ФРТК

iSCALARE

intel

Лаборатория суперкомпьютерных технологий для биомедицины, фармакологии и малоразмерных структур

Двоичная трансляция и симуляция

Григорий Речистов

grigory.rechistov@phystech.edu

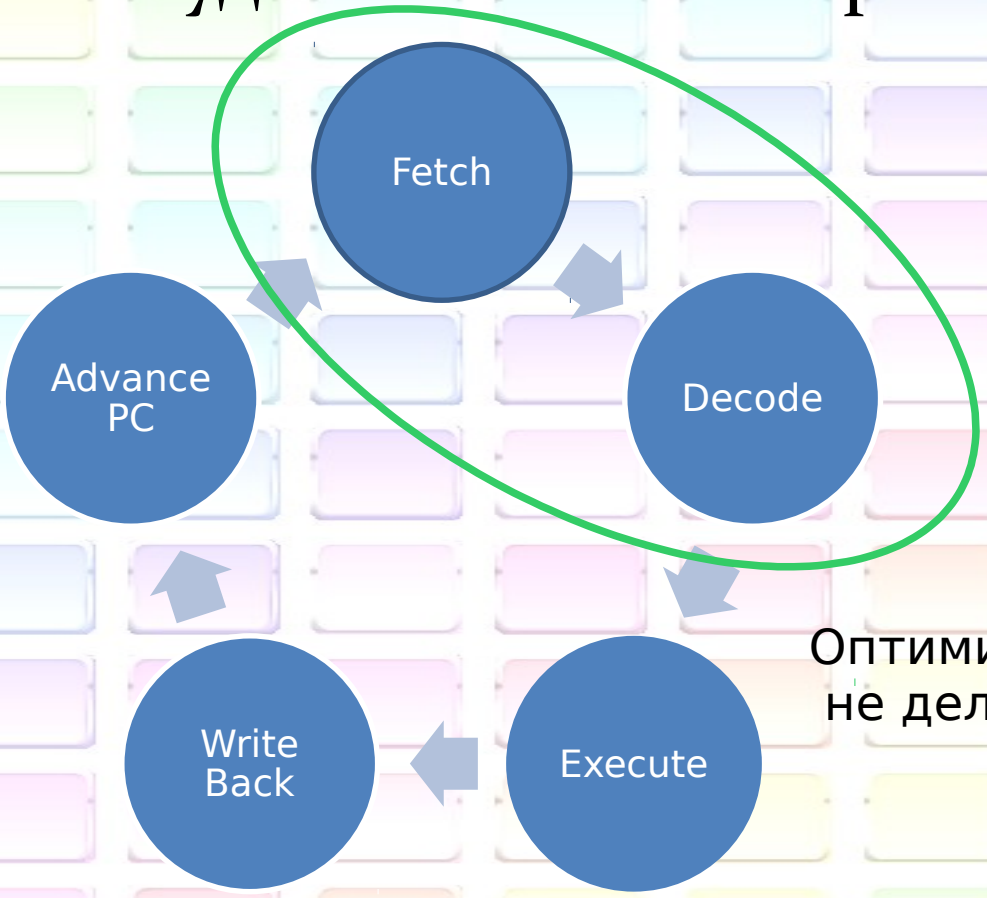
04.03.2013

- Статическая ДТ
- Динамическая ДТ
- Проблемы
- Решения

На предыдущих лекциях:

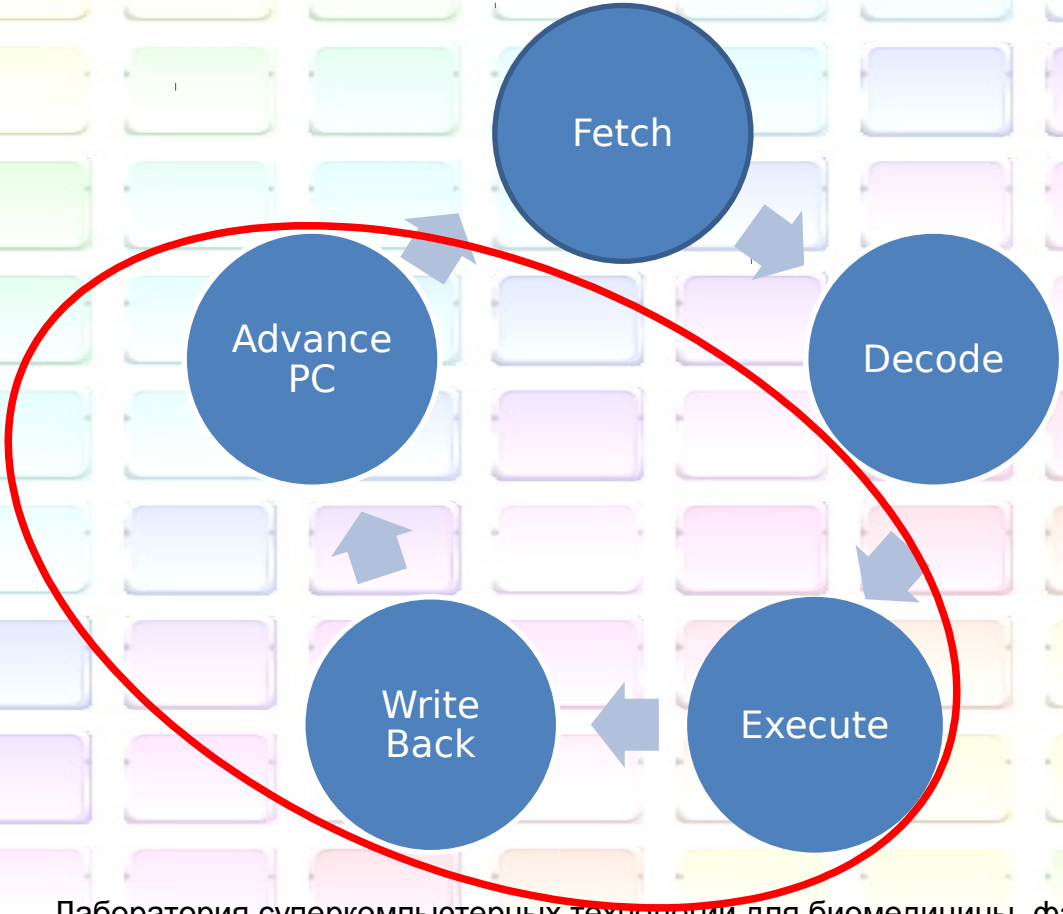
- Интерпретация является относительно простым механизмом симуляции основного цикла процессора
- В своей простейшей форме она обеспечивает малую скорость модели
- Улучшения интерпретатора основаны на переиспользовании предыдущих результатов (декодирования)

Что нам удалось оптимизировать



Оптимизировать — не делать вообще

Что бы нам ещё улучшить?

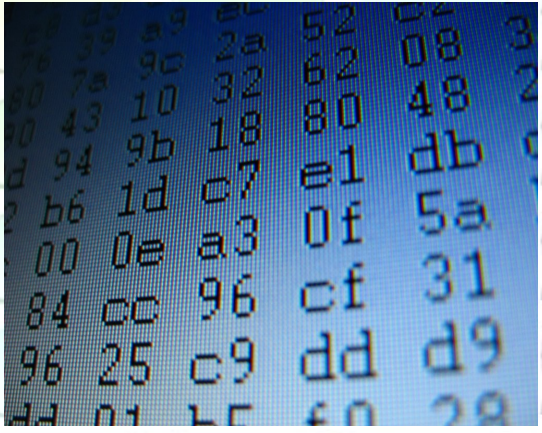


Языки высокого уровня, компиляция

- Basic — большинство существующих реализаций являются интерпретаторами
  - Прочитал строку — распознал команды — исполнил
  - Хорошо, но медленно
- FORTRAN, C, Pascal. Работа в два прохода
  1. Компилятор преобразует исходный код в машинный (опционально применяет оптимизации)
  2. Машинный код исполняется

Трансляция гостевого машинного кода в хозяйский

Входной язык —
гостевой машинный код



1. Трансляция

Целевой язык —
хозяйский машинный код

```
Disassembly of section .init:

0000000000401fd0 <_init>:
  401fd0:       48 83 ec 08             sub    $0x8,%r
  401fd4:       e8 f7 25 00 00          callq  4045d0
  401fd9:       e8 82 26 00 00          callq  404660
  401fde:       e8 dd f7 00 00          callq  4117c0
  401fe3:       48 83 c4 08             add    $0x8,%r
  401fe7:       c3                      retq

Disassembly of section .plt:

0000000000401ff0 <__ctype_toupper_loc@plt-0x10>:
  401ff0:       ff 35 fa 6f 21 00       pushq  0x216
  401ff6:       ff 25 fc 6f 21 00       jmpq   *0x216
  401ffc:       0f 1f 40 00             nopl   0x0(%rax
```

2. Исполнение

Двоичная трансляция (Binary translation) ДТ (BT)

- Перевод кода из гостевого ISA в *эквивалентный* код хозяйского ISA
- Результаты трансляции можно переиспользовать (если они сохранены и валидны)
- Исполнение не испытывает замедления на «родной» аппаратуре

Алгоритм

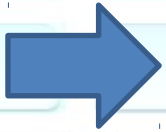
```
translate() {
PC = start_addr;
bufptr = start_buf;
while (! enough) {
   instr = fetch(PC);
   opcode = decode(instr);
   capsule = capsules[opcode];
   memcpy(capsule, bufptr);
   PC ++;
   bufptr ++;
};
memcpy(return_jmp, bufptr);
};
```

```
execute(){
   setjmp(back);
   goto start_buf;
   back: ;
};
```

Капсула

Гостевой код, архитектура IA-32 EMT (64 бит)

```
addq (%rbx), %rax
```



Хозяйский код, архитектура IA-32 (32 бит)

```
push RBX_OFF(%ebp) ; vaddr
call v2h ; eax <- host_addr
movl  (%eax), %edx
movl 4(%eax), %ebx
addl  %edx,   RAX_OFF(%ebp)
addcl %ebx, 4+RAX_OFF(%ebp)
```

ebp – указатель на гостевое состояние
RxX_OFF – константа-смещение внутри гостевого состояния для регистра RxX.
v2h – функция преобразования адресов

Статическая ДТ

- Стадия ДТ исполняется заранее, до исполнения
- Результат ДТ сохраняется на диске
- Мы можем применить агрессивные многопроходные оптимизации
- Пример: *Digital FX!32*. **IA-32 → Alpha**
  http://www.usenix.org/publications/library/proceedings/usenix-nt97/full_papers/chernoff/chernoff.pdf
- Бонус-сценарий: двоичная оптимизация

Динамическая ДТ

- Происходит непосредственно во время симуляции, результат хранится в памяти
- Фаза ДТ чередуется с исполнением → не может быть длительной
- ДТ ограничена в оптимизациях
- Может обработать самомодифицирующийся код
- Корректная полноплатформенная статическая ДТ *невозможна* без механизма поддержки времени исполнения, т. е. динамической ДТ

Статическая vs динамическая ДТ

- В настоящее время динамическая ДТ имеет более широкую область применения, чем статическая
- JIT ДТ
  - Java
  - .NET
  - Python

Почему оптимизации при ДТ затруднительны

- В отличие от ЯВО, машинный код содержит меньше информации об исходном алгоритме
- Мы не можем делать многие предположения, необходимые для компиляторных оптимизаций без нарушения корректности
  - Адреса переменных — их нет
  - Границы процедур — их нет
  - Адреса переходов — известна только часть из них

Самомодифицирующийся код (self modifying code, SMC)

- Необходимо следить за тем, чтобы трансляции оставались валидны
- При необходимости симулировать код, для которого не существует актуальной трансляции, необходимо выполнить ДТ
- Частые ретрансляции сводят преимущество ДТ над интерпретаторами на нет

Как детектировать SMC (1/2)

```
write(paddr, data) {
   <...>
   vhptr = v2h(paddr);
   if (tc_has_cached(paddr)) {
      tc_invalidate(paddr);
      tc_retranslate(paddr);// а можно и
        подождать
   }
}
```

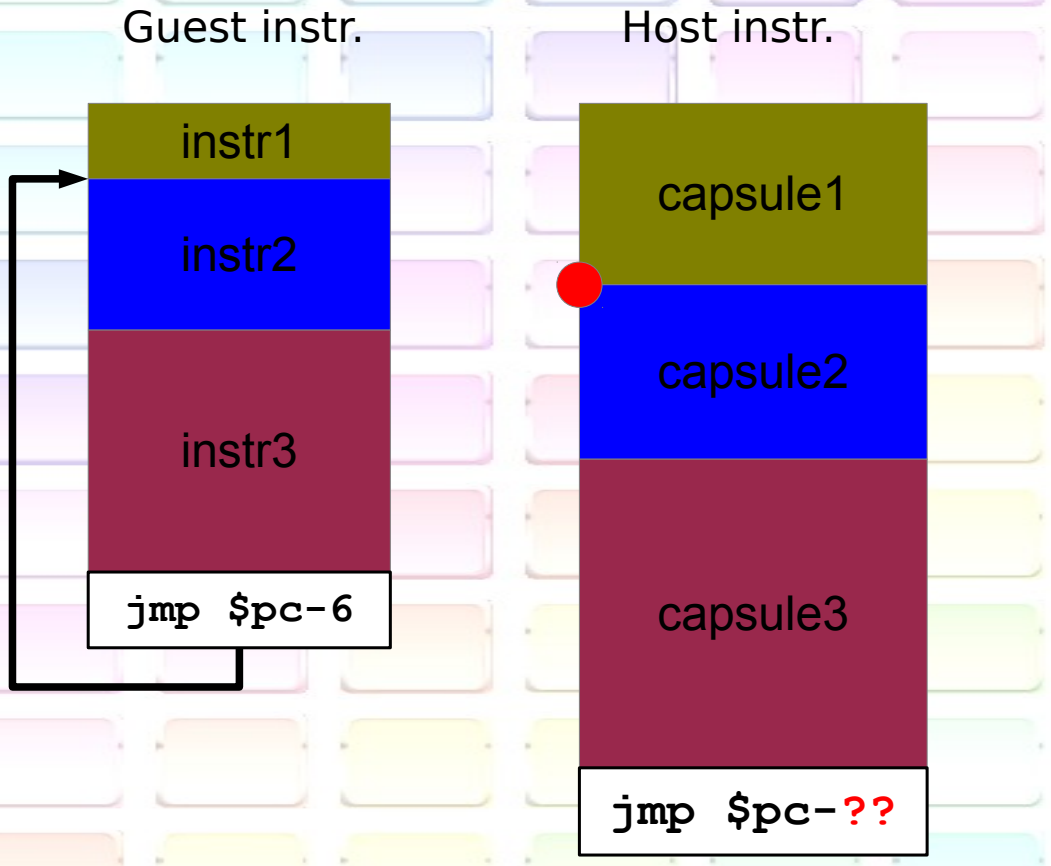
Как детектировать SMC (2/2)

- Поддержка со стороны хозяина: помечать транслированные страницы как «read-only», при попытках записи возникает исключение, обработчик сбрасывает связанную трансляцию

Обнаружение кода

Code discovery problem

- Найти границы инструкций
- Отличить код от данных



Блоки для трансляции

```
while (! enough) {…}
```

Чем ограничивать длину блоков трансляции?

1. Гостевая страница
2. Цепочка инструкций (трасса), исполнявшаяся ранее

Гостевая страница

Трасса ранее исполнявшихся инструкций

- Цепочка, однажды исполнившаяся, скорее всего, будет исполняться ещё раз в таком же порядке
- Отдельные трассы могут захватывать одни и те же адреса, но достигнутые различными путями
- Трансляция трассы прекращается, когда адрес следующей инструкции неизвестен: условный/непрямой переход, вызов процедуры и т. п.
- Средняя длина трассы: 6 инструкций. 20 инструкций — уже пора обрывать

И ещё:

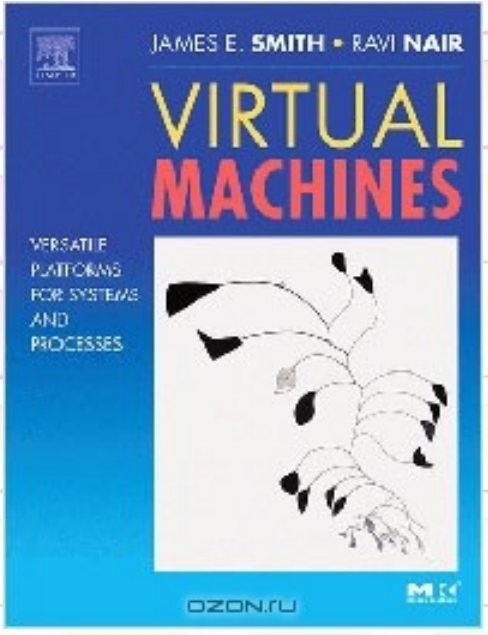
- Трансляция становится неактуальной при изменении режима процессора
- Смысл машинного кода изменяется
- Но: можно хранить результаты ДТ с ассоциированным режимом, для которого они валидны
- Иметь несколько трасс для одного и того же региона памяти
- Можно шарить кэш трансляций между несколькими гостевыми ЦПУ

Итоги

- Интерпретация, компиляция (трансляция)
- Двоичная трансляция. Статическая, динамическая трансляция
- Капсула
- SMC
- Code discovery
- (Не)возможность оптимизации кода при ДТ

Рекомендуемая литература (1/2)

Jim Smith, Ravi Nair. Virtual Machines: Versatile Platforms for Systems and Processes. 2005



Рекомендуемая литература (2/2)

Fabrice Bellard. **QEMU, a Fast and Portable Dynamic Translator**

- `http://www.usenix.org/publications/library/proceedings/usenix05/tech/freenix/full_papers/bellard/bellard.pdf`

Mathieu Brethes. **Atome - Binary Translation for Accurate Simulation**

- `http://www8.cs.umu.se/education/examina/Rapporter/MathieuBrethes.pdf`

Nigel Topham, Daniel Jones. **High Speed CPU Simulation using JIT Binary Translation**

- `http://homepages.inf.ed.ac.uk/npt/pubs/mobs-07.pdf`

На следующей лекции:

11 марта 2013 — пары не будет

- Ещё быстрее!
- Прямое исполнение
- Аппаратная поддержка

Спасибо за внимание!

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