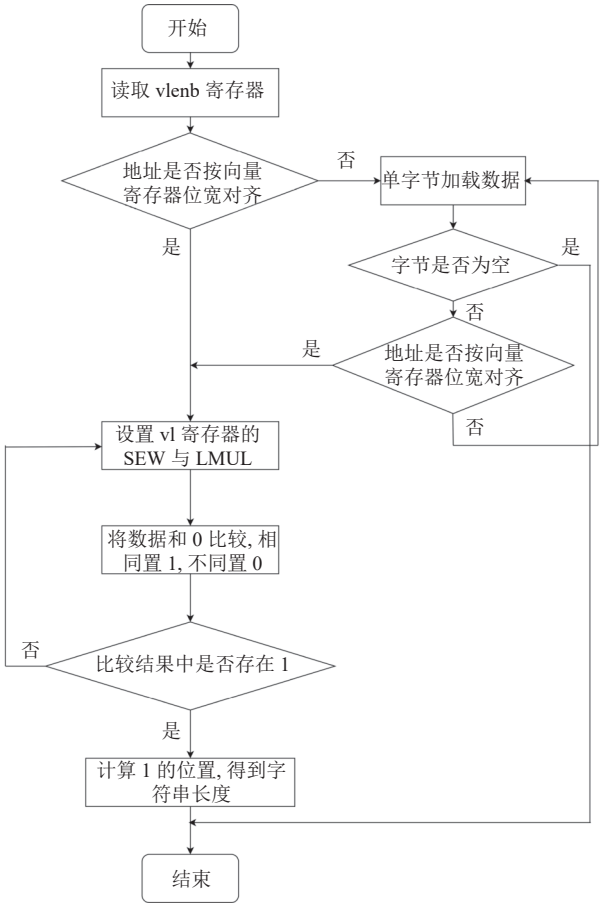

图 4 strlen RVV 指令实现流程图

循环展开: 在 memset 函数中, 循环展开的目的是将字节填充操作展开成多个重复的指令序列, 以提高执行效率. 通过将循环体内的代码重复多次, 减少了循环的迭代次数, 降低了循环控制开销, 同时利用处理器的指令并行和高速缓存, 改善了内存访问模式, 减少对内存的访问延迟. 因此, 本文针对核心循环段展开 32 次, 核心循环代码段如下所示.

代码清单 2. memset 循环展开

```
loop:
    sd a1, 0(t0)
    sd a1, 8(t0)
    sd a1, 2*8(t0)
    sd a1, 3*8(t0)
    … //循环展开
    sd a1, 29*8(t0)
    sd a1, 30*8(t0)
    sd a1, 31*8(t0)
    sd t0, t0, 32*8
    bltu t0, a3, loop
```

地址跳转: 由于循环展开了 32 次, 则要求数据量至少是 256 字节, 因此本文针对数据量小于 256 字节时, 计算其需要展开的次数以及与 loop 循环段的地址偏移, 直接跳转到 loop 内执行, 这样数据量小于 256 字节时仍能够循环展开, 部分代码如下所示.

代码清单 3. memset 地址跳转

```
/* 判断能否做 32 次循环展开 */
andi a4, a4, 31*8
beqz a4, loop
/*计算与 loop 段的偏移*/
neg a4, a4
addi a4, a4, 32*8
sub t0, t0, a4
/* 加载 loop 循环段地址, 并跳转至 loop 内*/
la a5, loop
srli a4, a4, 1
add a5, a5, a4
jr a5
```

尾部处理: 当剩余字节无法循环展开存储, 往往采用的方式是逐字节存储, 本文利用双指针思想, 从剩余字节的头和尾部进行存储, 这种做法虽然会产生重复的存储, 但指令并行与减少跳转次数会带来更大的收益, 部分代码如下所示.

代码清单 4. memset 尾部处理

```
sb a1, 0(t0) //头部存储
sb a1, –1(a3) //尾部存储
li a4, 2 //数据量判断
bgeu a4, a2, 6f
…
sb a1, 5(t0)
sb a1, 6(t0)
sb a1, –6(a3)
sb a1, –7(a3)
li a4, 14
bgeu a4, a2, 6f
sb a1, 7(t0)
```

3.3.2 memset RVV 指令集实现

RVV 指令集特性以及传输指令使得 memset 向量化编程实现更为简洁化. 其中 vmv 指令能够高效地在向量寄存器之间以及向量与标量之间传输和复制数据, 同时配合 vsetvli 指令控制复制的个数, 就不需要像 memset 基础指令集实现那样考虑数据量不同的处理和尾部处理. 核心循环代码如下.

代码清单 5. memset RVV 指令集实现

```
loop:
```

```
vsetvli t1, a2, e8, m8, ta, ma
vmv.v.x v0, a1
sub a2, a2, t1
vse8.v v0, (t0)
add t0, t0, t1
bnez a2, loop
```

4 测试结果及分析

由于目前 RISC-V 硬件支持 RVV 不够完善, gem5 上游中科院软件所 PLCT 实验室提交的 RVV 补丁正在 review, 还未完全合入, 因此本文使用了下游中科院软件所 PLCT 实验室 gem5 的代码仓[12], 并在 gem5 模拟器上测试, 具有一定的参考价值. 测试集采用了 ARM 官方提供的测试用例[13], 由于该测试集中包含了对 ARM 接口函数的调用, 本文将其注释以便在模拟器上运行.

4.1 strlen 函数性能测试

性能测试结果如表 3–表 5 所示, 其中 small aligned 测试了首地址对齐且数据量较小时的性能、small unaligned 测试了首地址不对齐且数据量较小时的性能、medium 测试了数据量中等时的性能, 值越大代表着性能越好.

表 3 small aligned strlen 性能对比

数据量 (B)	musl C语言实现 (B/ns)	基础指令汇编实现 (B/ns)	RVV汇编实现 (B/ns)
1	0.04	0.04	0.03
2	0.08	0.08	0.07
4	0.13	0.14	0.14
8	0.21	0.24	0.27
16	0.38	0.43	0.55
32	0.62	0.73	1.10
64	0.92	1.10	2.20
平均	0.34	0.39	0.62

表 4 small unaligned strlen 性能对比

数据量 (B)	musl C语言实现 (B/ns)	基础指令汇编实现 (B/ns)	RVV汇编实现 (B/ns)
1	0.05	0.05	0.03
2	0.09	0.09	0.07
4	0.14	0.14	0.13
8	0.17	0.17	0.26
16	0.31	0.32	0.52
32	0.53	0.57	1.05
64	0.81	0.91	2.09
平均	0.30	0.32	0.59

从测试结果来看, 基础指令实现的 strlen 与 musl C 语言实现性能相当, 这是因为两者的算法相同导致, 这对于只支持基础指令集的 RISC-V 硬件也可以获得与 C 实现相当的性能; RVV 实现的 strlen 相比于 C 实现, small aligned 测试性能平均提升 83%, small unaligned 测试性能平均提升 98%, medium 测试性能平均提升 703%.

表 5 medium strlen 性能对比

数据量	musl C语言实现 (B/ns)	基础指令汇编实现 (B/ns)	RVV汇编实现 (B/ns)
128 B	1.23	1.51	4.46
256 B	1.45	1.82	9.28
512 B	1.60	2.03	12.46
1 KB	1.68	2.15	15.04
2 KB	1.73	2.22	16.78
4 KB	1.75	2.25	17.80
平均	1.57	2.00	12.64

4.2 memset 函数性能测试

性能测试结果如表 6–表 8 所示, 其中 random 测试了在对应数据量附近时的性能、medium 测试了数据量中等时的性能、large 测试了数据量较大时的性能, 值越大代表着性能越好.

表 6 random memset 性能对比

数据量 (KB)	musl C语言实现 (B/ns)	基础指令汇编实现 (B/ns)	RVV汇编实现 (B/ns)
32	0.67	0.81	1.25
64	0.67	0.80	1.24
128	0.68	0.81	1.25
256	0.67	0.80	1.24
512	0.67	0.80	1.24
1 024	0.67	0.80	1.23
平均	0.67	0.80	1.24

表 7 medium memset 性能对比

数据量 (B)	musl C语言实现 (B/ns)	基础指令汇编实现 (B/ns)	RVV汇编实现 (B/ns)
8	0.39	0.36	0.53
16	0.49	0.54	1.06
32	0.75	1.05	2.12
64	1.32	1.96	3.97
128	2.26	3.05	7.06
256	3.53	6.72	11.57
512	4.89	9.29	17.59
平均	1.95	3.28	6.27

从测试结果来看, 无论是基础指令还是 RVV 实现的 memset, 其性能均好于 musl 库中 C 语言实现的 memset, 一方面基础指令实现的性能提升得益于巧妙地利用循环展开、地址跳转、尾部处理等编程优化,

而这些对于编译器则是无法生成的, 在 random 测试中, 基础指令集实现相比于 C 实现性能提升了 20%, 在 medium 测试中性能提升了 69%, 在 large 测试中性能提升了 88%; 另一方面 RVV 实现的 memset 性能最优, 得益于 RVV 指令集的丰富性, 能够最大粒度地并行处理数据, 在 random 测试中, RVV 实现相比于 C 实现性能提升了 85%, 在 medium 测试中性能提升了 222%, 在 large 测试中性能提升了 334%.

表 8 large memset 性能对比

数据量 (KB)	musl C语言实现 (B/ns)	基础指令汇编实现 (B/ns)	RVV汇编实现 (B/ns)
1	6.02	11.30	22.94
2	6.87	12.91	28.19
4	7.39	13.90	31.84
8	7.68	14.46	34.04
16	7.84	14.75	35.26
32	7.92	14.90	35.90
64	7.96	14.98	36.23
平均	7.38	13.89	32.06

5 结语

针对目前 musl libc 库 RVV 扩展优化还不完善, 提出了基础指令集与 RVV 扩展实现并存的解决方案, 详细介绍了 strlen 与 memset 函数的基础指令集与 RVV 扩展实现的算法流程, 从实验结果看出, RVV 优化的函数具有很大的性能提升. 由于基础 C 库的函数还有很多, 在未来工作中将结合 RVV 扩展优化其他函数, 丰富 RISC-V 基础软件生态.

参考文献

1 冯竞舸, 贺也平, 陶秋铭. 自动向量化: 近期进展与展望. 通信学报, 2022, 43(3): 180–195. [doi: 10.11959/j.issn.1000-436x.2022051]

2 高伟. 面向 SIMD 的自动向量化优化技术研究 [硕士学位论文]. 郑州: 解放军信息工程大学, 2013.

3 Waterman A, Lee Y, Avizienis R, *et al*. The RISC-V instruction set. Proceedings of the 2013 IEEE Hot Chips 25 Symposium. Stanford: IEEE, 2013. 1.

4 musl libc 官网. http://www.musl-libc.org/. (2023-05-01) [2023-05-05].

5 Bakthavatsalam G, Mehata KM. A case for hybrid instruction encoding for reducing code size in embedded system-on-chips based on RISC processor cores. Journal of Computer Science, 2014, 10(3): 411–422. [doi: 10.3844/jcssp.2014.411.422]

6 王诲喆, 唐丹, 余子濠, 等. 开源芯片、RISC-V 与敏捷开发. 大数据, 2019, 7(4): 50–66. [doi: 10.11959/j.issn.2096-0271.2019032]

7 刘畅, 武延军, 吴敬征, 等. RISC-V 指令集架构研究综述. 软件学报, 2021, 32(12): 3992–4024. [doi: 10.13328/j.cnki.jos.006490]

8 Asanović K. Vector microprocessors [Ph.D. Thesis]. Berkeley: University of California, 1998.

9 RISC-V 向量扩展规范. https://github.com/riscv/riscv-v-spec. (2021-09-18)[2023-05-05].

10 叶锡聪, 庄灿锋, 王宇木, 等. RISC-V 向量指令集的 Compute Library 函数库移植. 单片机与嵌入式系统应用, 2021, 21(1): 8–13.

11 李恺, 翁玉萍. 基于龙芯 2F 的 Glibc 库优化. 电子技术, 2010, 37(10): 27–29.

12 gem5 模拟器. https://github.com/plctlab/plct-gem5. (2023-04-25) [2023-05-05].

13 ARM 官方测试集. https://github.com/ARM-software/optimized-routines/tree/master/string/bench. (2022-02-10)[2023-05-05].

(校对责编: 孙君艳)