БУДУЩЕЕ
В НАШИХ
РУКАХ

Инструмент для автоматической генерации тестов на подсистему памяти RISC-V CPU

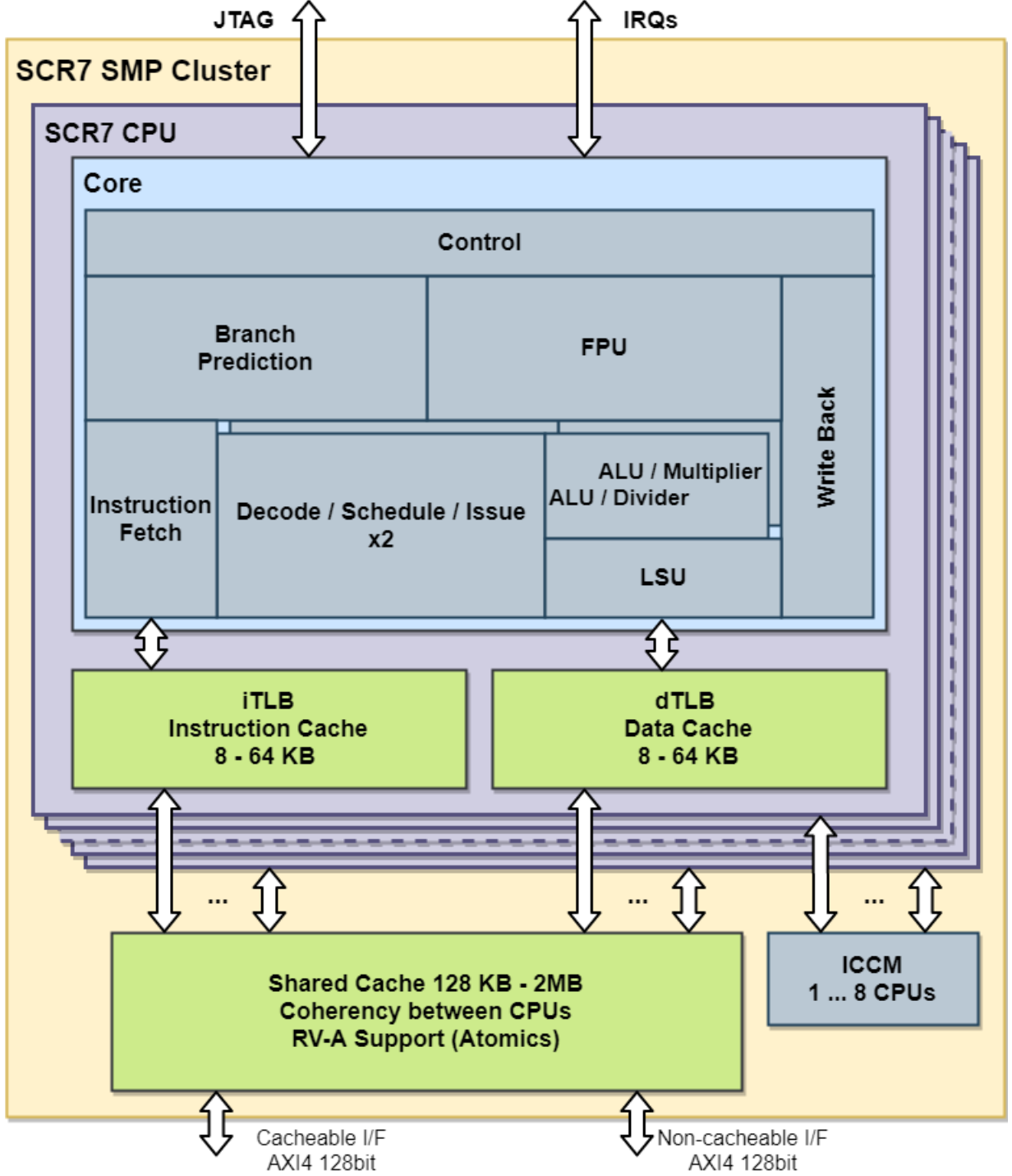
Данила Печенев <d.pechenev@spbu.ru>
Научный руководитель: руководитель Лаборатории YADRO СПбГУ, Я. А. Кириленко
Консультант: ведущий инженер-программист, ООО «КНС ГРУПП», Д. В. Донцов

RISC-V Russia

Syntacore™

Custom cores and tools

АЛЬЯНС RISC-V



Тесты/бенчмарки – приложения для измерения производительности различных компонентов процессора

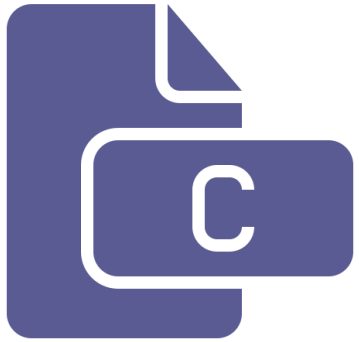
- Кэши разного уровня
- DRAM (оперативная память)
- MMU/TLB
- Предсказатель переходов
- Префетчер

Картинка:
https://syntacore.com/page/products/processor-ip/scr7

Недостатки используемых тестов

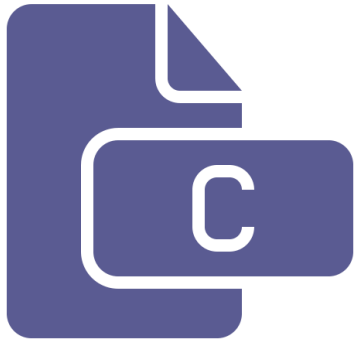
- Нет гибкости изменения параметров
  - Мешают оптимизации компилятора: перестановка инструкций, выбрасывание кода

- Некоторые тесты можно запускать только под операционной системой
  - Эффекты работы ОС снижают точность: многозадачность, прерывания, демоны
  - Промахи в TLB



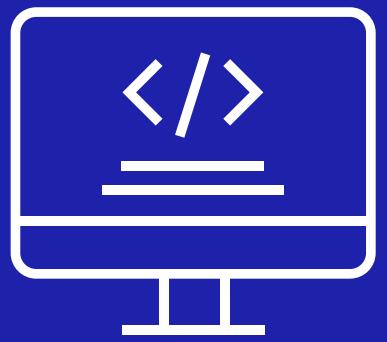
Недостатки используемых тестов

- Нет гибкости изменения параметров
  - Мешают оптимизации компилятора: перестановка инструкций, выбрасывание кода
- Некоторые тесты можно запускать только под операционной системой
  - Эффекты работы ОС снижают точность: многозадачность, прерывания, демоны
  - Промахи в TLB



Инструмент для генерации тестов

- ✓ Основу тестов составляют неизменяемые ассемблерные вставки
  - Нет влияния компилятора
  - Можно задать любую последовательность инструкций
- ✓ Генерируемые тесты запускаются под baremetal (режим работы без операционной системы)
  - Нет влияния эффектов работы ОС
  - Использование MMU можно контролировать



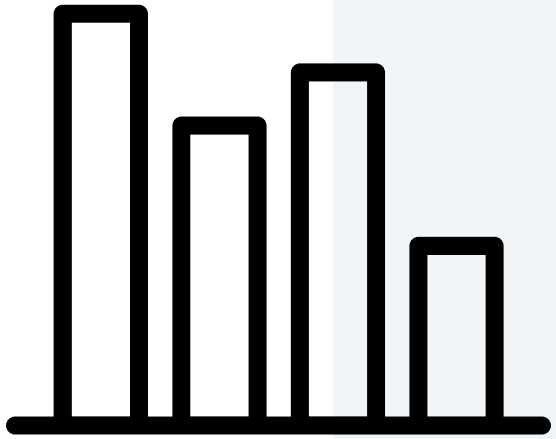
Важнейшие характеристики подсистемы памяти

Время доступа (англ. latency) — задержка в тактовых циклах, которую создает load-инструкция в цепочке **зависимых** инструкций

Пропускная способность (англ. bandwidth, throughput) — максимальное количество **независимых** инструкций, взаимодействующих с памятью (load- и store- инструкции в любом соотношении), которые могут быть исполнены за один тактовый цикл

load-инструкция — инструкция, выгружающая данные из памяти (из кэша какого-либо уровня либо DRAM) в регистр

store-инструкция — инструкция, сохраняющая данные в память (в кэш либо DRAM)



Цель и задачи

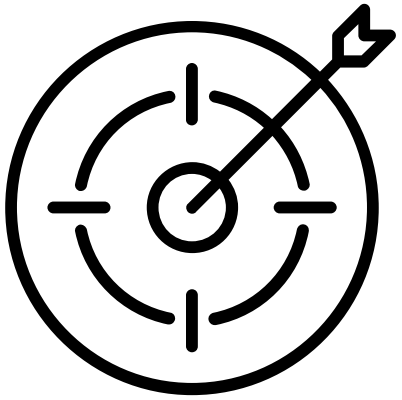
Целью работы является разработка инструмента, способного автоматически генерировать тесты на подсистему памяти RISC-V CPU

Цель и задачи

Целью работы является разработка инструмента, способного автоматически генерировать тесты на подсистему памяти RISC-V CPU

Задачи:

- Определить и обосновать формат конфигурации теста, включая доступные параметры и ограничения на них
- Реализовать генерацию тестов на подсистему памяти RISC-V CPU с поддержкой всех необходимых параметров конфигурации
- Проверить средствами моделирования сгенерированные бенчмарки для некоторых популярных сценариев, а именно тесты на измерение
  - времени доступа к L1 кэшу
  - времени доступа к L2 кэшу
  - времени доступа к L3 кэшу
  - времени доступа к DRAM (при правильном предсказании префетчера)
  - времени доступа к DRAM (при неверном предсказании префетчера)
  - пропускной способности L1 кэша
  - времени поиска страницы при промахе в TLB



Формат конфигурации теста

config.json

```
cpu_architecture: [value]
cpu_part: [value]
mode: [value]
hardware_configuration:
    — start_address: [value]
test_configuration:
    — use_mmu: [value]
    — warmup_iterations: [value]
    — stride: [value]
    — load_store_pattern: [value]
    — blocks_number: [value]
    — iterations: [value]
    — dependent_operations: [value]
    — unroll_loop: [value]
    — offset: [value]
```

Общие параметры

Конфигурация оборудования

Конфигурация теста

Пример: конфигурация теста на время доступа к L1 кэшу

```
cpu_architecture: risc-v
cpu_part: memory_subsystem
mode: memory_pass
hardware_configuration:
    - start_address: 0
test_configuration:
    - use_mmu: false
    - warmup_iterations: 10
    - stride: 64
    - load_store_pattern: 1
    - blocks_number: 128
    - iterations: 100
    - dependent_operations: true
    - unroll_loop: false
    - offset: 0
```

Положим размер кэш-линии L1 кэша равным 64

Проверка корректности тестов

Сгенерированы тесты на

- время доступа к L1 кэшу
- время доступа к L2 кэшу
- время доступа к L3 кэшу
- пропускную способность L1 кэша
- время поиска страницы при промахе в TLB
  (L1 TLB miss latency – L1 latency)
- время доступа к DRAM

Проверка корректности тестов

Сгенерированы тесты на

- время доступа к L1 кэшу
- время доступа к L2 кэшу
- время доступа к L3 кэшу
- пропускную способность L1 кэша
- время поиска страницы при промахе в TLB
  (L1 TLB miss latency – L1 latency)
- время доступа к DRAM

Проверка тестов осуществляется в три этапа:

1) Тест запускается и отрабатывает – на любом симуляторе
2) Тест выдает ожидаемый результат – на функциональном симуляторе. Сравнение производится с теоретически ожидаемыми результатами и с результатами существующих эталонных бенчмарков
3) Тест выдает ожидаемый результат – на FPGA

Проверка корректности тестов

Сгенерированы тесты на

- ✓ время доступа к L1 кэшу
- ✓ время доступа к L2 кэшу
- ✓ время доступа к L3 кэшу
- ✓ пропускную способность L1 кэша
- ✓ время поиска страницы при промахе в TLB
  (L1 TLB miss latency – L1 latency)
- ✗ время доступа к DRAM

Проверка тестов осуществляется в три этапа:

1) Тест запускается и отрабатывает – на любом симуляторе
2) Тест выдает ожидаемый результат – на функциональном симуляторе. Сравнение производится с теоретически ожидаемыми результатами и с результатами существующих эталонных бенчмарков
3) Тест выдает ожидаемый результат – на FPGA

Проверка корректности тестов

Сгенерированы тесты на

- ✓ время доступа к L1 кэшу
- ✓ время доступа к L2 кэшу
- ✓ время доступа к L3 кэшу
- ✓ пропускную способность L1 кэша
- ✓ время поиска страницы при промахе в TLB
  (L1 TLB miss latency – L1 latency)
- ✗ время доступа к DRAM

Результат получился сильно занижен

Проверка тестов осуществляется в три этапа:

1) Тест запускается и отрабатывает – на любом симуляторе
2) Тест выдает ожидаемый результат – на функциональном симуляторе. Сравнение производится с теоретически ожидаемыми результатами и с результатами существующих эталонных бенчмарков
3) Тест выдает ожидаемый результат – на FPGA

Проблема и решение

- Префетчер DRAM легко распознает линейный паттерн доступа

Проблема и решение

- Префетчер DRAM легко распознает линейный паттерн доступа
- Из-за этого еще до наступления load-инструкций начинается их спекулятивное исполнение

Проблема и решение

- Префетчер DRAM легко распознает линейный паттерн доступа
- Из-за этого еще до наступления load-инструкций начинается их спекулятивное исполнение
- Необходимо это избежать – будем “запутывать” префетчер рандомным доступом

Проблема и решение

- Префетчер DRAM легко распознает линейный паттерн доступа
- Из-за этого еще до наступления load-инструкций начинается их спекулятивное исполнение
- Необходимо это избежать – будем “запутывать” префетчер рандомным доступом
- Для этого в инструмент была добавлена возможность вместо константного шага (stride) указать функцию преобразования номера шага в его величину

Проблема и решение

- Префетчер DRAM легко распознает линейный паттерн доступа
- Из-за этого еще до наступления load-инструкций начинается их спекулятивное исполнение
- Необходимо это избежать – будем “запутывать” префетчер рандомным доступом
- Для этого в инструмент была добавлена возможность вместо константного шага (stride) указать функцию преобразования номера шага в его величину
- Теперь вместо последовательного доступа мы получим (псевдо) рандомный

Проблема и решение

- Префетчер DRAM легко распознает линейный паттерн доступа
- Из-за этого еще до наступления load-инструкций начинается их спекулятивное исполнение
- Необходимо это избежать – будем “запутывать” префетчер рандомным доступом
- Для этого в инструмент была добавлена возможность вместо константного шага (stride) указать функцию преобразования номера шага в его величину
- Теперь вместо последовательного доступа мы получим (псевдо) рандомный

✔ Результаты такого теста на время доступа к DRAM оказались корректными.

Заключение

В рамках работы над производственной практикой были выполнены следующие задачи.

- Определен и обоснован формат конфигурации теста
- Реализована генерация тестов на подсистему памяти RISC-V CPU с поддержкой всех необходимых параметров конфигурации
- Проверены средствами моделирования сгенерированные бенчмарки для некоторых популярных сценариев

Заключение

В рамках работы над производственной практикой были выполнены следующие задачи.

- Определен и обоснован формат конфигурации теста
- Реализована генерация тестов на подсистему памяти RISC-V CPU с поддержкой всех необходимых параметров конфигурации
- Проверены средствами моделирования сгенерированные бенчмарки для некоторых популярных сценариев

Комментарии

- Преимущество инструмента состоит в гибкости конфигурации теста, возможности точно задавать желаемую нагрузку
- С помощью инструмента удалось сгенерировать новые тесты, которых нет в эталонных наборах бенчмарков. Например, тесты с разными паттернами доступа и тесты с невыровненным доступом

БУДУЩЕЕ
В НАШИХ
РУКАХ

Приложение 1. Требования на инструмент

Исходя из того, какие бенчмарки были бы полезны на практике, на инструмент накладываются следующие **требования:**

- Шаблон доступа к памяти (последовательность операций load и store в определенном порядке) является конфигурируемым параметром
- Предусмотрена возможность указать, будут ли инструкции зависимыми или нет
- Есть возможность генерирования как тестов, взаимодействующих с памятью напрямую, так и тестов, использующих MMU (на данном этапе достаточно будет транслировать виртуальные адреса в физические как 1:1)
- Возможно генерировать тесты как с выровненным, так и с невыровненным доступом к памяти
- Остальные классические для memory-тестов параметры тоже должны быть поддержаны

Приложение 2. Структура сгенерированного теста

```
/*
This test was created for the following configuration:
* cpu_part: memory_subsystem
* cpu_architecture: risc-v
* mode: None
* hardware_configuration:
      - start_address: 0
* test_configuration:
      - dependent_operations: True
      ...
*/

#include <stdint.h>  // custom baremetal implementation!
#include <stdio.h>   // custom baremetal implementation!
```

Приложение 2. Структура сгенерированного теста

```
__attribute__ ((section(".page0x32000"), aligned(0x1000), used))
static uint8_t page0x32000[] = {
        0, 49, 3, 0, 0, 0, 0, 0, 0, 0, ...
};

__attribute__ ((section(".page0x33000"), aligned(0x1000), used))
static uint8_t page0x33000[] = {
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
};

__attribute__ ((section(".page0x34000"), aligned(0x1000), used))
static uint8_t page0x34000[] = {
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
};

...
```

+ линкер-скрипт

Приложение 2. Структура сгенерированного теста

Если используется MMU

```
__attribute__ ((section(".page0x143000"), aligned(0x1000), used))
static uint64_t pgd[] =  {
        0x51001, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, ...
};

__attribute__ ((section(".page0x144000"), aligned(0x1000), used))
static uint64_t pmd0x144000[] =  {
        0x51401, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, ...
};

__attribute__ ((section(".page0x145000"), aligned(0x1000), used))
static uint64_t pte0x145000[] =  {
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, ...
};
```

Приложение 2. Структура сгенерированного теста

```
__attribute__((always_inline)) static inline void PRELIMINARY_ACTION() {
        // init some registers
        // PGD address -> satp
}

__attribute__((always_inline)) static inline void POST_ACTION() {
        ...
}
```

Приложение 2. Структура сгенерированного теста

```
__attribute__((always_inline)) static inline void ONE_ITERATION() {
    asm volatile(
        "ld x9, -2048(x31);"
        "sd x10, -2032(x31);"
        "ld x11, -2016(x31);"
        "ld x12, -2000(x31);"
        "sd x13, -1984(x31);"
        "ld x14, -1968(x31);"
        "ld x15, -1952(x31);"
       ...
    );
}
```

Приложение 2. Структура сгенерированного теста

```
__attribute__((always_inline)) static inline void HOTSPOT(uint64_t* result) {
    asm volatile(
        "li x7, 110;"
        "li x8, 100;"
        "rdcycle x5;"
        "loop: addi x7, x7, -1;"
    );
    ONE_ITERATION();
    asm volatile(
        "bne x7, x8, new_iteration;"
        "rdcycle x5;"
        "new_iteration: bne x7, x6, loop;"
    );
    asm volatile(
        "rdcycle x6;"
        "sub %0, x6, x5;"
        : "=r"(*result)
        :
    );
}
```

unroll_loop = false

Приложение 2. Структура сгенерированного теста

```
__attribute__((always_inline)) static inline void HOTSPOT(uint64_t* result) {
    /* WARMUP ITERATIONS */
    ONE_ITERATION();
    ONE_ITERATION();
    ...
    /* EXPERIMENT ITERATIONS */
    asm volatile(
        "rdcycle x5;"
    );
    ONE_ITERATION();    // постоянно поступают новые инструкции,
    ONE_ITERATION();    // которых нет в кэше инструкций
    ...
    asm volatile(
        "rdcycle x6;"
        "sub %0, x6, x5;"
        : "=r"(*result)
        :
    );
}
```

unroll_loop = true

Приложение 2. Структура сгенерированного теста

```c
int main() {
    uint64_t result;

    PRELIMINARY_ACTION();
    HOTSPOT(&result);
    POST_ACTION();

    printf("Number of executed instruction blocks: 25600\n");
    printf("Number of cycles: %ld\n", result);
    printf("Average number of cycles per instruction block: %f\n", (double)result / 25600);

    return 0;
}
```