Syntacore™
Custom cores and tools

JVM для RISC-V

Владимир Кемпик

Апрель 2023

О себе

- JVM Инженер в Oracle 2013-2017
- JVM Инженер в Zulu JDK в Azul 2019-2022
  - JEP 391: macOS/AArch64 Port со-автор
- JVM Инженер в JetBrains Runtime 2022
- JVM Инженер в Syntacore с 2022

Компания Синтакор

Разработка и лицензирование процессорных ядер RISC-V

- Со-основатель и Premier учасник RISC-V International
- Основана в 2015
- Fabless

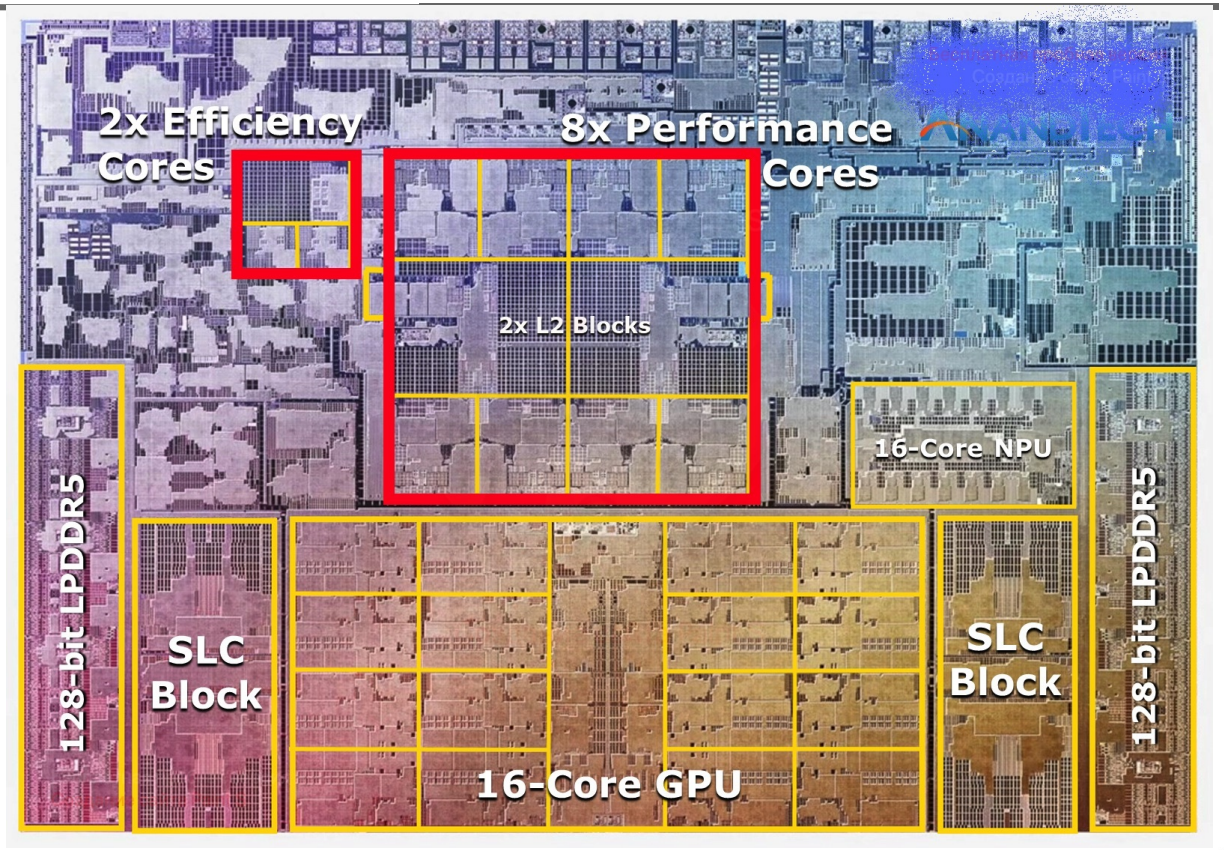
Процессорные блоки?

В чипе много разных блоков

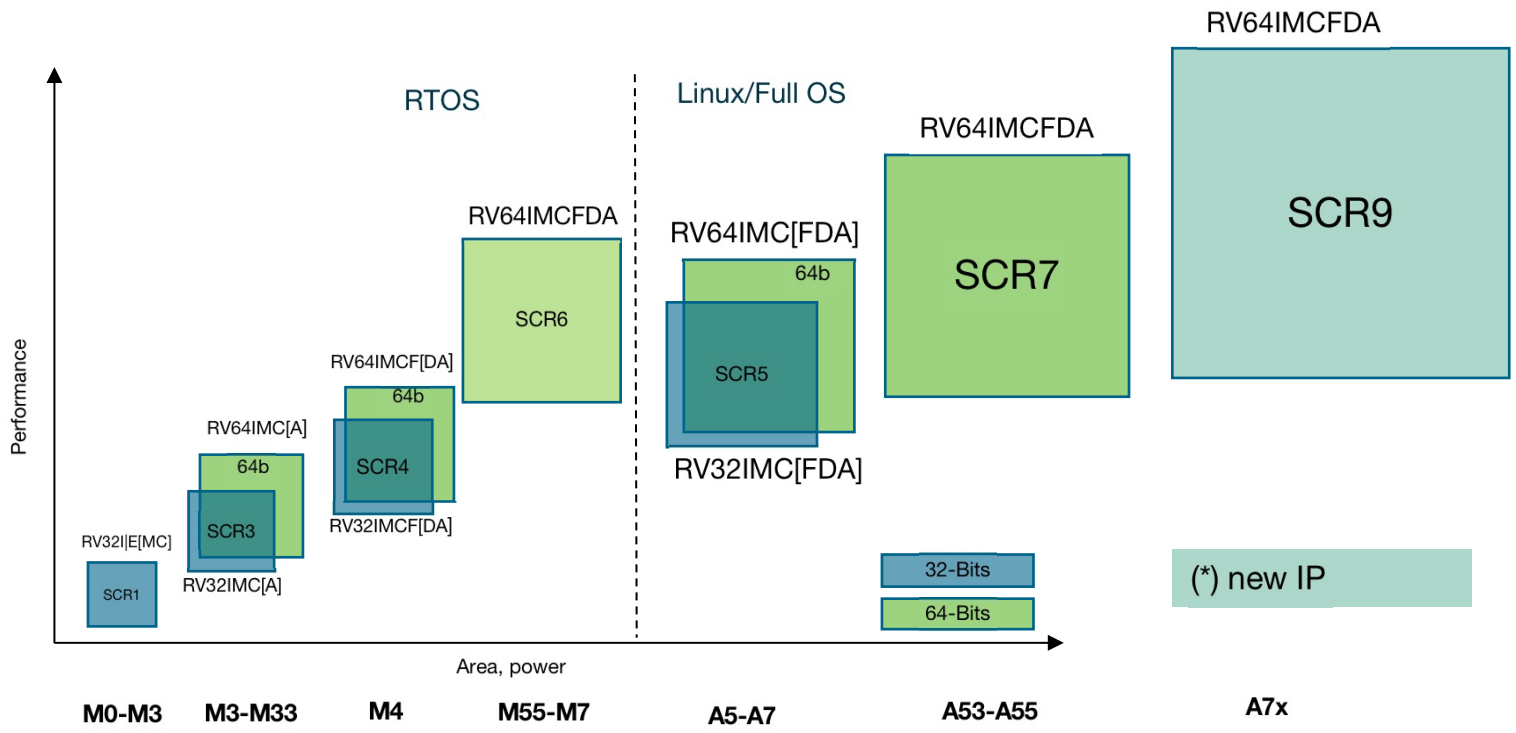
Часто от разных компаний

Syntacore создает CPU блоки

На картинке фото рандомного чипа, с CPU блоками выделенными красным



RISC-V CPU IP на примере SCRx



SCR1: Компактное микроконтроллерное ядро для встроенных систем, Open Source

SCR3: Микроконтроллерное ядро с режимами привилегий и MPU (32 или 64 бит)

SCR4: Микроконтроллерное ядро с высокопроизводительным FPU (32 или 64 бит)

SCR5: Ядро начального уровня APU с поддержкой Linux и SMP (32 или 64 бит)

SCR7: Высокопроизводительное 64-битное APU ядро с поддержкой SMP 8-16 ядер

SCR9: Высокопроизводительное 64-битное APU ядро серверного класса

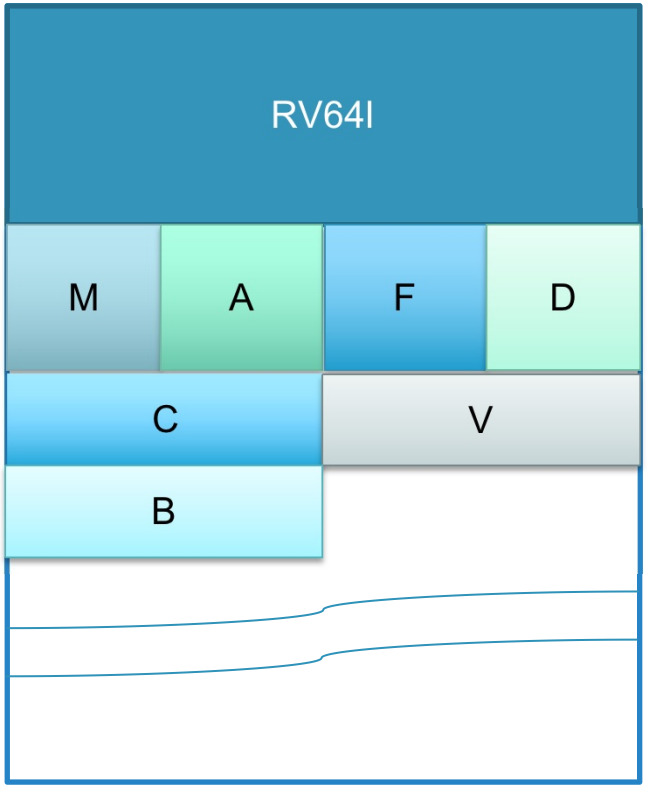
Про RISC-V

- RISC-V - свободная архитектура команд, родом из университета Беркли
- Развивается с 2010 года
- В 2015 для её развития был основан RISC-V Foundation
  - Syntacore - Founding member в RISC-V Foundation

ISA	База+расширения	Компактность	Четверная точность	32-бит адресация	64-бит адресация	128-бит адресация	GCC	LLVM	Linux	QEMU	JDK
SPARC			+	+	+		+	+	+	+	+*
OpenRISC				+	+		+	+	+	+	
RISC-V	+	+	+	+	+	+	+	+	+	+	+

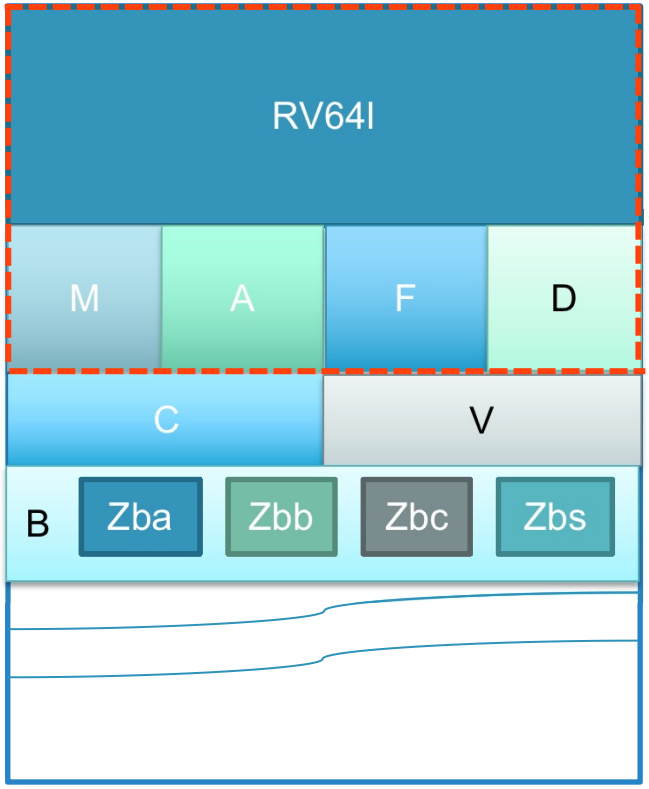
RISC-V расширяемость

- Говорим только про 64-битный вариант
- RV64I – набор базовых команд
- M, A, F, D, C, V, B… - расширения
- Multiplication, Atomic, Float, Double, Compressed, Vector, Bitmanip and more
- Поднаборы расширений начинаются с Z (например Zba) для стандартизованных, с X (например Xventana) для вендор специфичных



RISC-V расширяемость

- RV64IMAFD = RV64G (General Purpose), большинство ОС/софта требуют его (+ C), в т.ч. openJDK
- C – сжатые инструкции
- V – векторное расширение
- B – битовые манипуляции, разбили на несколько поднаборов – Zba, Zbb, Zbc, Zbs, ...

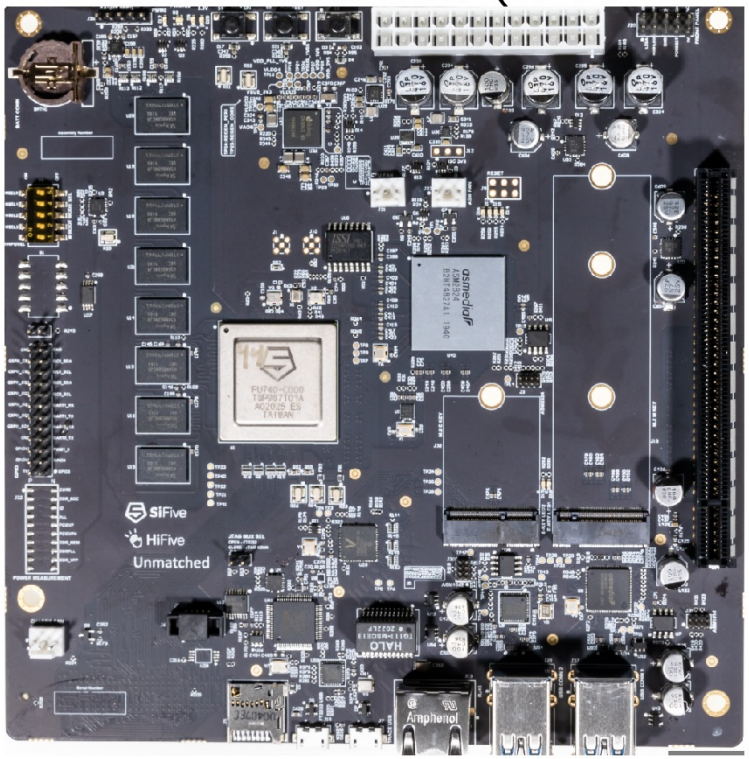


На чем запускать ?

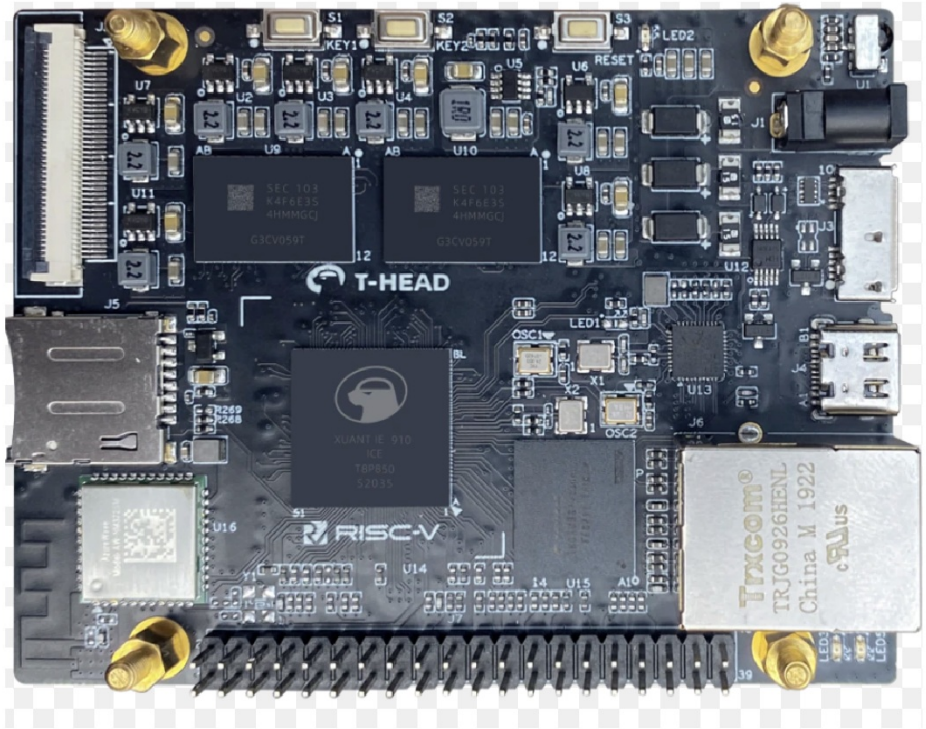
- qemu (системный эмулятор) версии 7.X поддерживает эмуляцию RV64GCV (1.0) + Zba + Zbb + Zbs + Zbc
- Может запускать Linux/risc-v и софт
- Qemu это своя особая микроархитектура, можно проверять корректность, но нельзя оценивать производительность
- Нужны реальные железки

RISC-V общедоступные платы RV64GC

Hifive Unmatched (и VisionFive 2)



Thead rvb-ice



JEP 422: Linux/RISC-V Port

- Вошел в OpenJDK 19
- Изначально поддерживал RV64GCVB, где C, V, B были опциональны и выключены по умолчанию
- Шаблонный интерпретатор, C1/C2 JIT, все сборщики мусора, в т.ч. ZGC и ShenandoahGC

JEP 422: Linux/RISC-V Port

- Часть интринсиков Vector-only
- Часть функций jit-а Zbb-only (расширеные битовые манипуляции), часть Zba-only (формирование адреса)
- Vector версии 1.0 и Zba/Zbb не найти в общедоступных платах (оценить производительность этой части кода пока невозможно)
- Часть preview функций отстает в поддержке от остальных платформ (например virtual threads)

Изменения в openJDK 20


```
  product(bool, UseRVV, false, EXPERIMENTAL, "Use RVV instructions")
  product(bool, UseRVC, false, EXPERIMENTAL, "Use RVC instructions")
  product(bool, UseZba, false, EXPERIMENTAL, "Use Zba instructions")
  product(bool, UseZbb, false, EXPERIMENTAL, "Use Zbb instructions")
+ product(bool, UseZbs, false, EXPERIMENTAL, "Use Zbs instructions")
+ product(bool, UseZfhmin, false, EXPERIMENTAL, "Use Zfhmin instructions")
+ product(bool, UseZic64b, false, EXPERIMENTAL, "Use Zic64b instructions")
+ product(bool, UseZicbom, false, EXPERIMENTAL, "Use Zicbom instructions")
+ product(bool, UseZicbop, false, EXPERIMENTAL, "Use Zicbop instructions")
+ product(bool, UseZicboz, false, EXPERIMENTAL, "Use Zicboz instructions")
+ product(bool, UseZihintpause, false, EXPERIMENTAL, "Use Zihintpause instructions")
```

Изменения в openJDK 20

К счастью есть профили, которые группируют наборы расширений

- RVA20U64 – обязательно наличие RV64GC + много расширений, которые не используется в OpenJDK. Опция -XX:+UseRVA20U64 включена по умолчанию в OpenJDK20
- RVA22U64 - +Zba +Zbb +Zbs +Zicboz (стирание кеша и памяти блоками) + много еще разных расширений
- RVA23U64 (draft) + Vectors

Things yet to come

J extension:

- Набор расширений для потенциального ускорения виртуальных машин (как JVM, не qemu/kvm)
- Интенсивно обсуждается в J комитете
  - Zjpm - Pointer masking: Модель памяти (SV39, SV48, SV57) требует каноничные адреса, но может пустить верхние биты на благо народного хозяйства ?

Предупреждение

Обзор архитектуры RISC-V и поддержки RISC-V в JVM закончен, далее будет хардкор, в т.ч. Ассемблер

Можно ли в продакшн?

- Необходимо оценить производительность
- На чем?
  - Есть несколько общедоступных плат
- С чем сравнивать?
  - Порт был основан на aarch64, давайте с ним и сравнивать. Только без читов в виде NEON (SIMD для ARM, аналог SSE)
- Как?

Можно ли в продакшн?

- Необходимо оценить производительность
- На чем?
  - Есть несколько общедоступных плат
- С чем сравнивать?
  - Порт был основан на aarch64, давайте с ним и сравнивать. Только без читов в виде NEON (SIMD для ARM, аналог SSE)
- Как?
  - JMH + набор микротестов из openJDK

JMH + microtests

- Java Microbenchmark Harness — набор библиотек для тестирования производительности небольших функций
- Тесты из jdk/test/micro/org/openjdk
- Все собрать в один benchmarks.jar
- java -jar benchmarks.jar -prof perfasm bench.vm > results.txt
  - для замера прогон без профайлера

JMH + microtests

- Подождать 3 дня, и это только bench.vm, все тесты займут больше двух недель
  - не зависит от платформы, это методика тестирования
  - на медленных платформах чуть дольше (+ до 50%)
- Сравниваем cortex-A72 (1.5 ghz) и risc-v (1.2 ghz)
- Результаты нормируем по мгц (приводим результаты к одной частоте)

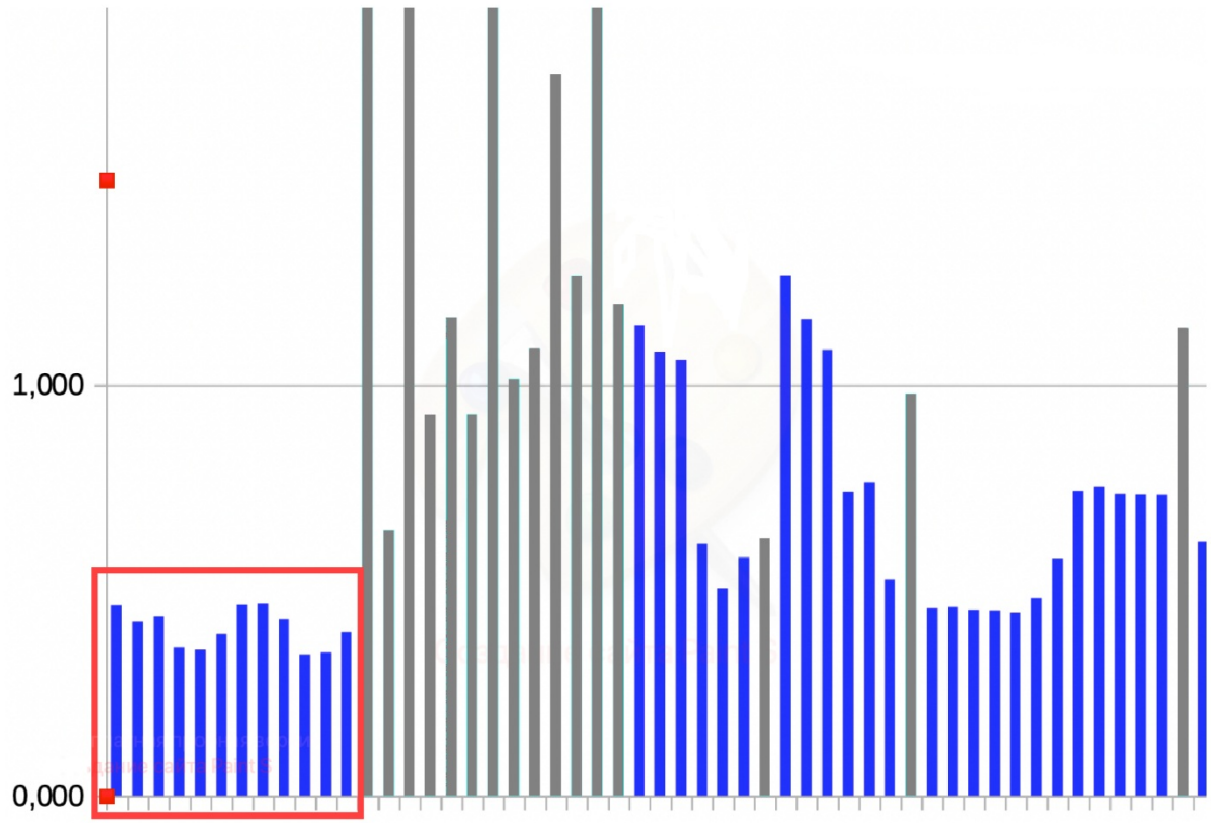
Результаты

Больше 400 тестов, покажу только часть

1 - результат A72

Скорость ОЗУ играет существеную роль

Скорость ОЗУ влияет слабо

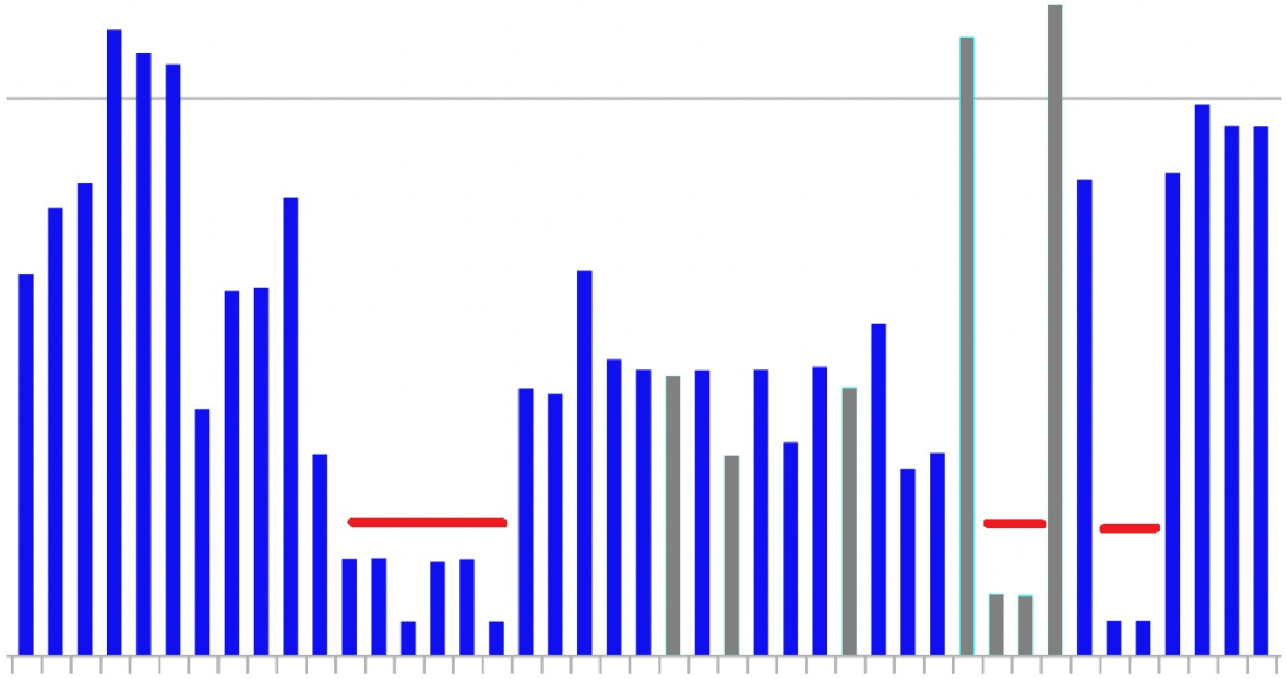


Результаты ч. 2

1 - результат A72

Скорость ОЗУ играет существеную роль

Скорость ОЗУ влияет слабо



Case study #1

- JDK-8296602: RISC-V: improve performance of copy_memory stub (used in System.arraycopy)

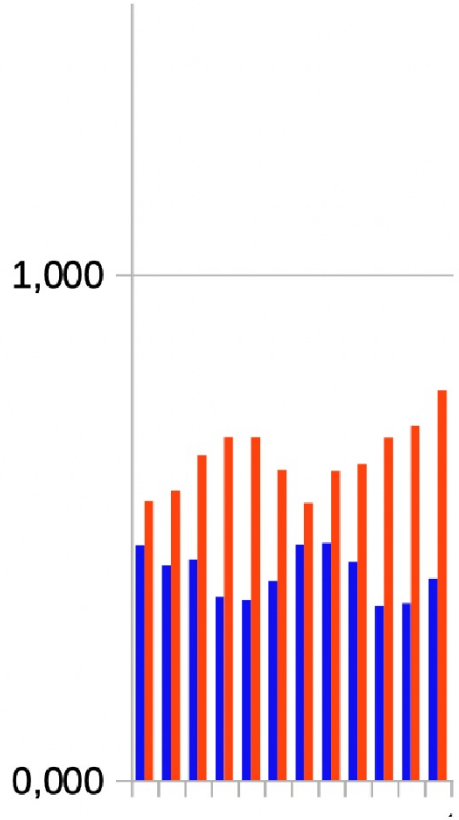
```
 0.49%  0x0000003fad94bc10:   addi   t5,t5,-8
        0x0000003fad94bc14:   addi   t6,t6,-8
 0.83%  0x0000003fad94bc18:   ld     a6,0(t5)
62.71%  0x0000003fad94bc1c:   sd     a6,0(t6)
30.06%  0x0000003fad94bc20:   addi   a5,a5,-8
 0.00%  0x0000003fad94bc24:   addi   a7,a5,-8
 0.59%  0x0000003fad94bc28:   bgez   a7, 0x03fad94bc10
```

Case study #1


```
__ ld(tmp3, Address(src));
__ ld(tmp4, Address(src, 8));
__ ld(tmp5, Address(src, 16));
__ ld(tmp6, Address(src, 24));
__ sd(tmp3, Address(dst));
__ sd(tmp4, Address(dst, 8));
__ sd(tmp5, Address(dst, 16));
__ sd(tmp6, Address(dst, 24));
```

Было

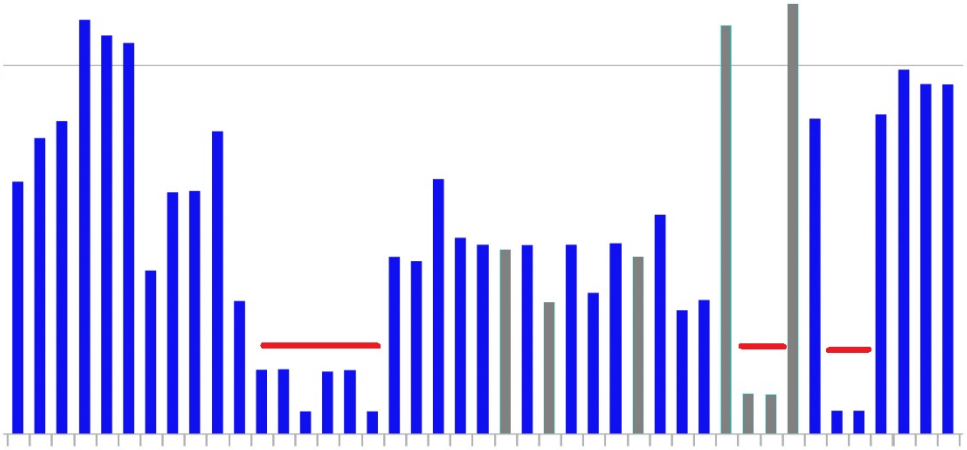
Стало



Case study #2

- Floating single/double min/max SLOOOOOW
- Это интринсик
- Если один из аргументов NaN (Not a Number), то вернуть NaN

- В risc-v, fmin/fmax, если один из аргументов NaN то возвращается другой аргумент
- Нужна дополнительная логика



Case study #2

o.o.b.v.compiler.MaxMinOptimizeTest.fAdd	0,853
o.o.b.v.compiler.MaxMinOptimizeTest.fMax	0,061
o.o.b.v.compiler.MaxMinOptimizeTest.fMin	0,061
o.o.b.v.compiler.MaxMinOptimizeTest.fMul	0,865

```
private float fAddBench(float a, float b) {
    return Math.max(a, b) + Math.min(a, b);
}

private float fMulBench(float a, float b) {
    return Math.max(a, b) * Math.min(a, b);
}
private float fMaxBench(float a, float b) {
    return Math.max(Math.max(a, b), Math.min(a, b));
}
```

Case study #2

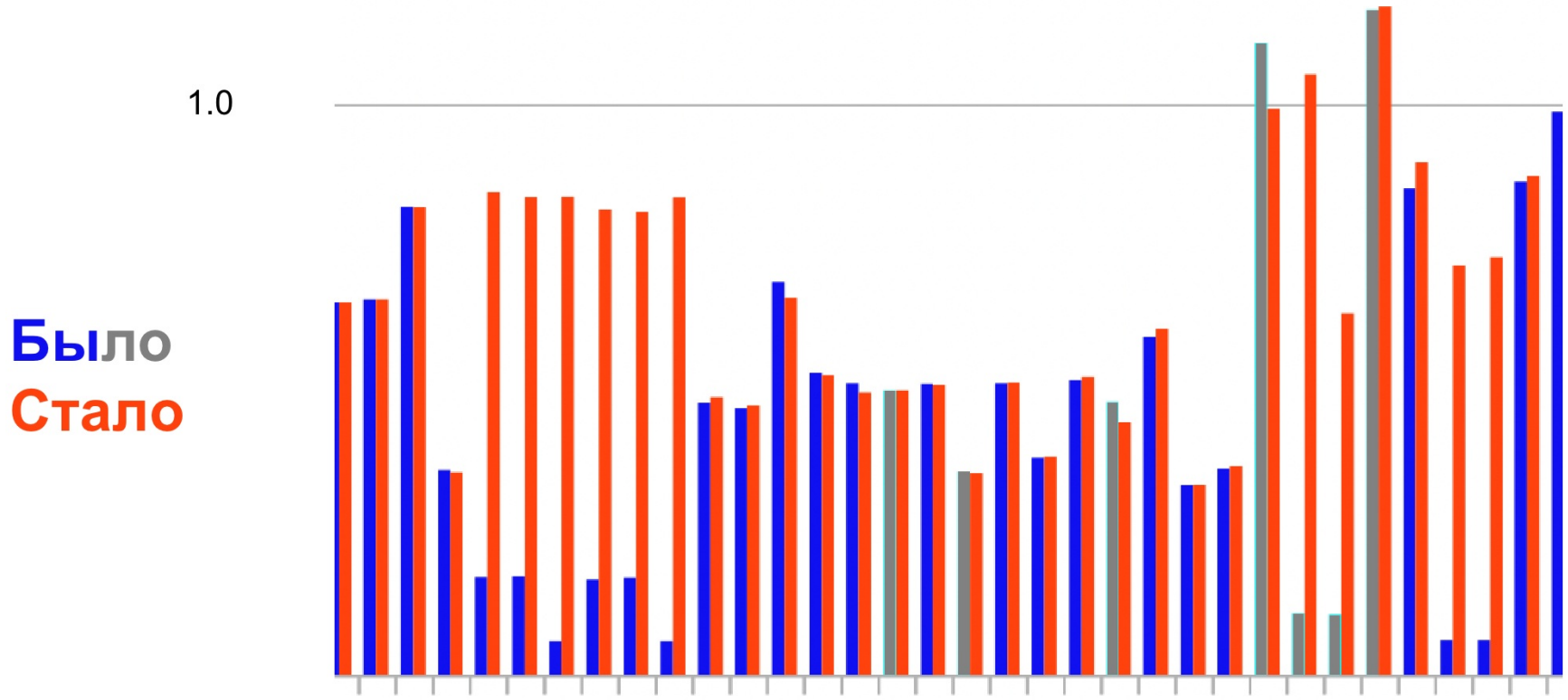
- JDK-8297359: RISC-V: improve performance of floating Max Min intrinsics

```
14.92%   0x0000003fc540d008:   fmin.d   ft2,ft1,ft0
проверка NaN:
16.44%   0x0000003fc540d00c:   flt.d    zero,ft1,ft0
 4.56%   0x0000003fc540d010:   frflags  t0
14.81%   0x0000003fc540d014:   beqz     t0,0x0003fc540d01c
         0x0000003fc540d018:   fadd.d   ft2,ft1,ft0
```

Case study #2


```
 0.01%    0x0000003fcd40c794:   fclass.d    t0,ft1
          0x0000003fcd40c798:   fclass.d    t1,ft0
 2.39%    0x0000003fcd40c79c:   or    t0,t0,t1
 2.69%    0x0000003fcd40c7a0:   andi  t0,t0,768
 9.98%    0x0000003fcd40c7a4:   beqz  t0,  0x03fcd40c7b0
          0x0000003fcd40c7a8:   fadd.d      ft2,ft1,ft0
          0x0000003fcd40c7ac:   j     0x0000003fcd40c7b4
14.51%    0x0000003fcd40c7b0:   fmin.d      ft2,ft1,ft0
```

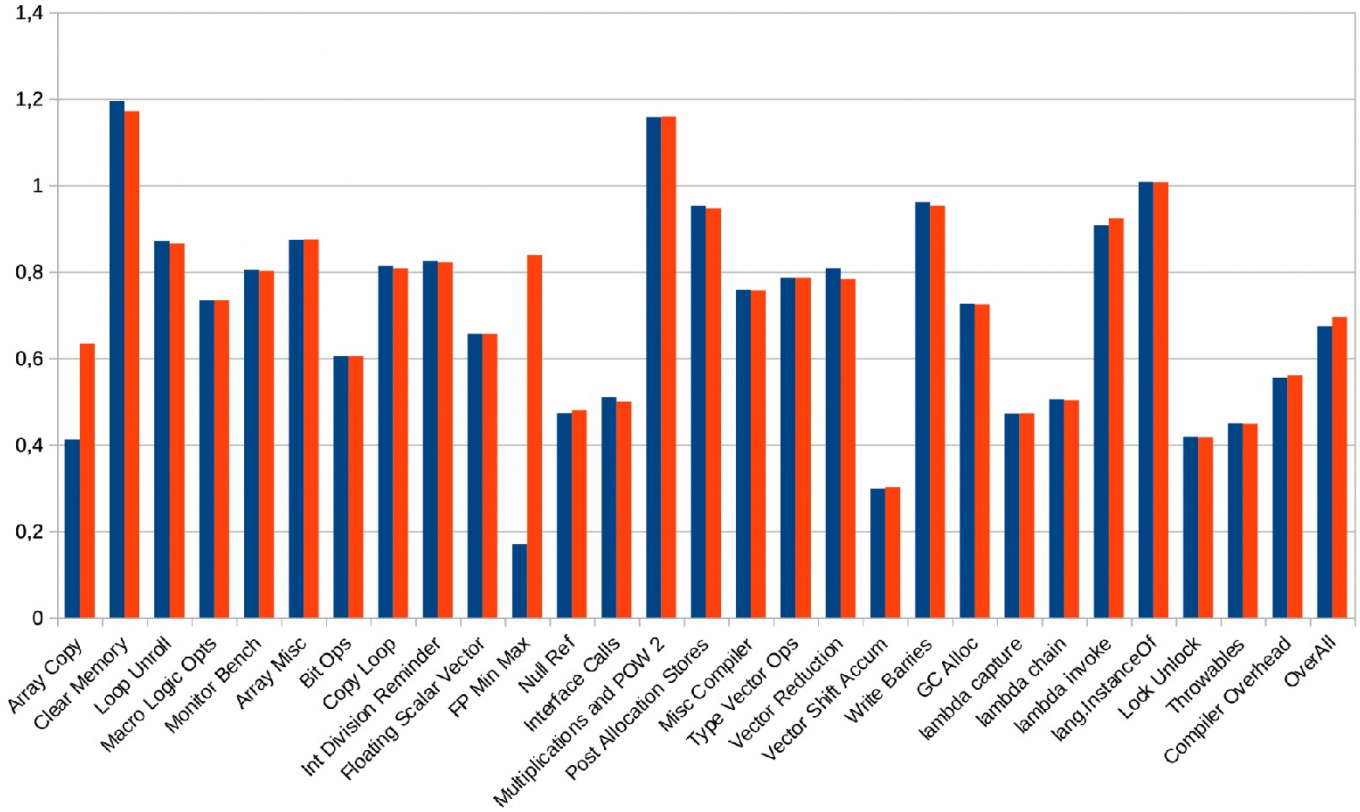
Case study #2



Медианные результаты

Было

Стало



Невыровненный доступ к памяти

Выровненный доступ (aligned):

0x100:

Натуральное выравнивание - адрес объекта кратен размеру объекта при обращении.

Невыровненный доступ (misaligned):

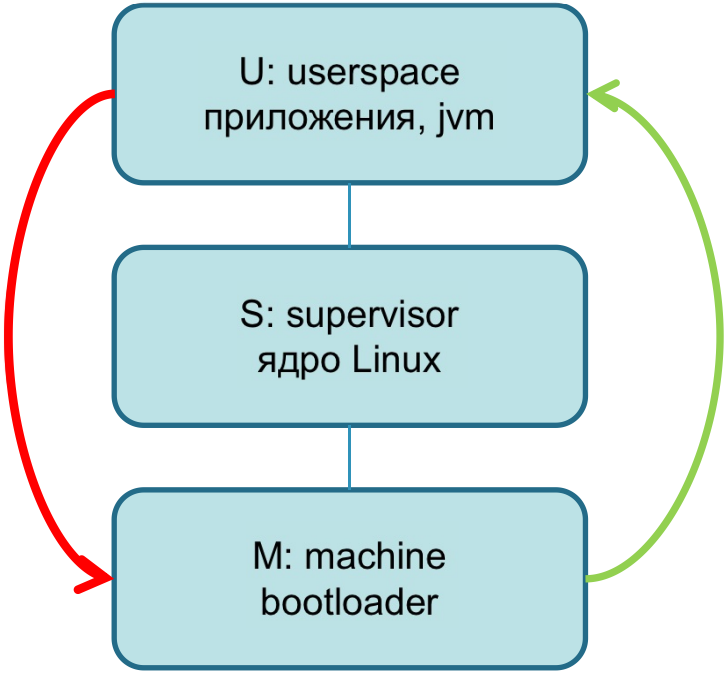
0x100:

Почему misaligned плохо

Некоторые процессоры поддерживают аппаратный misaligned доступ к памяти

Некоторые производят эмуляцию в M-mode: сотни дополнительных инструкций на один случай misaligned доступа

Уровни привелегий на RISC-V



perf идет на помощь


```
perf stat -e trp_lam,trp_sam,instructions java -version
...
Performance counter stats for 'java -version':

              127088      trp_lam (misaligned loads)
                 116      trp_sam (misaligned stores)
           377233009      instructions

        4.711040972 seconds time elapsed
```

Соотношение trp_lam : instruction ~ 1:3000

perf идет на помощь

Заменим perf stat на perf record, произойдет запись результатов в файл, анализируем только событие trp_lam

Посмотрим результаты с помощью perf report:

```
8.85%  java            [JIT] tid 51229    [.] 0x0000003f88e5f9ac
7.76%  java            [JIT] tid 51229    [.] 0x0000003f88e50230
7.17%  java            [JIT] tid 51229    [.] 0x0000003f88e63208
6.68%  java            [JIT] tid 51229    [.] 0x0000003f88e61b04
6.68%  java            [JIT] tid 51229    [.] 0x0000003f88e62a08
5.60%  java            [JIT] tid 51229    [.] 0x0000003f88e50298
4.75%  java            [JIT] tid 51229    [.] 0x0000003f88e6684c
```

perfasm.jar идет на помощь

- perfasm.jar - профилировщик из JMH, можно использовать отдельно от JMH
- Скачиваем здесь - https://builds.shipilev.net/perfasm/
- Запускаем java с перфом:

```
perf record -e trp_lam ./java -
XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly -
XX:+PrintNMethods -XX:+PrintNativeNMethods -
XX:+PrintSignatureHandlers -XX:+PrintAdapterHandlers -
XX:+PrintStubCode -XX:+PrintInterpreter -
XX:+LogCompilation -XX:LogFile=hotspot.log -version
```

perfasm.jar идет на помощь

Обрабатываем результаты профилирования perfasm-ом

```
java -jar perfasm.jar
"event=trp_lam;delay=1000;length=10000000;hotThresho
ld=0.01" hotspot.log perf.data
```

Результаты работы perfasm.jar


```
...[Hottest Regions]...
13.13%          interpreter  sipush  17 sipush
 7.58%          interpreter  goto  167 goto
 6.39%          interpreter  getstatic  178 getstatic
 6.07%          interpreter  fast_aaccess_0  222 fast_aaccess_0
 5.45%          interpreter  invokestatic  184 invokestatic
 5.30%          interpreter  invokevirtual  182 invokevirtual
 5.20%          interpreter  return entry points
 4.95%          interpreter  return entry points
 4.19%          interpreter  ifne  154 ifne
```

Результаты работы perfasm.jar


```
....[Hottest Region 1]......
interpreter, sipush  17 sipush  (0 bytes)

           0x0000003f98e58568:   sd     zero,8(s4)
           0x0000003f98e5856c:   sd     a0,0(s4)
           0x0000003f98e58570:   j 0x0000003f98e58580
           0x0000003f98e58574:   addi   s4,s4,-8
           0x0000003f98e58578:   addw   a0,a0,zero
           0x0000003f98e5857c:   sd     a0,0(s4)
13.13%     0x0000003f98e58580:   lhu    a0,1(s6)
           0x0000003f98e58584:   srli   t1,a0,0x10
```

Пример фикса


```
- lhu(reg, Address(xbcp, bcp_offset));
+ lbu(t1, Address(xbcp, bcp_offset));
+ lbu(reg, Address(xbcp, bcp_offset + 1));
+ slli(reg, reg, 8);
+ add(reg, t1, reg);
```

Результаты изменений в JVM


```
java -Xint -version
```

Было	Стало
208036 trp_lam	0 trp_lam
real 0m5.003s	real 0m4.061s
user 0m4.672s	user 0m3.772s

Результаты изменений на железе

Одна итерация renaissance philosophers в режиме -Xint
Диапазон результатов 5 прогонов.

На плате с аппаратной поддержкой misaligned loads

Было 671-684 секунд

Стало 657-689 секунд

УРА, не стало хуже!

Результаты изменений на железе

Одна итерация renaissance philosophers в режиме -Xint
Диапазон результатов 5 прогонов.

На плате без аппаратной поддержкой misaligned loads

Было 2638-2663 секунд

Стало 1489-1504 секунд

Выводы*

- В продакшн пока рано, нет подходящих систем RV64GCVB
  - Но они уже анонсированы (MIPS eVocore P8700, Ventana Veyron v1)
- Слабые места - ассемблерный код, написаный человеком
- Производительность JVM на Risc-V пока немного не дотягивает до ARM64, но активно ведется работа по улучшению JVM и написанию новых интринсиков, например SHA*/MD5 на базе Vector Crypto Extensions (пока в драфте)

* - личное мнение автора доклада

Выводы

- Готовить свои стеки под RISC-V стоит начать уже сейчас
  - Если найдете проблемы, похожие на показанные здесь, пишите в riscv-port-dev@openjdk.org
- Проверять лучше на реальном железе, qemu подходит в основном для проверки корректности работы

Полезные ссылки

- JDK17 (LTS) :

  https://github.com/syntacore/syntaj17

  https://github.com/openjdk/riscv-port-jdk17u

- JDK11 (LTS):

  sometime soon:

  https://github.com/openjdk/riscv-port-jdk11u

  now:

  https://github.com/isrc-cas/bishengjdk-11-mirror

Постскриптум

We are growing.

Резюме можно прислать на recruiter@syntacore.com

Общие вопросы info@syntacore.com

Вопросы по jdk/risc-v - дискуссионная зона