OpenJDK on RISC-V Update

Fei Yang – PLCT Lab, ISCAS

yangfei@iscas.ac.cn

OpenJDK Reviewer & RISC-V Port Project Lead

https://openjdk.org/census#fyang

Project Goal

To provide first class Java support on RISC-V 64-bit architecture (RV64GCV)

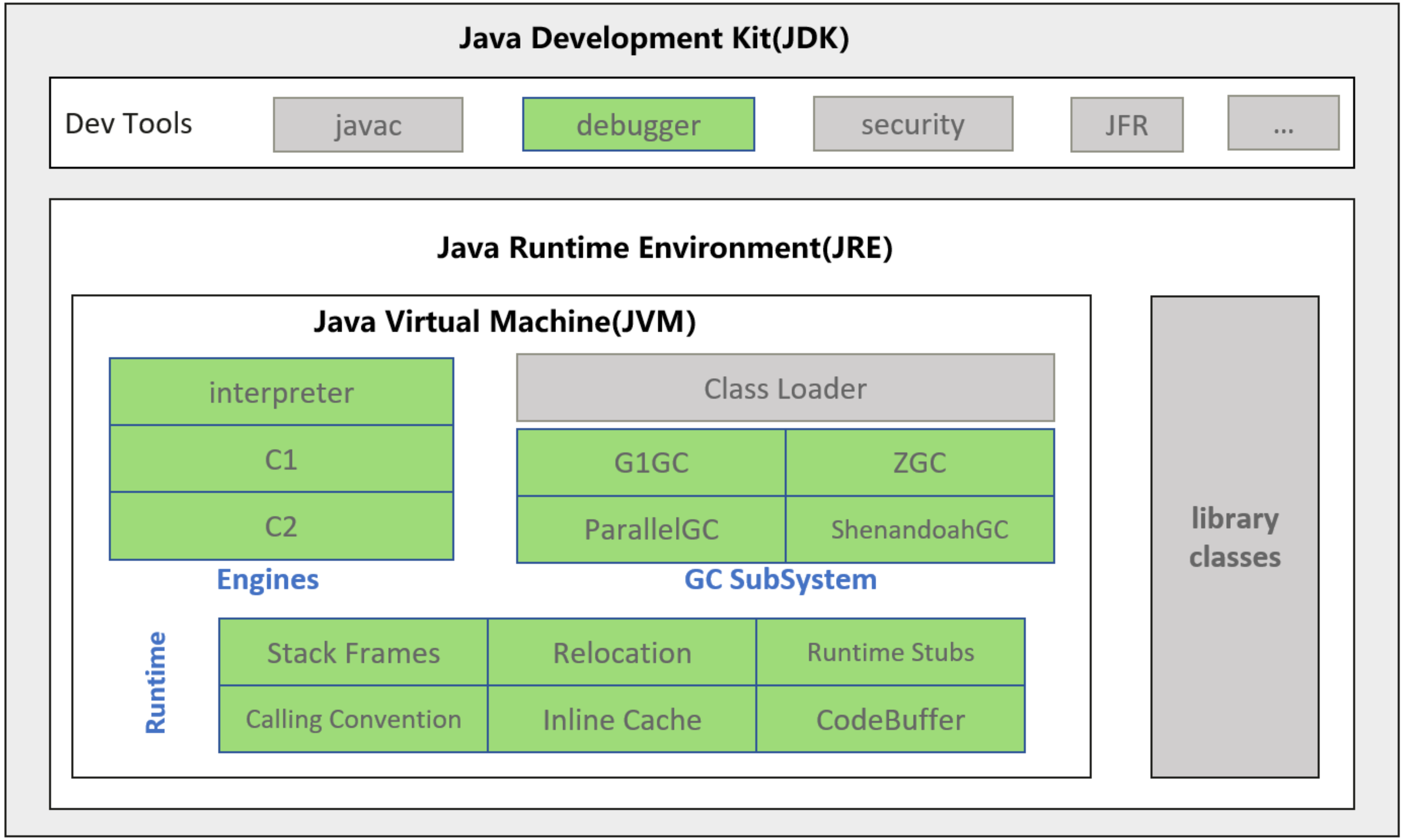
https://openjdk.org/projects/riscv-port

https://mail.openjdk.org/pipermail/riscv-port-dev

Project Timeline

- Mar 2022 - Upstreamed to OpenJDK mainline
  - https://github.com/openjdk/jdk/pull/6294 (8276799: Implementation of JEP 422: Linux/RISC-V Port)
  - Available in JDK 19 - 23 (including JDK 21 LTS)
- Jul 2023 - Backported to OpenJDK 17u master
  - https://github.com/openjdk/jdk17u-dev/pull/1427 (8276799: Implementation of JEP 422: Linux/RISC-V Port)
  - Available in JDK 17.0.9+ release
- Feb 2024 - Backported to riscv-port-11u project repo
  - https://github.com/openjdk/riscv-port-jdk11u/pull/3 (8276799: Implementation of JEP 422: Linux/RISC-V Port)
  - Placeholder: https://github.com/openjdk/riscv-port-jdk8u

Overview: A full-featured Java port



New features supported

- New RISC-V extensions support
- New Lightweight Locking
- Virtual Threads
- Vector API (Incubator)
- Foreign Function & Memory API
- Generational ZGC & ShenandoahGC
- Various performance tunings

* Contributions by: ISCAS, Rivos, Huawei, Alibaba, …
* 300+ upstream PR reviewed & merged, mostly RISC-V related.

New RISC-V extensions supported

Zba
Zbb
Zbs

Zvbb
Zvkn
Zvkned

Zfh
Zvfh
Zcb

Zic64b
Zicbom
Zicboz

Zacas
Ztso
Zihintpause

I	M	A	F	D	C	V

New RISC-V extensions supported

RISC-V Hardware Probing Syscall:

```
struct riscv_hwprobe {
    __s64 key;
    __u64 value;
};

long sys_riscv_hwprobe(struct riscv_hwprobe *pairs, size_t pair_count,
                       size_t cpu_count, cpu_set_t *cpus,
                       unsigned int flags);
```

- https://www.kernel.org/doc/html/v6.5-rc2/riscv/hwprobe.html

New RISC-V extensions supported

Vendor-specific tunning:

```
void VM_Version::vendor_features() {
  if (!mvendorid.enabled()) {
    return;
  }
  switch (mvendorid.value()) {
    case RIVOS:
    rivos_features();
    break;
    default:
    break;
  }
}
```

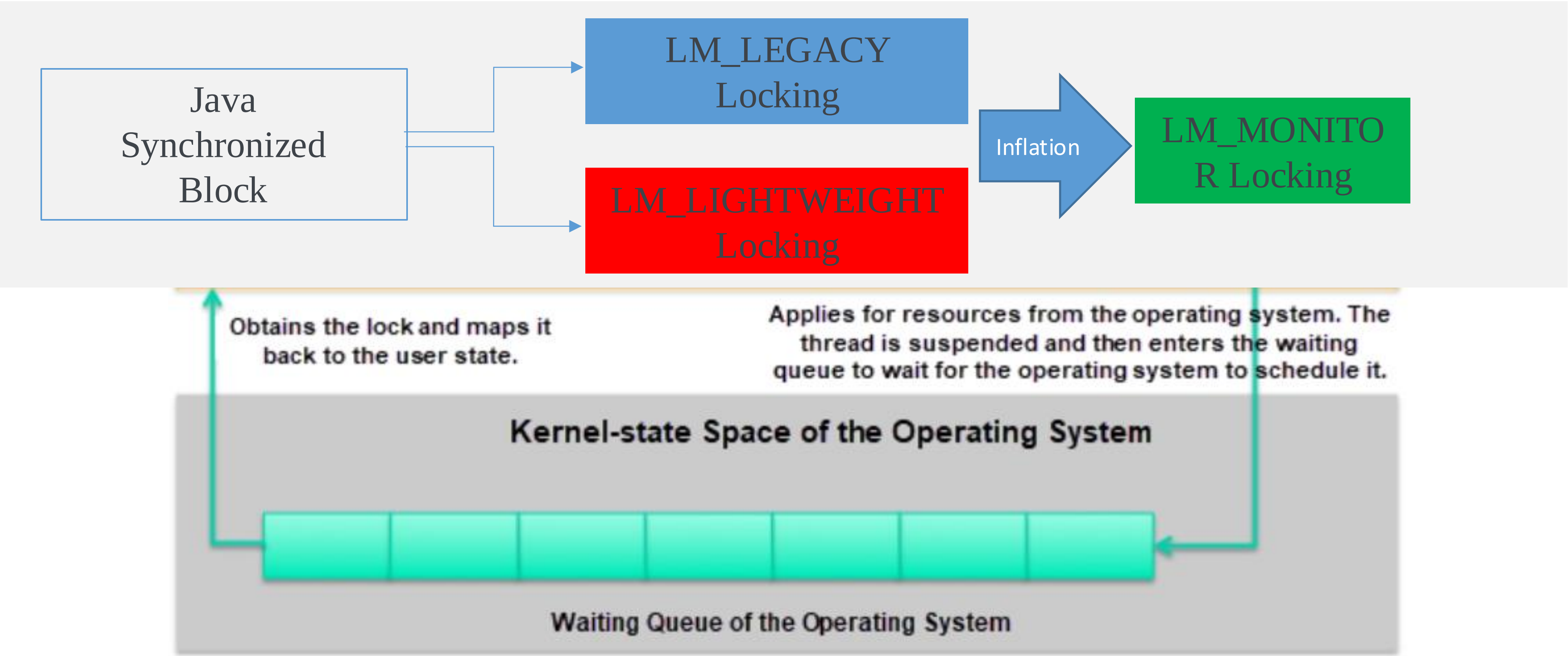
```
void VM_Version::rivos_features() {
  // Enable common features not dependent on marchid/mimpid.
  ext_Zicbom.enable_feature();
  ext_Zicboz.enable_feature();
  ext_Zicbop.enable_feature();

  // If we running on a pre-6.5 kernel
  ext_Zba.enable_feature();
  ext_Zbb.enable_feature();
  ext_Zbs.enable_feature();

  ext_Zcb.enable_feature();

  ext_Zfh.enable_feature();

  ext_Zicboz.enable_feature();
  ext_Zicsr.enable_feature();
  ext_Zifencei.enable_feature();
  ext_Zic64b.enable_feature();
  ext_Ztso.enable_feature();
```

- https://github.com/openjdk/jdk/blob/master/src/hotspot/os_cpu/linux_riscv/vm_version_linux_riscv.cpp#L229

New Lightweight Locking

Three different locking modes in latest HotSpot JVM (-XX:LockingMode):

- LM_MONITOR : Heavy monitors only (the same as current -XX:+UseHeavyMonitors);
- LM_LEGACY : Legacy stack-locking, with monitors as second tier;
- LM_LIGHTWEIGHT : New lightweight locking, with monitors as second tier (current default);

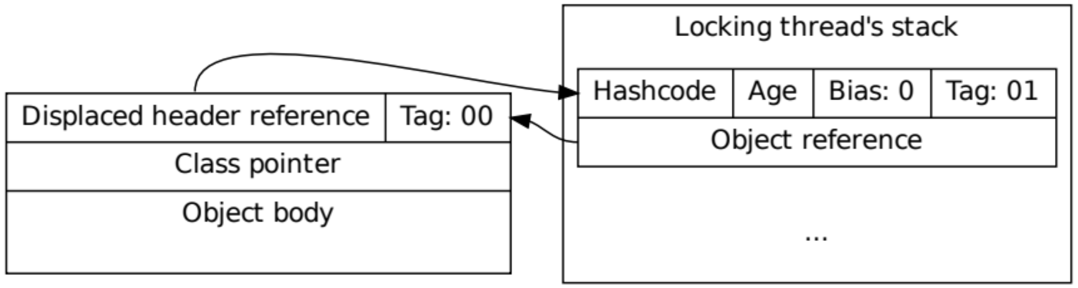


New Lightweight Locking

Java Object Header

```
+------------------+      +------------------+
| Lock-word        |      | Lock-word        |
+------------------+      +------------------+
| Class pointer    |      | Class pointer    |
+------------------+      +------------------+
| Field 1          |      | Array-Length     |
+--------+---------+      +--------+---------+
| Field 2| Field 3 |      | Elem 1 | Elem 2  |
+--------+---------+      +--------+---------+
| etc              |      | etc              |
```

Problem with LM_LEGACY locking mode



Example of a locked object (stacking locking mode)

Lock-word states and layout

Locking: The 3 lowest bits are used for locking, and can take the following combinations:

- `[ ptr             | 00 ]` Locked, the upper bits interpreted as a pointer point to real header on stack
- `[ header | 0 | 01 ]` Unlocked, upper bits are regular object header
- `[ ptr             | 10 ]` Monitor, the upper bits point to inflated lock, header is swapped out
- `[ 0 ...... 0 | 00 ]` Inflating in progress
- `[ ptr             | 11 ]` Forwarded, used by GC to indicate that upper bits point to forwarded object, which also contains the real header

New Lightweight Locking

The solution (Project Lilliput):

- Still CAS the lowest two header bits to ’00’ to indicate 'fast-locked' but does not overload the upper bits with a stack-pointer. Instead, it pushes the object-reference to a thread-local lock-stack.

- The lock-stack is fixed size, currently with 8 elements. Check for overflow in the fast-paths and when the lock-stack is full, take the slow-path, which would inflate the lock to a monitor. That case should be very rare.

- JDK-8291555: Implement alternative fast-locking scheme

  https://github.com/openjdk/jdk/pull/10907

- JDK-8319796: Recursive lightweight locking

  https://github.com/openjdk/jdk/pull/17554

Virtual Threads

Virtual threads implementation (Project Loom):

- Built on continuations, a lower level construct implemented in HotSpot JVM

- A virtual thread wraps a task in a continuation:
  - Freeze: The continuation yields when the task needs to block
  - Thaw: The continuation is continued when the task is ready to continue

- Scheduler executes the tasks for virtual threads on a pool of carrier threads
  - M : N threading model
  - The scheduler is a java.util.concurrent.ForkJoinPool
    - FIFO mode
    - Parallelism defaults to the number of hardware threads

- JEP 425: Virtual Threads

  https://openjdk.org/jeps/425 / https://git.openjdk.java.net/jdk/pull/8166

- JDK-8286301: Port JEP 425 to RISC-V

  https://github.com/openjdk/jdk/pull/10917

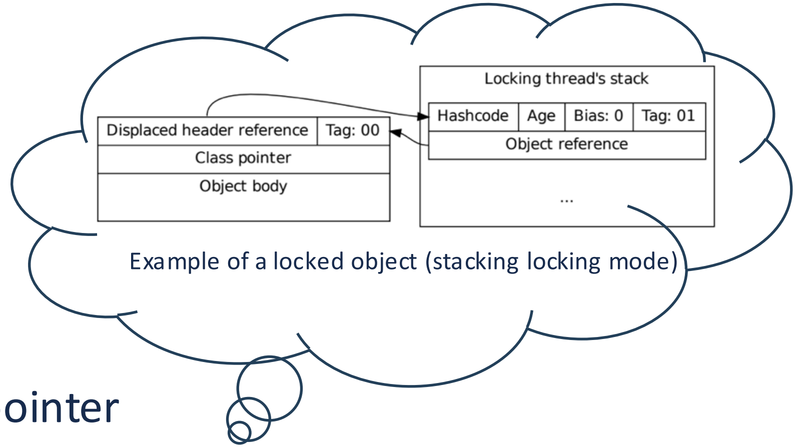
Virtual Threads

Biggest pain point at this time (Java monitors / virtual thread pinning issue):

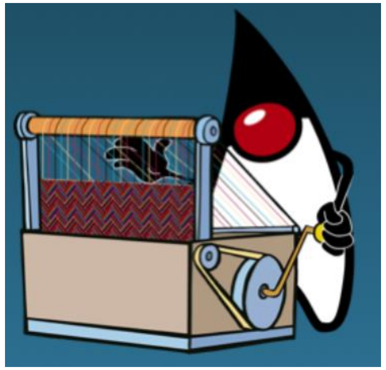
- LM_LEGACY:
    - Lock record per monitor in Java frame

    - Fast case installs pointer to lock record in object header (stack-locked)
        - Yield/Freeze copies frames into heap, which invalidate installed pointer
        - Continue/Thaw may copy back to a different stack address

    - Slow case uses heavyweight monitor (inflated)
        - Inflated monitor has a list of waiting threads
        - Owner field is JavaThread that corresponds to the carrier thread



- LM_LIGHTWEIGHT
    - Each JavaThread has a lock stack
    - Uncontended case does push/pop object to the JavaThread's lock stack
    - Slow case is the same as LM_LEGACY, uses inflated monitors



Reference: https://fosdem.org/2024/schedule/event/fosdem-2024-3255-virtual-thread-s-next-steps

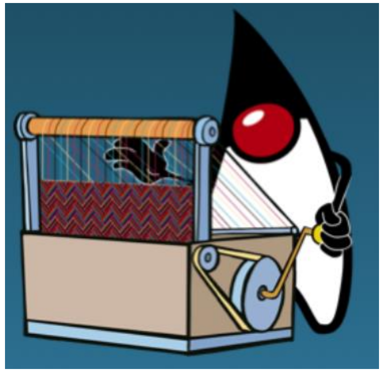
Virtual Threads

Allow yielding while holding monitor (WIP):

- Only the "lightweight locking mode“(LM_LIGHTWEIGHT) is changed

- Legacy locking mode (LM_LEGACY) will continue to pin carriers if selected

- LM_LIGHTWEIGHT
  - No pointer to Java stack in object’s header
  - Freeze copies LockStack to heap, thaw copies it back
  - Slow case: Make owner be the Thread.tid, no extra overhead on freeze/thaw



- JDK-8337395: Adapt Object Monitors for Virtual Threads:

  - https://bugs.openjdk.org/browse/JDK-8337395

- Work in progress on RISC-V: https://github.com/RealFYang/loom/tree/monitors-riscv-port

Virtual Threads

Unmount at contended monitorenter (WIP):

- Contended monitorenter calls into runtime

- Preempt at monitorenter instead of blocking on carrier
  - Copy Java frames to the heap, same as normal freeze
  - Add virtual thread to the monitor's waiter queue
  - Return from monitorenter "as if" monitor had been acquired
  - Preempt stub resets stack, equivalent to returning from normal freeze
  - Return back to Java in "BLOCKING" state
  - Unmount, transition to "BLOCKED" state

- JDK-8337395: Adapt Object Monitors for Virtual Threads:

  https://bugs.openjdk.org/browse/JDK-8337395

- WIP for RISC-V: https://github.com/RealFYang/loom/tree/monitors-riscv-port

Vector API (Incubating)

Motivation (Project Panama):

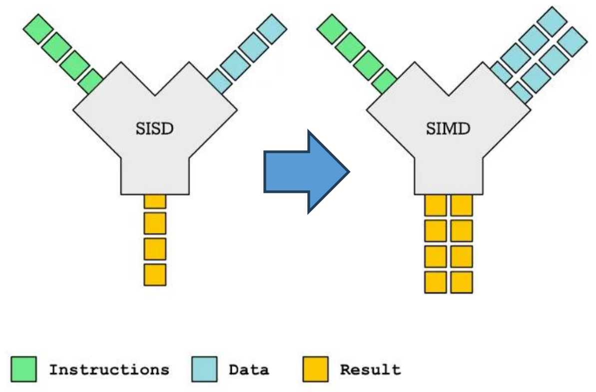
- Auto-vectorization can be fragile and complex to enhance compilers (especially in HotSpot C2 JIT compiler)
  - Generally, transforming scalar code into parallel code is hard

- Vector API enables explicit cross platform data parallel programming
  - For cases where the runtime compiler's auto-vectorizer cannot reliably identify data parallelism from sequential code

- Vector API expressions are reliably compiled to CPU vector instructions, when available
  - Otherwise, falls back to scalar code

Reference: https://openjdk.org/jeps/338

Vector API (Incubating)



Single Instruction Single Data (SISD) vs Single Instruction Multiple Data (SIMD)

```java
public static int[] simpleSum(int[] a, int[] b) {
    var c = new int[a.length];

    for (var i = 0; i < a.length; i++) {
        c[i] = a[i] + b[i];
    }

    return c;
}
```

```java
private static final VectorSpecies<Integer> SPECIES =
IntVector.SPECIES_PREFERRED;

public static int[] vectorSum(int[] a, int[] b) {
    var c = new int[a.length];
    var upperBound = SPECIES.loopBound(a.length);

    var i = 0;
    for (; i < upperBound; i += SPECIES.length()) {
        var va = IntVector.fromArray(SPECIES, a, i);
        var vb = IntVector.fromArray(SPECIES, b, i);
        var vc = va.add(vb);
        vc.intoArray(c, i);
    }

    // Compute elements not fitting in the vector alignment.
    for (; i < a.length; i++) {
        c[i] = a[i] + b[i];
    }

    return c;

}
```

Reference: https://ichi.pro/de/apples-m1-secret-coprozessor-111709032268257

Vector API (Incubating)

- Vector API operators:
  - Arithmetic
  - Compress / Expand
  - Reduction
  - Shift
  - Load / Store
  - Rearrange / Shuffle
  - Cast
  - Mask

```
// vector add

instruct vadd(vReg dst, vReg src1, vReg src2) %{
  match(Set dst (AddVB src1 src2));
  match(Set dst (AddVS src1 src2));
  match(Set dst (AddVI src1 src2));
  match(Set dst (AddVL src1 src2));
  ins_cost(VEC_COST);
  format %{ "vadd $dst, $src1, $src2" %}
  ins_encode %{
    BasicType bt = Matcher::vector_element_basic_type(this);
    __ vsetvli_helper(bt, Matcher::vector_length(this));
    __ vadd_vv(as_VectorRegister($dst$$reg),
               as_VectorRegister($src1$$reg),
               as_VectorRegister($src2$$reg));
  %}
  ins_pipe(pipe_slow);
%}
```

```
static const char *vector_list[] = {
  "AddVB","AddVS","AddVI","AddVL","AddVF","AddVD",
  "SubVB","SubVS","SubVI","SubVL","SubVF","SubVD",
  "MulVB","MulVS","MulVI","MulVL","MulVF","MulVD",
  "DivVF","DivVD",
  "AbsVB","AbsVS","AbsVI","AbsVL","AbsVF","AbsVD",
  "NegVF","NegVD","NegVI","NegVL",
  "SqrtVD","SqrtVF",
  "AndV" ,"XorV" ,"OrV",
  "MaxV", "MinV",
  "CompressV", "ExpandV", "CompressM", "CompressBitsV", "ExpandBitsV",
  "AddReductionVI", "AddReductionVL",
  "AddReductionVF", "AddReductionVD",
  "MulReductionVI", "MulReductionVL",
  "MulReductionVF", "MulReductionVD",
  "MaxReductionV", "MinReductionV",
  "AndReductionV", "OrReductionV", "XorReductionV",
  "MulAddVS2VI", "MacroLogicV",
  "LShiftCntV","RShiftCntV",
  "LShiftVB","LShiftVS","LShiftVI","LShiftVL",
  "RShiftVB","RShiftVS","RShiftVI","RShiftVL",
  "URShiftVB","URShiftVS","URShiftVI","URShiftVL",
  "Replicate","ReverseV","ReverseBytesV",
  "RoundDoubleModeV","RotateLeftV" , "RotateRightV", "LoadVector","StoreVector",
  "LoadVectorGather", "StoreVectorScatter", "LoadVectorGatherMasked", "StoreVectorScatterMasked",
  "VectorTest", "VectorLoadMask", "VectorStoreMask", "VectorBlend", "VectorInsert",
  "VectorRearrange","VectorLoadShuffle", "VectorLoadConst",
  "VectorCastB2X", "VectorCastS2X", "VectorCastI2X",
  "VectorCastL2X", "VectorCastF2X", "VectorCastD2X", "VectorCastF2HF", "VectorCastHF2F",
  "VectorUCastB2X", "VectorUCastS2X", "VectorUCastI2X",
  "VectorMaskWrapper","VectorMaskCmp","VectorReinterpret","LoadVectorMasked","StoreVectorMasked",
  "FmaVD","FmaVF","PopCountVI","PopCountVL","PopulateIndex","VectorLongToMask",
  "CountLeadingZerosV", "CountTrailingZerosV", "SignumVF", "SignumVD",
  // Next are vector mask ops.
  "MaskAll", "AndVMask", "OrVMask", "XorVMask", "VectorMaskCast",
  "RoundVF", "RoundVD",
  // Next are not supported currently.
  "PackB","PackS","PackI","PackL","PackF","PackD","Pack2L","Pack2D",
  "ExtractB","ExtractUB","ExtractC","ExtractS","ExtractI","ExtractL","ExtractF","ExtractD"
};
```

Source code: https://github.com/openjdk/jdk/blob/master/src/hotspot/cpu/riscv/riscv_v.ad

Foreign Function & Memory API

Java and native libraries (Project Panama):

- Calling native functions is possible, with the JNI:
  - Hard to use, brittle combination of Java and C
  - Expensive to maintain, error-prone, poor error checking
  - JNI errors can crash the HotSpot JVM

- Allocating off-heap memory is supported by the ByteBuffer API:
  - 2G addressing limit
  - No way to free/unmap a direct buffer

- Common problems:
  - How to automate the generation of JNI stubs?
  - How to pass data from Java code to native and back?

Reference: https://openjdk.org/jeps/454

Foreign Function & Memory API

- An example:
  - Java code that obtains a method handle for a C library function radixsort and then uses it to sort four strings which start life in a Java array (a few details are elided)
  - Far clearer than any solution that uses JNI, since the implicit conversions and memory accesses that would have been hidden behind native method calls are now expressed directly in Java code.

```
// 1. Find foreign function on the C library path
Linker linker          = Linker.nativeLinker();
SymbolLookup stdlib    = linker.defaultLookup();
MethodHandle radixsort = linker.downcallHandle(stdlib.find("radixsort"), ...);
// 2. Allocate on-heap memory to store four strings
String[] javaStrings = { "mouse", "cat", "dog", "car" };
// 3. Use try-with-resources to manage the lifetime of off-heap memory
try (Arena offHeap = Arena.ofConfined()) {
    // 4. Allocate a region of off-heap memory to store four pointers
    MemorySegment pointers
        = offHeap.allocate(ValueLayout.ADDRESS, javaStrings.length);
    // 5. Copy the strings from on-heap to off-heap
    for (int i = 0; i < javaStrings.length; i++) {
        MemorySegment cString = offHeap.allocateFrom(javaStrings[i]);
        pointers.setAtIndex(ValueLayout.ADDRESS, i, cString);
    }
    // 6. Sort the off-heap data by calling the foreign function
    radixsort.invoke(pointers, javaStrings.length, MemorySegment.NULL, '\0');
    // 7. Copy the (reordered) strings from off-heap to on-heap
    for (int i = 0; i < javaStrings.length; i++) {
        MemorySegment cString = pointers.getAtIndex(ValueLayout.ADDRESS, i);
        javaStrings[i] = cString.reinterpret(...).getString(0);
    }
} // 8. All off-heap memory is deallocated here
assert Arrays.equals(javaStrings,
                     new String[] {"car", "cat", "dog", "mouse"});  // true
```

- JDK-8293841: RISC-V: Implementation of Foreign Function & Memory API (Preview)

  https://github.com/openjdk/jdk/pull/11004

Generational ZGC & ShenandoahGC

JEP 439: Generational ZGC

Owner	Stefan Karlsson
Type	Feature
Scope	Implementation
Status	Closed / Delivered
Release	21
Component	hotspot / gc
Discussion	hotspot dash gc dash dev at openjdk dot org
Effort	XL
Duration	XL
Relates to	JEP 377: ZGC: A Scalable Low-Latency Garbage Collector (Production)
Reviewed by	Erik Helin, Erik Österlund, Vladimir Kozlov
Endorsed by	Vladimir Kozlov
Created	2021/08/25 12:01
Updated	2023/10/26 17:51
Issue	8272979

Summary

Improve application performance by extending the Z Garbage Collector (ZGC) to maintain separate generations for young and old objects. This will allow ZGC to collect young objects — which tend to die young — more frequently.

Goals

Applications running with Generational ZGC should enjoy

- Lower risks of allocations stalls,
- Lower required heap memory overhead, and
- Lower garbage collection CPU overhead.

These benefits should come without significant throughput reduction compared to non-generational ZGC. The essential properties of non-generational ZGC should be preserved:

- Pause times should not exceed 1 millisecond,
- Heap sizes from a few hundred megabytes up to many terabytes should be supported, and
- Minimal manual configuration should be needed.

As examples of the last point, there should be no need to manually configure

- The size of the generations,
- The number of threads used by the garbage collector, or
- For how long objects should reside in the young generation.

Finally, Generational ZGC should be a better solution for most use cases than non-generational ZGC. We should eventually be able to replace the latter with the former in order to reduce long-term maintenance costs.

JEP 404: Generational Shenandoah (Experimental)

Authors	Bernd Mathiske, Kelvin Nilsen, William Kemper, and Ramki Ramakrishna
Owner	William Kemper
Type	Feature
Scope	Implementation
Status	Candidate
Release	24
Component	hotspot / gc
Discussion	hotspot dash gc dash dev at openjdk dot java dot net
Effort	L
Duration	L
Reviewed by	Aleksey Shipilev, Roman Kennke
Endorsed by	Vladimir Kozlov
Created	2021/02/01 22:49
Updated	2024/07/31 17:33
Issue	8260865

Summary

Enhance the Shenandoah garbage collector with experimental generational collection capabilities to improve sustainable throughput, load-spike resilience, and memory utilization.

Goals

The main goal is to provide an experimental generational mode, without breaking non-generational Shenandoah, with the intent to make the generational mode the default in a future release.

Other goals are set relative to non-generational Shenandoah:

- Reduce the sustained memory footprint without sacrificing the low GC pauses.
- Reduce CPU and power usage.
- Decrease the risk of incurring degenerated and full collections during allocation spikes.
- Sustain high throughput.
- Continue to support compressed object pointers.
- Initially support x64 and AArch64, with support for other instruction sets added as this experimental mode progresses to readiness as the default option.

- Reference:
  - https://openjdk.org/jeps/439
  - https://openjdk.org/jeps/404

- JDK-8307058: Implementation of Generational ZGC
  - https://github.com/openjdk/jdk/pull/13771
- JDK-8337511: Implement JEP-404: Generational Shenandoah (Experimental)
  - https://github.com/openjdk/jdk/pull/20395

Various performance tunings

HotSpot C2 patterns and runtime stub performance tunings:

Sub-Tasks

1. RISC-V: C2 VectorizedHashCode
2. RISC-V: C2 CompressBits
3. RISC-V: C2 ExpandBits
4. RISC-V: C2 ReverseI
5. RISC-V: C2 ReverseL
6. RISC-V: C2 CmpU3
7. RISC-V: C2 CmpUL3
8. RISC-V: C2 UDivI
9. RISC-V: C2 UModI
10. RISC-V: C2 UModL
11. RISC-V: C2 ConvHF2F
12. RISC-V: C2 ConvF2HF
13. RISC-V: C2 GetAndAddB
14. RISC-V: C2 GetAndAddS
15. RISC-V: C2 GetAndSetB
16. RISC-V: C2 GetAndSetS
17. RISC-V: C2 OverflowAdd/Sub/Mul I/L
18. RISC-V: C2 Digit
19. RISC-V: C2 LowerCase
20. RISC-V: C2 UpperCase
21. RISC-V: C2 Whitespace
22. RISC-V: C2 CopySignD
23. RISC-V: C2 CopySignF
24. RISC-V: C2 UDivL
25. RISC-V: C2 VectorCastHF2F
26. RISC-V: C2 VectorCastF2HF
27. RISC-V: C2 DivModI
28. RISC-V: C2 DivModL
29. RISC-V: C2 UDivModI
30. RISC-V: C2 UDivModL
31. RISC-V: C2 LoadD_unaligned
32. RISC-V: C2 LoadL_unaligned
33. RISC-V: C2 MulAddS2I
34. RISC-V: C2 NegI
35. RISC-V: C2 NegL
36. RISC-V: C2 OverflowAddI
37. RISC-V: C2 OverflowSubI
38. RISC-V: C2 OverflowMulI
39. RISC-V: C2 OverflowAddL
40. RISC-V: C2 OverflowSubL
41. RISC-V: C2 OverflowMulL
42. RISC-V: C2 PopCountVI
43. RISC-V: C2 PopCountVL
44. RISC-V: C2 ReverseV
45. RISC-V: C2 ReverseBytesV
46. RISC-V: C2 RoundDoubleModeV
47. RISC-V: C2 RotateLeftV
48. RISC-V: C2 RotateRightV
49. RISC-V: C2 SignumVF
50. RISC-V: C2 SignumVD
51. RISC-V: C2 MulReductionVI
52. RISC-V: C2 MulReductionVL
53. RISC-V: C2 MulReductionVF
54. RISC-V: C2 MulReductionVD
55. RISC-V: C2 MulAddVS2VI
56. RISC-V: C2 VectorCmpMasked
57. RISC-V: C2 RoundVF
58. RISC-V: C2 RoundVD
59. RISC-V: C2 ExtractUB
60. RISC-V: C2 VectorLoadShuffle
61. RISC-V: C2 VectorCastF2HF
62. RISC-V: C2 VectorCastHF2F
63. RISC-V: C2 VectorUCastB2X
64. RISC-V: C2 VectorUCastS2X
65. RISC-V: C2 VectorUCastI2X
66. RISC-V: C2 CountTrailingZerosV
67. RISC-V: C2 CountLeadingZerosV
68. RISC-V: C2 CompressBitsV
69. RISC-V: C2 ExpandBitsV
70. RISC-V: minL/maxL

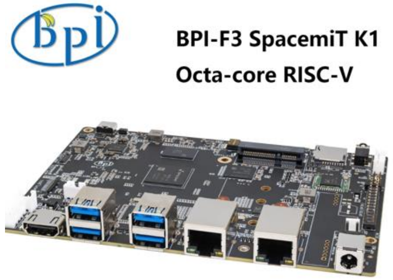
Sub-Tasks

1. RISC-V: C2: Support Zvbb Vector And-Not instruction
2. RISC-V: C2: Support Zvbb Vector Reverse Bits in Elements instruction
3. RISC-V: C2: Support Zvbb Vector Reverse Bits in Bytes instruction
4. RISC-V: C2: Support Zvbb Vector Reverse Bytes instruction
5. RISC-V: C2: Support Zvbb Vector Count Leading Zeros instruction
6. RISC-V: C2: Support Zvbb Vector Count Trailing Zeros instruction
7. RISC-V: C2: Support Zvbb Vector Population Count instruction
8. RISC-V: C2: Support Zvbb Vector Rotate Left instruction
9. RISC-V: C2: Support Zvbb Vector Rotate Right instruction
10. RISC-V: C2: Support Zvbb Vector Widening Shift Left Logical instruction

- JDK-8318216: RISC-V: Missing C2 optional matcher Ops

  https://bugs.openjdk.org/browse/JDK-8318216

Platforms we are testing on



Java workloads we are running

spec

- SPECjbb2005 benchmark
- SPECjbb2015 benchmark
- SPECjvm2008 benchmark

the DaCapo benchmark suite

Renaissance Benchmark Suite

- OpenJDK Code Tools Project:
  - Regression Test Harness (JTreg)
  - Java Concurrency Stress tests (JCstress)
  - Java Microbenchmark Harness (JMH)

Apache NetBeans

Fits the Pieces Together

Development Environment, Tooling Platform and Application Framework.

Minecraft Java server

So where to get a JDK binary?

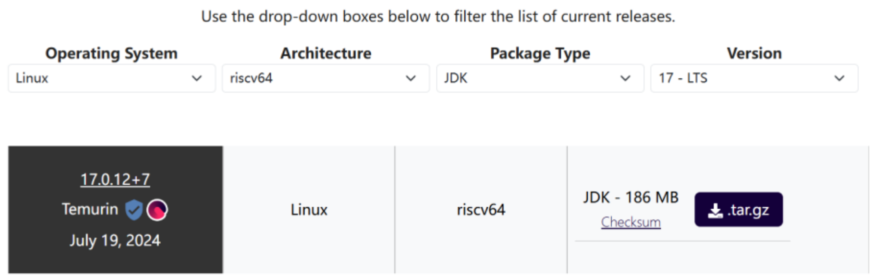
- Available from Eclipse Temurin releases:
  - https://adoptium.net/temurin/releases

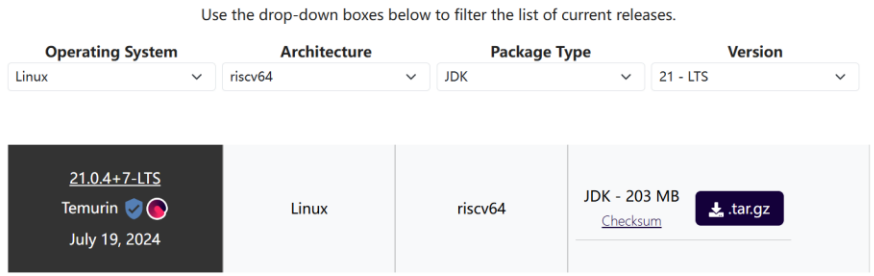
- Available from Bellsoft Liberica releases:
  - https://bell-sw.com/pages/downloads

- OpenJDK RISC-V daily builds available here:
  - https://builds.shipilev.net/openjdk-jdk
  - https://builds.shipilev.net/openjdk-jdk21-dev
  - https://builds.shipilev.net/openjdk-jdk17-dev
  - https://builds.shipilev.net/openjdk-jdk11-riscv

- To build from source code:
  - sh configure --with-debug-level=release --with-jvm-variants=server --with-zlib=system --with-boot-jdk=/home/ubuntu/tools/boot-jdk --with-native-debug-symbols=internal --disable-precompiled-headers --with-jtreg=/home/ubuntu/tools/jtreg --with-gtest=/home/ubuntu/tools/googletest

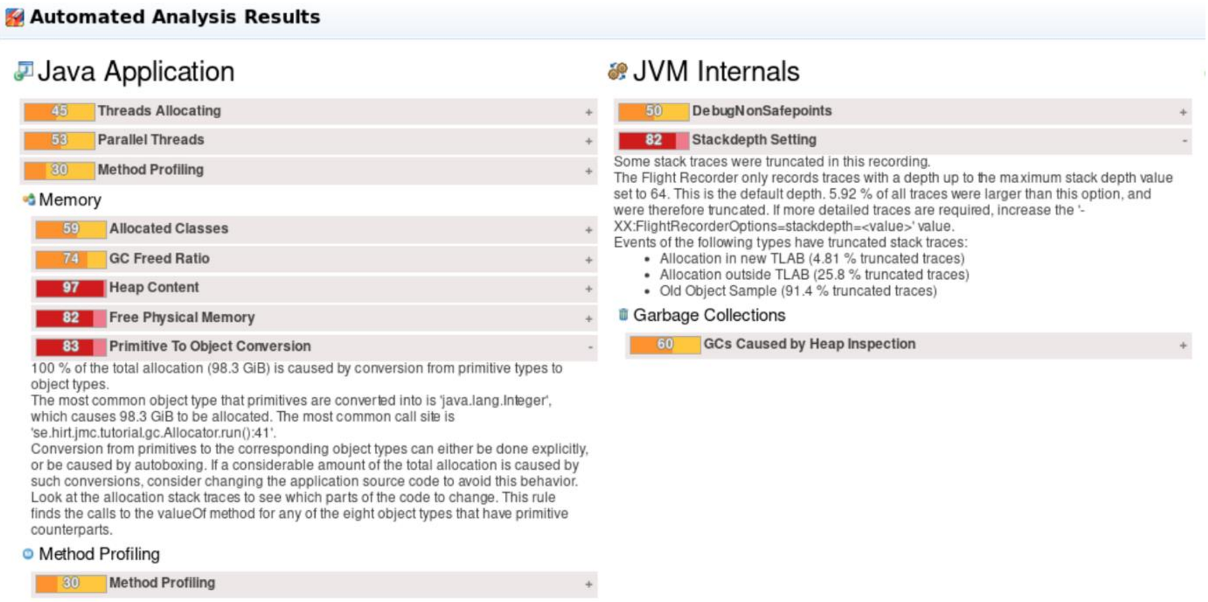
  - make images JOBS=48





Profiling and Diagnostics

Async-profiler: Basic RISC-V support
https://github.com/async-profiler/async-profiler/pull/644



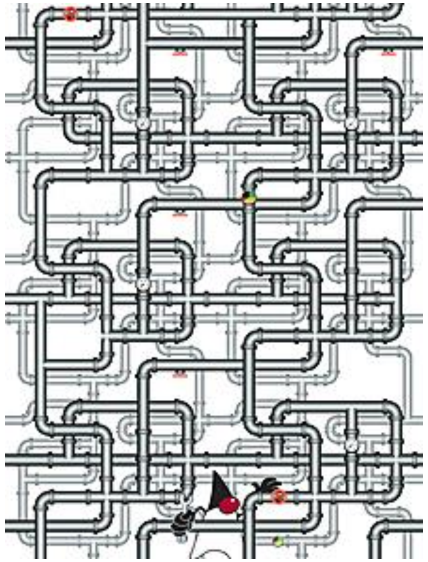
JDK Mission Control (JMC)

Get Involved

- It’s fun to working on Java for brand new architecture!
- Download the source, try a build: https://github.com/openjdk/jdk

- TODO list:

- Support for other OpenJDK projects like Lilliput, Leyden, etc
- Support for other new ratified RISC-V extensions
- Performance tunning: The C2 patterns and runtime stubs could be tuned on hardwares from different vendors



Thank You!