

**中国科学院软件研究所**

**工作周报**

**姓名： 赵士杰**

**时间：2025年08月03日**

目录

[1. 工作任务 3](#_Toc205180798)

[1.1. 学术资料阅读 3](#_Toc205180799)

[1.2. 实践环节 3](#_Toc205180800)

[1.3. 本周出勤情况 3](#_Toc205180801)

[2. 工作进展 3](#_Toc205180802)

[2.1. 学术资料阅读进展 3](#_Toc205180803)

[2.2. 实践环节进展 3](#_Toc205180804)

[2.2.1 有关RVV与C语言版本优化性能异常的分析 3](#_Toc205180805)

[2.2.2寻找新的三方库 7](#_Toc205180806)

[2.2.3 issue 7](#_Toc205180807)

[3. 学术资料阅读笔记 8](#_Toc205180808)

[4. 下周计划 9](#_Toc205180809)

[4.1. 学术资料阅读 9](#_Toc205180810)

[4.2. 实践环节 9](#_Toc205180811)

[5. 参考文献 9](#_Toc205180812)

# 工作任务

## 学术资料阅读

[【JAVA核心】Java GC机制详解-CSDN博客](https://blog.csdn.net/laomo_bible/article/details/83112622)

[JIT（Just-In-Time）编译技术到底是干什么的？使用场景是什么？底层原理是什么？\_jit编译-CSDN博客](https://blog.csdn.net/qq_36777143/article/details/145887340)

[即时编译（JIT） - 知乎](https://zhuanlan.zhihu.com/p/361250220)

[Java中针对Timestamp数据类型操作\_java判断值是timestamp类型-CSDN博客](https://blog.csdn.net/psp0001060/article/details/78283787)

## 实践环节

* 探究有关RVV优化时出现性能差异的原因
* 寻找新的三方库
* 追踪有关SptialFlink的相关issue

## 本周出勤情况

出勤4天：7.29，7.30，7.31，8.1

# 工作进展

## 学术资料阅读进展

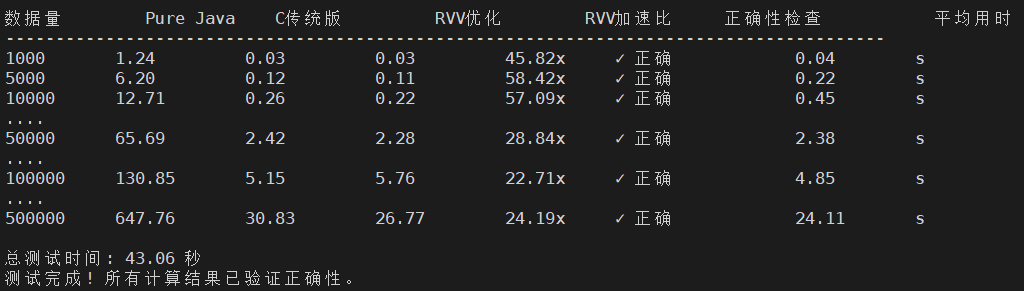
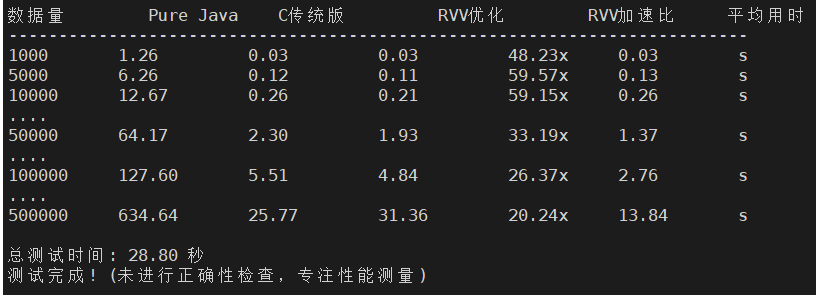
辅助了探查差异原因的代码

## 实践环节进展

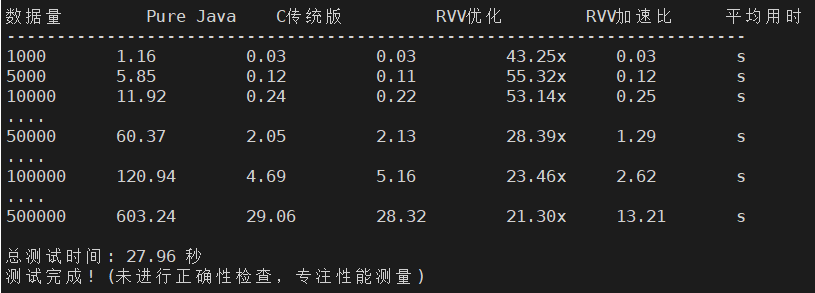
### 2.2.1 有关RVV与C语言版本优化性能异常的分析

1. 因为之前的那一版性能测试的代码写的有些潦草，进行了重构。

2. 重构之后发现无法复现原先代码的问题：重构之后的代码，RVV的速度普遍>C，符合直觉，只有在500k的大数据集的情况下出现了异常但是异常与之前相反：开启校验后，RVV比C快，关闭校验后，RVV比C慢。索性先开始探索这个问题。



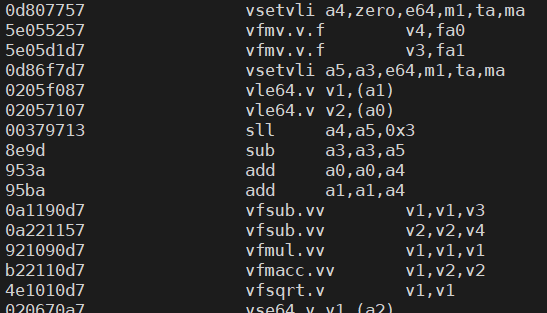
3. 调整了执行顺序：从JAVA、C、RVV，改为了RVV和C的执行顺序随机。在500k数据集的含量下无论是否开启校验，执行RVV速度均对比C有提升。



电脑萤幕画面

AI 生成的内容可能不正确。

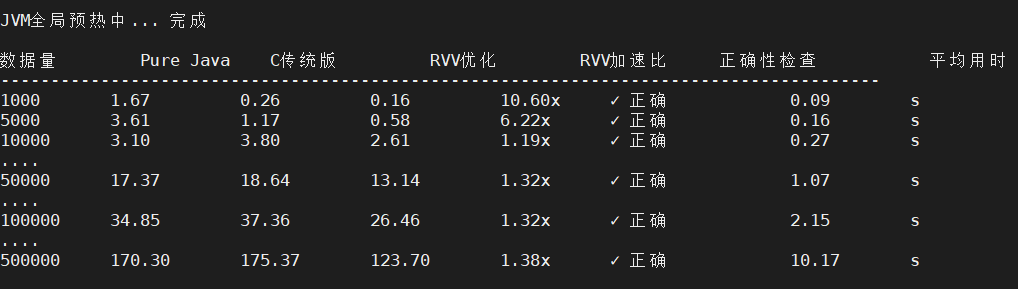
4. 发现随机顺序之后，在50000和100000两个数据量下，RVV速度均比C慢，这一点于之前相符。因此，对此进行探究，使用`objdump -d c/librvvdist.so | grep -A 20 calculate\_distances\_java`看了一眼汇编，发现这是因为编译器已经执行了自动向量化，而且设置的是m1，手工优化时设置的是m8，因此产生了一个有关向量化固定开销和数据量相关收益的曲线



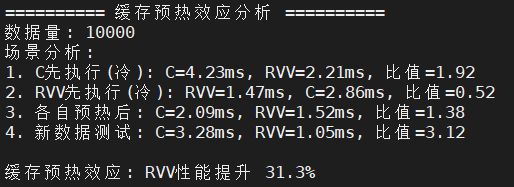
5. 有关最后的是否开启校验的问题现在有以下怀疑：每次校验会进行如下操作：`if (!arraysEqual(referenceResult, cJavaResult, 1e-10)) // 遍历整个数组`，可能会误打误撞产生"内存预热"的效应，

6. 按照思路继续进行相关测试，但是发现性能测试机/home目录挂载的磁盘占用100％，导致无法写入文件，多次尝试无果，遂整体迁移到自己的QEMU里边进行进一步模拟。

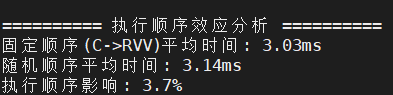
7. 在自己的QEMU中重新配置，但是发现与性能测试主机的性能差异较大，与之相对的时间差别也很大。因此得出的结果可能不是准确的，无法进行准确的性能测试。



8. 对于缓存预热效应的证明如下：说明缓存预热对于RVV的效应提升较大，会造成结果的偏差。



9. 对于是否为顺序执行的探究结果如下：说明是否为顺序执行并非问题根源，而是因为缓存预热



10. 与原先代码进行比对，发现可疑的会造成结果发生异常的点：

```java

System.gc();

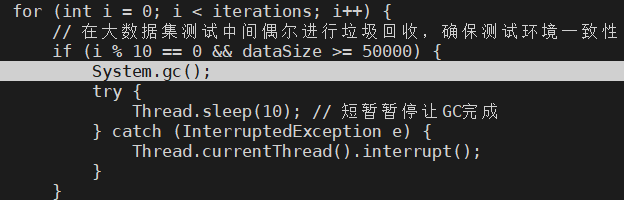
// 可能会造成：

// **内存碎片化**：GC后RVV的向量内存访问变的不连续

// **JNI内存布局破坏**：GC移动对象，影响JNI数组的内存局部

// **缓存污染**：GC过程污染了CPU缓存，RVV失去缓存优势

```



### 2.2.2寻找新的三方库

正在寻找有价值的三方库，目前已经找到一部分：筛选出其中三个：<https://github.com/apache/paimon、https://github.com/apache/arrow、https://github.com/apache/parquet-java>。对于这三个进行判断，寻找可能的更加适合优化的库。paimon的Parquet向量化读取可能是一个好的方向，因此找到了parquet-java这个库。正在进行具体代码的分析，寻找可能的可优化点位。

### 2.2.3 issue

目前正在等待回应。

# 学术资料阅读笔记

**总结下我所遇到的有关RVV与C语言版本的优化的性能差异的问题：**

Q：为什么会呈现出在50000和100000两个数据数量下的RVV反而比C慢的情况？

R：查看其汇编代码

A：原先的C语言代码会被编译器自动向量化——自动向量化时，参数为m1，而手工书写时因为默认数据量大，设为了m8。向量化固定开销和数据量相关的收益互相制衡导致了曲线的出现。可以通过设置适当的大小来调整。

Q（自提，突然发现并产生想法）：为什么这里的JAVA执行速度会这么慢？

R：按说JAVA本身的也有一定的优化，可是在这里依旧还是比较慢。在分析时，遇到报错。后来经过寻找，发现`OpenJDK 64-Bit Zero VM Bisheng (build 25.412-b08, interpreted mode)`。即使用Zero VM（解释器模式），不支持JIT编译。

A：编译器使用的是Zero VM，即：没有JIT编译器 - 所有Java代码都是解释执行；没有即时优化 - 无编译、内联、循环优化等

Q：为什么会测试出RVV普遍比C快的情况？

R：研究过程见上。

A：缓存预热。RVV作为最后执行者，享受到了优势：前面两次计算已经把数据加载到L1/L2/L3缓存，RVV执行时几乎全部命中缓存，延迟大幅降低；CPU的硬件预取器已经"学会"了数据访问模式，RVV执行时预取器提前加载数据，减少cache miss。将二者的执行顺序随机化，发现正常，也出现了性能曲线。

Q：为什么原先一版的代码会出现和现在截然相反的结果？

R：通过分析代码，找可能的原因。

A：手动gc可能导致RVV性能退化。

Q：如何解决缓存预热导致的性能数据差异？来获取准确的性能数据？

R：

A：可以让C和RVV版本进行交替和顺序执行。

Q：如何让RVV的性能更高一些？

R：

A：进行缓存预热。

# 下周计划

## 学术资料阅读

调研其他具有优化潜力的库并着手进行分析和优化。

## 实践环节

对于选定的库判断优化点位并进行优化。

# 参考文献