自适应的申威 SIMD 指令优化内存连续读写方法

毛旻凯[1] 顾雨晨[1] 郜 晨[1] 崔 巍[1]
MAO Minkai GU Yuchen GAO Chen CUI Wei

摘 要 提升高性能处理器性能是现代计算领域的关键任务之一，申威处理器以其显著的运算能力和技术创新受到了研究人员的广泛关注。当前申威处理器在执行大数据量连续内存读写时存在资源浪费的问题。为了解决上述问题，文章提出了一种基于申威架构的 SIMD 指令优化内存连续读写方法。由于 SIMD 指令的实施受到不对界异常等多种技术障碍的限制，进一步提出了一种自适应选择多种普通访存指令和 SIMD 访存指令的方法，以提高内存访问效率。在此基础上设计了应对不同响应条件与操作功能的各读写指令，利用 SIMD 指令访存指令数据较宽的特点，在内核内存读写的 memcpy 函数上设计了 SIMD 指令的实现方法，从而完成整个算法实例。通过 MBW 测试内存读写性能，在内存拷贝和内存块拷贝测试项中，分别获得了 449.4% 和 535.5% 的性能提升。

关键词 优化内存连续读写；自适应方法；SIMD 指令；内存拷贝

doi：10.3969/j.issn.1672-9528.2025.08.035

0 引言

如今，高性能处理器已广泛并深入部署于军事、企业、民用等众多领域，成为工业信息化时代下的技术焦点。申威系列处理器基于自主研发的 SW64 架构设计，拥有完全自主知识产权，推进其处理器、操作系统及软件生态的开发对我国芯片自主进程具有重大意义。

提升高性能处理器的性能是一个复杂的技术任务，需要软硬件的密切协同。在硬件层面，除了制程技术的突破，处理器的架构设计 [1] 至关重要，通过优化流水线、增加并行处理单元、改进缓存层次结构以及提升内存带宽等，可以显著提高处理器的运算速度和效率。在软件方面，编译系统上的优化如并行化和多线程优化 [2] 都可以较好地提升处理器性能，在操作系统中通过减少上下文切换的开销 [3]，优化内存读写等方法也可以充分发挥硬件的潜力，处理器的性能在很大程度上受制于内存子系统的效率，优化内存读写无疑是提升整体性能的关键步骤。

单指令多数据流（single instruction multiple data, SIMD）指令是现代计算机指令集的重要组成部分。相比于单指令单数据流，SIMD 指令支持数据级并行操作，实现单条指令操作多组数据，在数据密集型应用场景下可大幅减少指令数的同时提升程序运行效率。SIMD 指令集可以被利用来加速处理器运行中关键路径的运算比如内存读写，这种软硬件协同的方式，能够最大限度地挖掘处理器的潜力，实现高性能计算任务的高效执行。

当前包括申威在内的主流架构在使用普通访存指令执行内存读写时，面对大数据量的连续内存读写，容易造成处理器资源浪费，软件运行速度降低；如果用 SIMD 指令进行内存读写，处理未对齐的数据也是需要面对的问题。为了解决这一问题，一种基于申威架构的 SIMD 指令优化内存连续读写的方法被提出，其充分利用处理器核心的硬件访存通道带宽，从而提高内存读写速度。

本文的主要贡献包括 3 个方面：（1）提出了一种自适应的申威 SIMD 指令优化内存连续读写方法，根据数据地址对齐情况和数据长度动态选择多种普通访存指令和 SIMD 访存指令的方法，提高内存访问效率；（2）设计了应对不同响应条件与操作功能的各读写指令，利用 SIMD 指令访存指令数据较宽的特点，在内核内存读写的 memcpy 等函数上设计了 SIMD 指令的实现方法，并根据提出的内存读写方法设计了申威 SIMD 内存读写算法的实例；（3）提出的申威 SIMD 指令优化内存连续读写方法，在内存拷贝和内存块拷贝测试项中，分别获得了 449.4% 和 535.5% 的性能提升。

1 相关工作

1.1 申威处理器核心结构

申威处理器由指令部件、整数执行部件、浮点执行部件、数据 Cache 控制部件、二级 Cache 控制部件以及一级指令 Cache、一级数据 Cache 和二级 Cache 组成。采用短向量加速计算技术提高整数和浮点运算性能，支持浮点双 256 位

1. 无锡先进技术研究院 江苏无锡 214122

SIMD 流水线、整数单 256 位 SIMD 流水线，每个时钟周期可产生 11 个整数运算结果或 16 个浮点运算结果，申威处理器的核心结构 [4] 如图 1 所示。

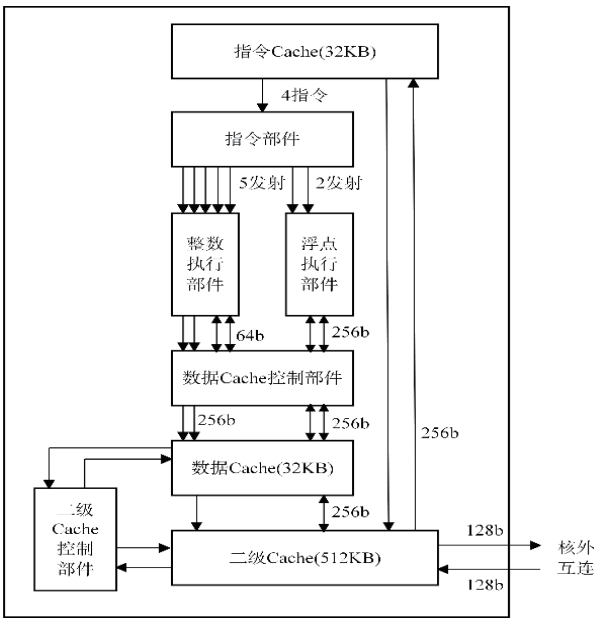

图 1 申威处理器的核心结构

1.2 优化内存读写方法

在内存读写优化领域，研究者们持续深入探索，致力于通过技术创新来提升数据处理的效率和性能，除了在编译器领域致力于提升性能，最新研究主要分布在硬件层面的优化 [4-6]、程序层面的优化 [7-9]，以及操作系统层面上 [10]。

在硬件层面的优化上，通常从缓存优化，内存层次结构、控制器上进行研究。魏森等人 [5] 提出了一种基于 DRAM-NVM 混合内存的 Spark 缓存系统，通过平面混合缓存模型和专用的数据结构设计，提出了基于最小重用代价的缓存管理策略，优先将高重用率的数据保存在 DRAM 中，并考虑了缓存块迁移的成本。吴婧雅等人 [6] 提出了一种基于 FPGA 的 HyperTree 的 B+ 树索引加速系统，设计了高效的流式计算架构和规则化的节点存储格式，以提升内存带宽利用效率，同时通过解耦合的子树结构减少索引维护时的冲突。

在程序层面的优化上，通常从缓存优化、数据结构和算法上进行研究。Chen 等人 [7] 提出了一种基于持久内存的高效键值存储引擎 FlatStore，通过将存储分为持久化日志结构和易失性索引，并引入紧凑日志格式和流水线式水平批处理技术，优化了持久内存的读写性能。Lin 等人 [8] 针对 LSM 树键值存储系统的读放大问题，提出了一种融合 B+ 树、单级热树和分区 Level 0 的 LSM 树的多树结构 MTDB，通过利用工作负载的局部性特征，显著提升了读取性能。Chen 等人 [13] 提出了一种结合缓存与刷新的优化策略，旨在提高基于 LDPC 纠错的闪存存储系统的读取性能，其中利用缓存机制来缩短 LDPC 解码的时间消耗，同时通过定期刷新被缓存替换的高延迟页，将其恢复至低延迟状态。Nie 等人 [9] 提出了一种名为动态请求交错的技术，将具有高局部性的连续逻辑页面分组，并将其编码数据交错存储在不同原始比特误码率的设备页面类型中，同时，还设计了一种有利于页面交错的页面分配方案，利用平面级并行性来加快读取操作并限制了读放大问题。

在操作系统层面上，除了虚拟内存中的大页支持和透明大页，通常从内存分配、页替换算法等内存管理模块的内容进行研究。王紫芮等人 [10] 提出了一种名为 Ultraswap 的页交换机制，以优化传统 Linux 页交换机制在超低延迟 SSD 环境下的性能问题，通过加入轮询请求处理，减少了 I/O 合并与调度的开销，并且对内核中页交换的换入与换出路径进行了改进，降低了关键路径的时间消耗。Li 等人 [12] 提出了一种新的持久化哈希表设计 Rewo-Hash，通过设计了一个无日志的原子写入机制来最小化写入请求的性能开销，并设计一致性的阴影同步方案和非阻塞轻量级哈希表重调整大小方案，最终有效优化了内存读写性能。

1.3 SIMD 指令

SIMD 指令最初被应用于超级计算机中，后在服务器、个人计算机中也得到广泛应用。在关键路径上使用 SIMD 指令可以通过并行处理多个数据元素来显著提升计算速度，研究人员目前在多个场景下运用 SIMD 指令，包括数值计算，多媒体处理，加密与解密，数据压缩与解压缩等方面。如图 2 所示。

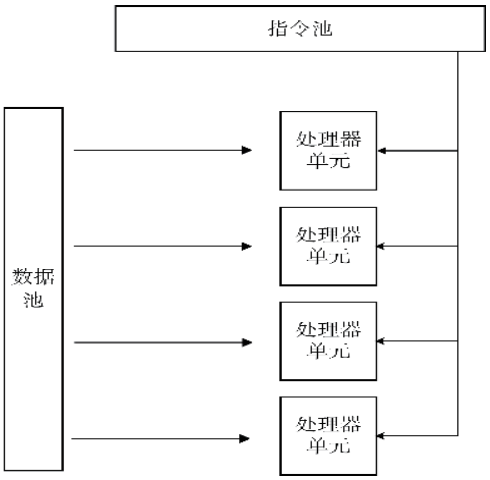

图 2 SIMD 概述图

SIMD 指令在数值计算领域扮演着至关重要的角色，极大地促进了计算效率的提升。Sengupta 等人 [13] 首次提出使用 SIMD 并行化技术加速整数分解算法中的筛选阶段，在数域筛法和多项式二次筛法，成功加速了筛选过程中每个核心的索引计算。沈洁 [14] 等人面向飞腾处理器平台，利用 SIMD 指令设计了一种支持分支处理的向量三角函数优化方法，该方法降低了原始 Sleef 向量三角函数中冗余计算的开销，同时保证了函数计算精度。

SIMD 指令也能为多媒体处理领域带来效率更高的图像、音视频等的处理能力，数据流单指令序列扩展指令集 [15]（streaming simd extensions, SSE）、高级向量扩展指令集 [16]（advanced vector extensions, AVX）均是 Intel 基于 SIMD 设计，大幅提高了多媒体应用的执行速度。Weibbrich 等人 [17] 提出了使用基于垂直和水平 SIMD 架构的方法，来加速嵌入式汽车计算机视觉系统中的尺度不变特征变换算法，提升了图像特征提取性能。Lee 等人 [18] 在医疗成像系统中对多张图像组成的大规模体数据，利用 AVX2 指令集加速三线性插值，并提出了一种改变重复和内存访问顺序的新方法，显著提高了渲染速度。

在加密与解密中，王闯等人 [19] 提出了一种跨平台的通用切片分组密码算法模型，并针对 SIMD 指令集提出了一种细粒度切片并行处理的 SM4 优化算法，显著提升了加密性能。Xu 等人 [20] 设计了一种基于 SIMD 的高吞吐量分组密码实现方法，在分片技术中利用 SIMD 的并行处理能力，优化 ***S*** 盒计算并得出最紧凑的分片表示，从而提高了效率。

SIMD 指令也为数据压缩和解压缩也提供了显著的应用潜力，Lemire 等人 [21] 提出了一种利用 SIMD 的二进制打包技术，实现了键值存储中整数键的高效压缩，特别是在密集键范围内显著提高了单线程查询速度，并提供了优异的压缩比。Hildebrandt 等人 [22] 在 SIMD 寄存器中压缩 k 个不同大小为 N 的块，保证了与标量方法相同的压缩比，降低压缩、解压缩计算开销的同时加速了查询处理。

2 自适应的申威 SIMD 指令优化内存连续读写方法

在探究高性能处理器领域，申威处理器以其显著的运算能力和技术创新而广受认可。经过一代代研究人员在独立自主研发道路上的不懈努力，申威形成了自主的核心处理器指令系统，包括基本指令系统和 SIMD 扩展指令系统。但是在内存读写方面，当前申威系列处理器中普遍使用普通访存指令执行内存读写，访存指令数据宽度不足，尤其是大数据量的连续内存读写时，造成处理器资源浪费，软件运行速度降低的结果。为了解决这个问题，一方面我们提出一种自适应优化的 SIMD 访存策略，通过根据目标地址的对界情况和数据长度动态选择普通访存指令和 SIMD 访存指令的组合的方法；另一方面在使用内核内存读写的相关函数上设计了 SIMD 指令的具体实现方法。

2.1 针对 SIMD 限制因素的自适应优化内存连续读写方法

在 SIMD 并行处理技术中，理论上 32 位 ×8 元素的向量操作能实现 8 倍于标量的性能，64 位 ×4 元素的向量操作能实现 4 倍于标量的性能，SIMD 内存读写性能提升可以近乎通过公式计算：

$$\text{Speedup} \approx \frac{\text{Data\_size}}{\text{Data}_{\text{SIMD\_op}}} \times \frac{\text{Cycles}_{\text{Scalar}}}{\text{Cycles}_{\text{SIMD}}} \quad (1)$$

式中：Speedup 是性能提升程度；Data_size 是总数据处理量；$\text{Data}_{\text{SIMD\_op}}$ 是每个 SIMD 操作处理的数据量；$\text{Cycles}_{\text{Scalar}}$ 是标量操作所需的周期数；$\text{Cycles}_{\text{SIMD}}$ 是 SIMD 操作所需的周期数。

理想情况下，数据的对齐能使 SIMD 运算以最优性能运行，因为所有数据块都能一次性加载到 SIMD 寄存器中进行并行运算。但将程序代码完全转化为 SIMD 向量操作的形式，在实际操作中常受到多种技术障碍的限制。尽管现代处理器的 SIMD 指令已经能处理未对齐的数据，但性能通常会显著下降。以申威处理器为例，SIMD 扩展部件的向量化实现由于向量长度和指令集功能的差异，在向量化过程中常面临错失向量化机会和向量化后性能倒退的问题，比如往往硬件的向量化访存操作必须是地址连续的，且要求 32 字节对界；除上述问题外，一旦发生非对齐的内存访问操作，执行过程中系统将频繁应对不对界异常，这种持续的错误处理机制会大幅减缓程序乃至整个系统的执行速度。为了充分发挥 SIMD 的潜力，需要结合指令集特征，形成简洁高效的向量指令。

针对上述 SIMD 限制条件，本文在针对源地址，目标地址对界情况以及剩余数据长度方面进行了充分考虑，本文提出了一种自适应的申威 SIMD 指令优化内存读写的方法，根据数据地址对齐情况和数据长度动态选择多种普通访存指令和 SIMD 访存指令的方法，以提高内存访问效率，如图 3 所示。

（1）响应于目标地址不满足 32 字节对界且剩余数据长度不小于 32 字节，执行前期普通指令访存方法。

（2）响应于目标地址满足 32 字节对界且剩余数据长度不小于 32 字节，执行 SIMD 指令访存方法。

（3）响应于剩余数据长度小于 32 字节，执行后期普通指令访存方法。

2.2 读写指令响应条件与操作功能设计

在前期普通指令访存方法、SIMD 指令访存方法、后期普通指令访存方法中，存在各指令分别能响应源地址、目标地址、剩余数据长度的不同情况，具体如表 1 所示。

表中涉及到的各指令具体内容如下：

RW1：1 字节读写指令。

RW2：8 字节读写指令。

RW3：先忽略地址低 3 位的 8 字节读，忽略源地址低 3 位，读取连续的 16 字节数据，再根据源地址的低 3 位，从连续 16 字节数据中取出 8 字节数据，写入目标地址，并将源地址与目标地址向后偏移 8 字节，剩余数据长度减 8。

RW4：64 读写指令。

RW5：从源地址读取连续 32 字节数据写入目标地址，并将源地址与目标地址向后偏移 32 字节，剩余数据长度减 32。

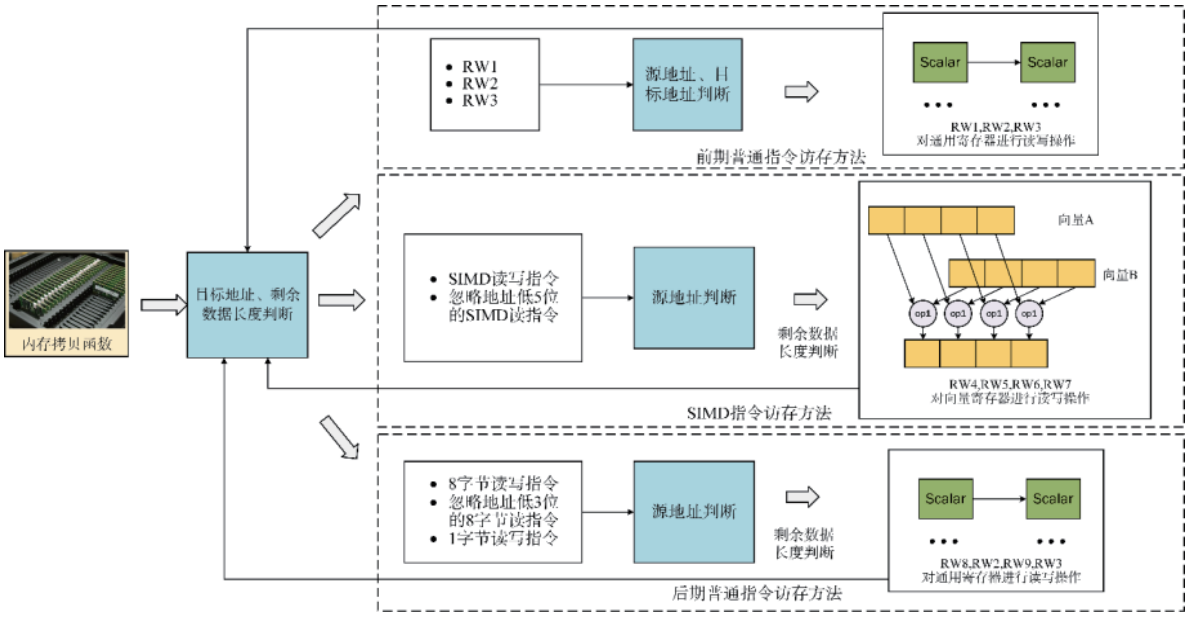

图 3 自适应的申威 SIMD 指令优化内存连续读写方法

表 1 涉及访存方法各指令及能够响应的各个条件

对应访存方法	读写指令类别	指令	目标地址	源地址	剩余数据长度
前期普通指令访存	1 字节读写指令	RW1	不满足 8 字节对界		
	8 字节读写指令	RW2	满足 8 字节对界	满足 8 字节对界	
	忽略地址低 3 位的 8 字节读写指令	RW3	满足 8 字节对界	不满足 8 字节对界	
SIMD 指令访存	SIMD 读写指令	RW4		满足 32 字节对界	不小于 64 字节
		RW5			小于 64 字节
	忽略地址低 5 位的 SIMD 读写指令	RW6		不满足 32 字节对界	不小于 64 字节
		RW7			小于 64 字节
后期普通指令访存	8 字节读写指令	RW8		满足 8 字节对界	不小于 16 字节
		RW2			不小于 8 字节
	忽略地址低 3 位的 8 字节读写指令	RW9		不满足 8 字节对界	不小于 16 字节
		RW3			不小于 8 字节
	1 字节读写指令	RW1			

RW6：忽略源地址低 5 位，读取连续的 96 字节数据，再根据源地址的低 5 位，从连续 96 字节数据中取出连续 64 字节数据，写入目标地址，并将源地址与目标地址向后偏移 64 字节，将剩余数据长度减 64。

RW7：忽略源地址低 5 位，读取连续的 64 字节数据，再根据源地址的低 5 位，从连续 64 字节数据中取出连续 32 字节数据，写入目标地址，并将源地址与目标地址向后偏移 32 字节，剩余数据长度减 32。

RW8：16 字节读写指令。

RW9：忽略源地址低 3 位，读取连续的 24 字节数据，再根据源地址的低 3 位，从连续 24 字节数据中取出连续 16 字节数据，写入目标地址，并将源地址与目标地址向后偏移 16 字节，剩余数据长度减 16。

上述指令规则及其功能如表 2 所示。

表 2 涉及指令规则及功能

指令名称	是否需要地址对界	操作的寄存器	操作的字节数	功能
SIMD 读指令	是	向量	32	读取
SIMD 写指令	是	向量	32	写入
8 字节读指令	是	通用	8	读取
8 字节写指令	是	通用	8	写入
1 字节读指令	否	通用	1	读取
1 字节写指令	否	通用	1	写入
忽略地址低 5 位的 SIMD 读指令	否	向量	32	读取
忽略地址低 3 位的 8 字节读指令	否	通用	8	读取

其中通用寄存器可存储 8 字节数据，向量寄存器可存储 32 字节数据。上述读写指令只能访问虚拟地址，SIMD 读写指令和 8 字节读写指令需要地址对界，其余指令不需要地址对界。忽略地址低 5 位的 SIMD 读指令从访问的虚拟地址中将 32 字节数据读入向量寄存器，忽略地址低 3 位的 8 字节读指令从访问的虚拟地址中将 8 字节数据读入通用寄存器。

2.3 内存拷贝算法设计

针对上述指令及其对应的使用场景，可以对需求场景进行合理的算法设计。以拷贝数据写入目标地址 memcpy 函数为例给出了两种合理的基于上述指令访存规则的内存读写算法：

算法 1 申威 SIMD 内存读写算法 1

输入：源地址 src，目标地址 dest，剩余数据长度 len

输出：拷贝是否成功完成 status

1：while dest 未满足 8 Byte 对界 do

2：　使用 RW1 拷贝 1 Byte 写入

3：　src = src+1 Byte, dest = dest+1 Byte, len = len-1 Byte

4：end while

5：while dest 未满足 32 Byte 对界 do

6：　if src 8 Byte 对界 then

7：　　使用 RW2 拷贝 8 Byte 写入

8：　else if src 非 8 Byte 对界 then

9：　　使用 RW3 读取 16 Byte 拷贝 8 Byte 写入

10：　end if

11：　src = src+8 Byte, dest = dest+8 Byte, len = len-8 Byte

12：end while

13：while len >= 64 Byte do

14：　if src 32 Byte 对界 then

15：　　使用 RW4 拷贝 64 Byte 写入

16：　else if src 非 32 Byte 对界 then

17：　　使用 RW6 读取 96 Byte 拷贝 64 Byte 写入 17：end if

18：　src = src+64 Byte, dest = dest+64 Byte, len = len-64 Byte

19：end while

20：while len >= 16 Byte do

21：　if src 8 Byte 对界 then

22：　　使用 RW2 拷贝 16 Byte 写入

23：　else if src 非 8 Byte 对界 then

24：　　使用 RW9 读取 24 Byte 拷贝 16 Byte 写入

25：　end if

26：　src = src+16 Byte, dest = dest+16 Byte, len = len-16 Byte

27：end while

28：while len >= 8 Byte do

29：　if src 8 Byte 对界 then

30：　　使用 RW2 拷贝 8 Byte 写入

31：　else if src 非 8 Byte 对界 then

32：　　使用 RW3 读取 16 Byte 拷贝 8 Byte 写入

33：　end if

34：　src = src+8 Byte, dest = dest+8 Byte, len = len-8 Byte

35：end while

36：while len >= 1 Byte do

37：　使用 RW1 拷贝 1 Byte 写入

38：　src = src+1 Byte, dest = dest+1 Byte, len = len-1 Byte

39：end while

40：status = true

根据上述算法流程，可以根据相应逻辑编写函数，并将 3 个参数源地址 src，目标地址 dest，剩余数据长度 len 传入子函数中，函数栈中将保存 3 个参数到特定寄存器中，随后使用汇编实现 __memcpy_sisd, __memcpy_simd, __memcpy_simd_align 的拷贝功能，在 __memcpy_sisd 中，利用汇编语句创建循环，按照特定字节拷贝或循环拷贝。

在 SIMD 实现中，在数据元素被打包到宽寄存器后，使用特定架构下单个读写指令 mov 从源地址将数据拷贝到目标地址来处理这些数据，大大提高了内存读写处理的吞吐量和效率。

3 实验与结果

3.1 实验内容

本文对上述算法进行实现并进行了性能测试。MBW 测试工具是 Linux 系统的内存带宽测试工具，可以帮助用户最直观了解系统内存的读写速度，从而为优化系统性能提供参考。本文使用 MBW 测试工具对内存带宽进行测试，流程如图 4 所示。

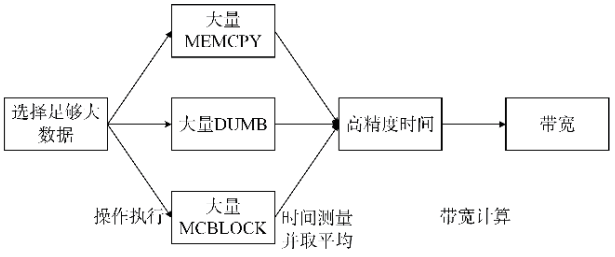

图 4 带宽测试流程

如果测试数据集的大小小于或等于 CPU 缓存的大小，那么数据可能会完全存放在缓存中。这样，测试的实际上是缓存的性能，而不是内存的带宽。选择足够大的数据集可以确保数据不会完全驻留在 CPU 缓存中，从而更准确地测量内存的带宽。另外，小数据块可能会因为较高的缓存命中率而表现出较好的性能，而大数据块则更能测试内存子系统的连续传输能力。

实验前后硬件环境相同，在 CPU 频率、内存频率、内存大小等一致情况下，对比使用申威 SIMD 内存读写方法前后的内存带宽。

3.2 评价指标

MBW 的测试项 MEMCPY、DUMB、MCBLOCK 分别测试了系统对内存空间的内存拷贝、字符串拷贝、内存块拷贝速率，具体过程如下：

（1）MEMCPY：专注于评估系统在执行内存区间之间数据复制任务时的效率，模仿了常见的内存数据迁移操作，在执行 MEMCPY 测试时，MBW 会遍历不同的数据块大小，以测量在不同数据量条件下的内存传输速率。

（2）DUMB：专注于对内存进行无意义的读写操作，在这个测试过程中，数据被反复写入并立即读取，以此来衡量内存的读写性能。

（3）MCBLOCK：专注于评估系统在处理内存块读写操作时的性能，涉及对内存中连续区域的一系列读写动作，通过 MCBLOCK 测试，可以了解系统在处理连续内存区域时的表现，这对于那些依赖高效内存块访问的应用至关重要。测试结果是反映系统对内存空间的内存拷贝、字符串拷贝、内存块拷贝带宽，带宽与性能成正比，性能提升率计算方法为：

$$\Delta_p = \frac{B' - B}{B} \times 100\% \quad (2)$$

式中：Δ_p 为性能提升率；B' 为改进后的带宽；B 为改进前的带宽。

3.3 实验环境

实验选取了申威研发的 6B 3231 处理器，架构为申威架构 sw_64，指令集为申威指令集，核心数为单线程 64 核，具体配置如表 3 所示。

表 3 实验环境配置

项目	配置
处理器	申威 6b 3231
架构	sw_64
核心数	64
L1d cache	2 MB
L1i cache	2 MB
L2 cache	32 MB
L3 cache	128 MB
操作系统	统信 uos
内核	4.19

3.4 实验结果

本文将所提出的申威 SIMD 指令优化内存连续读写的方法，优化内核 memcpy 函数以及其他相似函数，并使用 MBW 测试改进前和改进后的内存拷贝性能，MEMCPY、DUMB、MCBLOCK 测试项的结果如图 5 所示。

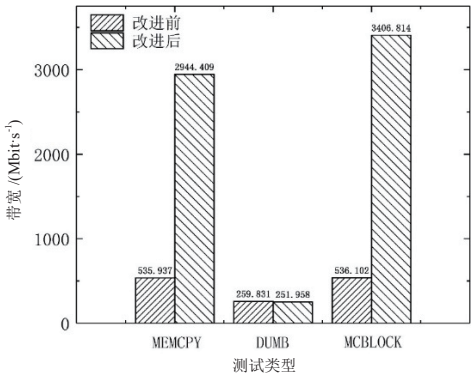

图 5 改进前后内存读写性能结果

从图 5 可以看出，使用所提出的申威 SIMD 指令优化内存连续读写的方法，优化内核 MEMCPY 函数以及其他相似函数，在 MEMCPY 和 MCBLOCK 的性能提升，DUMB 测试项提升效果不明显，性能提升百分比如表 4 所示。

表 4 内存读写性能提升情况

测试项	改进前	改进后	性能提升率 /%
MEMCPY	535.937	**2 944.409**	449.4
DUMB	**259.831**	251.958	-3.0
MCBLOCK	536.102	**3 406.814**	535.5

注：加粗值代表了最佳表现

从表 4 可以看出本文提出的内存读写方法在内存拷贝和内存块拷贝测试项中，分别获得了 449.4% 和 535.5% 的性能提升。MEMCPY 测试通常涉及大量的数据移动，而且这些数据移动往往是连续的。SIMD 指令集可以同时处理多个数据元素，因此在执行连续内存块的数据复制时，SIMD 可以显著提高数据传输的吞吐量。MCBLOCK 测试专注于连续内存块的读写操作，这与 MEMCPY 类似，也是 SIMD 指令集能够有效优化的操作类型，通过 SIMD 指令，可以在单个指令周期内处理更多的数据，从而提升整体的内存带宽和操作效率。

对于 DUMB 测试，其性能差距不大，DUMB 测试主要执行的是无意义的内存读写操作，这种操作通常是随机或非连续的。SIMD 指令集对这种操作的优化效果不如连续内存块操作明显，因为 SIMD 的优势在于处理连续的数据流。

4 结语

本文针对申威系列处理器在内存连续读写时存在的性能瓶颈，提出了一种基于申威架构的 SIMD 指令优化内存连续读写方法。该方法主要包括 3 个方面：（1）自适应选择多种普通访存指令和 SIMD 访存指令，提高内存访问效率；（2）在内核内存读写函数上设计 SIMD 指令实现方法，并给出算法实例；（3）通过实验验证，该方法在内存拷贝和内存块拷贝测试项中取得了显著性能提升。本文为申威处理器内存

读写性能优化提供了有效的技术手段，通过实验可以看出，在 MBW 的内存拷贝和内存块拷贝测试项中，分别获得了 549.4% 和 635.5% 的大幅度性能提升。

未来的工作包括两个方面：（1）进一步研究申威 SIMD 指令在其他领域的应用，如科学计算、图像处理等，以充分发挥其性能优势；（2）针对不同场景和需求，优化 SIMD 指令的调度策略，实现更高效的内存读写操作。

参考文献：

[1] 张峰, 翟季冬, 陈政, 等. 面向异构融合处理器的性能分析、优化及应用综述 [J]. 软件学报 , 2020, 31(8): 2603-2624.

[2] 周雍浩 , 徐金龙 , 李斌 , 等 . 面向神威高性能多核处理器的并行编译优化方法 [J]. 计算机工程 , 2022, 48(9): 130-138.

[3] LONG X, WU J G, WU Y L, et al. Context switch cost aware joint task merging and scheduling for deep learning applications[J]. Parallel computing, 2021, 102: 102753.

[4] 李爽 , 赵荣彩 , 王磊 . 面向申威 1621 通用矩阵乘算法的实现与优化 [J]. 计算机科学 , 2021, 48(S2): 699-704.

[5] 魏森 , 周浩然 , 胡创 , 等 . 基于混合内存的 Apache Spark 缓存系统实现与优化 [J]. 计算机科学 , 2023, 50(6): 10-21.

[6] 吴婧雅 , 卢文岩 , 鄢贵海 , 等 . HyperTree：高并发 B+ 树索引加速器 [J]. 计算机研究与发展 ,2023, 60(7): 1661-1677.

[7] CHEN Y M, LU Y Y, YANG F, et al. FlatStore: an efficient log-structured key-value storage engine for persistent memory[C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. NewYork: ACM, 2020: 1077-1091.

[8] LIN X W, PAN Y B, FENG W J, et al. MTDB: an LSM-tree-based key-value store using a multi-tree structure to improve read performance[J]. The journal of supercomputing, 2024, 80:23995-24025.

[9] NIE S Q, ZHANG Y T, WU W G, et al. DIR: dynamic request interleaving for improving the read performance of aged SSDs[C/OL]//2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA) .Piscataway: IEEE, 2024[2025-02-23].https://ieeexplore.ieee.org/document/8863520.DOI:10.1109/NVMSA.2019.8863520.

[10] 王紫芮 , 蒋德钧 . 基于超低延迟 SSD 的页交换机制关键技术 [J]. 计算机研究与发展 , 2024, 61(3): 557-570.

[11] CHEN J L, LI P X, XIE P. Combining cache and refresh to optimize SSD read performance scheme[J].Advanced parallel processing technologies, 2023:113-129.

[12] LI Z X, HUANG K X. A read-efficient and write-optimized hash table for Intel optane DC persistent memory[J]. Future generation computer systems, 2024, 161: 49-65.

[13] SENGUPTA B, DAS A. Use of SIMD-based data parallelism to speed up sieving in integer-factoring algorithms[J]. Applied mathematics and computation, 2017, 293: 204-217.

[14] 沈洁 , 龙标 , 姜浩 , 等 . 飞腾处理器上向量三角函数的设计实现与优化 [J]. 计算机研究与发展 , 2020, 57(12): 2610-2620.

[15] 王琰 , 向校萱 , 祁燕 . H.264 编码器的 SSE2 指令级优化 [J]. 计算机工程与应用 , 2012, 48(10): 217-221.

[16] 杨昊 , 刘哲 , 黄军浩 , 等 . AKCN-MLWE 算法 AVX2 高效实现 [J]. 计算机学报 , 2021, 44(12): 2560-2572.

[17] WEIBBRICH M, GARCÍA-ORTIZ A, PAYÁ-VAYÁ G. Comparing vertical and horizontal SIMD vector processor architectures for accelerated image feature extraction[J]. Journal of systems architecture, 2019, 100: 101647.

[18] LEE S, KYE H. Efficient MIP volume rendering via fast SIMD interpolation and memory access reordering[J]. Multimedia tools and applications, 2022,82: 10515-10534.

[19] 王闯 , 丁滟 , 黄辰林 , 等 . 面向 SIMD 指令集的 SM4 算法比特切片优化 [J]. 计算机研究与发展 , 2024, 61(8): 2097-2109.

[20] XU R Q, XIANG Z J, LIN D, et al. High-throughput block cipher implementations with SIMD[J]. Journal of information security and applications, 2022, 70: 103333.

[21] LEMIRE D, RUPP C. Upscaledb: efficient integer-key compression in a key-value store using SIMD instructions[J]. Information systems, 2017, 66: 13-23.

[22] HILDEBRANDT J, HABICH D, LEHNER W. BOUNCE: memory-efficient SIMD approach for lightweight integercompression[J]. Distributed and parallel databases, 2023, 41: 439-466.

【作者简介】

毛旻凯（1995—），男，江苏无锡人，本科，工程师，研究方向：操作系统、计算机体系结构，email: derekmmk@hotmail.com。

顾雨晨（1999—），男，江苏无锡人，硕士，助理工程师，研究方向：操作系统、深度学习，email: guyuchen0820@163.com。

郜晨（1997—），男，江苏镇江人，硕士，助理工程师，研究方向：计算机体系结构、操作系统，email: 1079439697@qq.com。

崔巍（1985—），男，陕西商洛人，硕士，副研究员，研究方向：操作系统、系统结构、系统安全，email: cuiwei@wxiat.com。

（收稿日期：2025-04-08 修回日期：2025-08-05）