
  int cpu = select_next_cpu(cpu_objs);
  set_current_cpu(cpu_n);
  uint quota = get_nearest_event(queue);
  uint executed = cpu.execute(quota);
  ASSERT(executed <= quota);
  sim_time += executed;
  advance(queue, executed);
  check_events(queue);
}
```

Пример: очереди сообщений

mc - /tmp_proj/grechist/iscalare/genx - Shell No. 2 - Konsole

Session Edit View Bookmarks Settings Help

```
simics> master0.mb.cpu0.core[0][0]->time_queue
[["sim", "Time Quantum End", "master0.mb.cpu0.core[0][0]", "time_quantum", 68631], ["master0.m
b.sb.uhci[0]", "frame_update", 0, "default", 2268631], ["master0.mb.sb.uhci[1]", "frame_update
", 0, "default", 2268631], ["master0.mb.sb.uhci[2]", "frame_update", 0, "default", 2268631], [
"master0.mb.sb.uhci[3]", "frame_update", 0, "default", 2268631], ["master0.mb.sb.uhci[4]", "fr
ame_update", 0, "default", 2268631], ["master0.mb.sb.uhci[5]", "frame_update", 0, "default", 2
268631], ["master0.mb.sb.timer", "handle irq", 0, "default", 31377806], ["master0.mb.sb.lpc",
"pm1_ovf", 0, "default", 6057665917]]
simics> master0.mb.cpu0.core[0][0]->step_queue
[]
simics> help master0.mb.cpu0.core[0][0]->step_queue
Attribute <x86-nehalem>.step_queue

   Optional attribute; read/write access; type: [[o|n,s,a,s,i]*].

   ((object, evclass, value, slot, step)*). Pending step queue events.

simics>
```

Пример: квота ЦПУ

mc - /tmp_proj/grechist/iscalare/genx - Shell No. 2 - Konsole

Session Edit View Bookmarks Settings Help

```
simics> cpu-switch-time
Current time quantum: 9.52381e-05 s
  314285.7  master0.mb.cpu0.core[0][0]
  314285.7  master0.mb.cpu0.core[1][0]
  314285.7  master0.mb.cpu1.core[0][0]
  314285.7  master0.mb.cpu1.core[1][0]
Default time quantum: 9.52381e-05 s
simics> master0.mb.cpu0.core[1][0]->freq
master0.mb.cpu0.core[1][0]->freq_mhz  master0.mb.cpu0.core[1][0]->frequency
simics> master0.mb.cpu0.core[1][0]->frequency
[3300000000, 1]
simics> master0.mb.cpu0.core[1][0]->freq_mhz
3300
simics> master0.mb.cpu0.core[1][0]->freq_mhz = 1000
simics> cpu-switch-time
Current time quantum: 9.52381e-05 s
  314285.7  master0.mb.cpu0.core[0][0]
   95238.1  master0.mb.cpu0.core[1][0]
  314285.7  master0.mb.cpu1.core[0][0]
  314285.7  master0.mb.cpu1.core[1][0]
Default time quantum: 9.52381e-05 s
simics>
```

Гиперсимуляция (1/3)

Иногда модель исполняется очень медленно [без видимой причины]

Ожидает ответа от замороженных агентов

Гиперсимуляция (2/3)

- Исполняемые инструкции не изменяют состояние внешних объектов
- Результат работы блока всегда один и тот же и нет смысла моделировать его пошагово
- Оптимизация:
  - Изменяем состояние сразу на конечное (конец квоты)
  - Продвигаем значение симулируемого времени

Гиперсимуляция (3/3)

Гиперсимуляция (4/3)

- Автоматическое нахождение шаблонов гиперсимуляции затруднено
- Их необходимо определять человеку
- Далеко не каждый блок гостевого кода может быть оптимизирован таким образом

Вопрос

При каких сценариях использования симулятора излишняя скорость вредна?

Рекомендуемая литература

M. C. (Mike) Albrecht. Introduction to Discrete Event Simulation. 2010.

http://www.albrechts.com/mike/DES/Introduction%20to%20DES.pdf

На следующих лекциях

- Параллельные симуляторы
  - PDES
  - Проблемы
  - Решения
- Примеры практических реализаций

Спасибо за внимание!

Все материалы курса выкладываются на сайте лаборатории:
http://iscalare.mipt.ru/material/course_materials/