



南開大學
Nankai University

计算机学院
并行程序设计实验报告

128 位整数计算的 SIMD 优化

姓名：钟坤原

学号：22124686

专业：计算机科学与技术

2024 年 4 月 28 日

目录

1 问题重述	2
1.1 问题描述	2
1.2 问题研究	2
2 实验环境	2
3 实验设计	2
3.1 SIMD 思路	2
3.2 运算实现	3
3.3 代码设计	3
4 实验结果分析	3
4.1 实验环境	3
4.2 实验分析	4
5 实验总结	4
6 代码	4
6.1 伪代码	4
6.2 代码	5
7 图和表格	6
7.1 图	6
8 其他会用到	6
8.1 枚举	6
8.2 公式	7
8.3 链接	7

1 问题重述

1.1 问题描述

传统的大整数运算依赖于基于标量的处理方式,这在处理多个数据点时会引入大量的计算延迟。单指令多数据流 (SIMD) 技术提供了一种有效的解决方案,它允许同时对多个数据点执行相同的操作,从而显著提高了数据处理的吞吐量。通过 SIMD 来加速 128 位整数的运算,是我要实现的任务。

1.2 问题研究

本研究提出了一种利用 SIMD 技术优化 128 位整数向量运算的新算法。通过设计适用于 SIMD 操作的数据结构和算法,如 U128x8,不仅提高了基本算术运算的效率,还简化了诸如加法、减法和模运算等更为复杂的数学操作。此外,算法还包括对整数向量进行有效管理和优化的策略,以最大限度地利用现代 CPU 的并行处理能力。

2 实验环境

操作系统	实验平台	CPU 型号	GPU 型号	编译选项	L1-L2-L3	语言
win10	x86	i7-9750H	RTX1650H	-m64、-std=c17	384KB-1.5MB-12MB	Rust

表 1: 实验环境

3 实验设计

3.1 SIMD 思路

本文选择用两个向量寄存器分别存储一组数的高低位

- lo: 低 64 位
- hi: 高 64 位

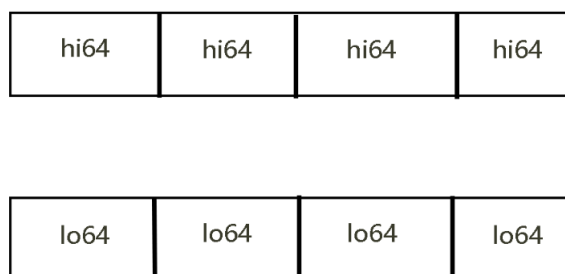


图 3.1: 向量寄存器存储情况

在这种数据排布下,一组向量寄存器内的数据是完全无关的,这样向量和标量的计算过程就完全统一了,在代码中直接把原来的数据类型和运算改成向量版本即可。

3.2 运算实现

- 数据载入 (Splat 操作): splat 函数将单个 u128 整数值拆分为高低两部分, 并将这些值分别载入 hi 和 lo 向量寄存器中。通过 splat 函数, 可以将一个 u64 值复制到一个 SIMD 向量的所有元素中, 实现并行处理的初始化。
- 算术运算:
 - 加法和减法: 通过直接对 hi 和 lo 进行向量加或向量减操作, 可以同时处理向量中的八个 128 位整数。
 - 条件减法 (sub_on_ge): 此操作涉及比较两个向量并根据比较结果选择性地执行减法。通过使用掩码和选择操作, 可以并行地对每个元素执行条件判断和相应的算术操作。
 - 乘法和加法结合 (mul10_add): 此函数实现了向量中每个元素乘以 10 后加上一个常数值。这涉及到对 lo 和 hi 分别进行乘法和加法操作, 并处理可能的进位。这一操作充分利用了 SIMD 的并行能力来加速连续的算术操作。
 - 位运算: 通过重载位与 (&) 和位或 (|) 操作符, U128 支持在 SIMD 寄存器上执行向量位运算, 这对于执行逻辑操作和某些类型的数值处理非常有用。

3.3 代码设计

- 数据预处理: 使用一个结构体将 u128 的前 64 位和后 64 位存储起来。我使用 rust 的 splat 方法将这个 128 的每一位位置零, 再使用 splat 接收一个 128 位的数字并拆出它的高位和低位分别存储到 hi 和 lo。
- 减法: 把每一位相加/相减, 然后处理每一位的借位情况
- 除法: 假设除以 m, 可以把每个位置大于 4*m 的减去 4*m, 如果大于 2*m 再减去 2*m, 最后如果大于 m 再减去 m, 最终可以得到除以 m 的数。
- 位运算: 位运算可以直接使用 & 和 | 来进行每位的快速运算, 重载即可。

4 实验结果分析

4.1 实验环境

由于鲲鹏服务器没有 avx2 和 avx512 扩展指令, 我使用 x86 架构的 ubuntu 系统来运行。

```
[s2212468@master ~]$ cat /proc/cpuinfo
processor       : 0
BogoMIPS      : 200.00
Features       : fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp asimdhp c
puid asimdrdm jscvt fcma dcpop asimddp asimdghm
CPU implementer : 0x48
CPU architecture: 8
CPU variant    : 0x1
CPU part       : 0xd01
CPU revision   : 0
```

图 4.2: 鲲鹏服务器指令集

4.2 实验分析

我使用 benchmark 来测试我的程序运行时间，分别在开销标量，avx2 和 avx512 扩展指令集上测试了乘加运算、取模运算、N=256 计算一轮整体的时间。

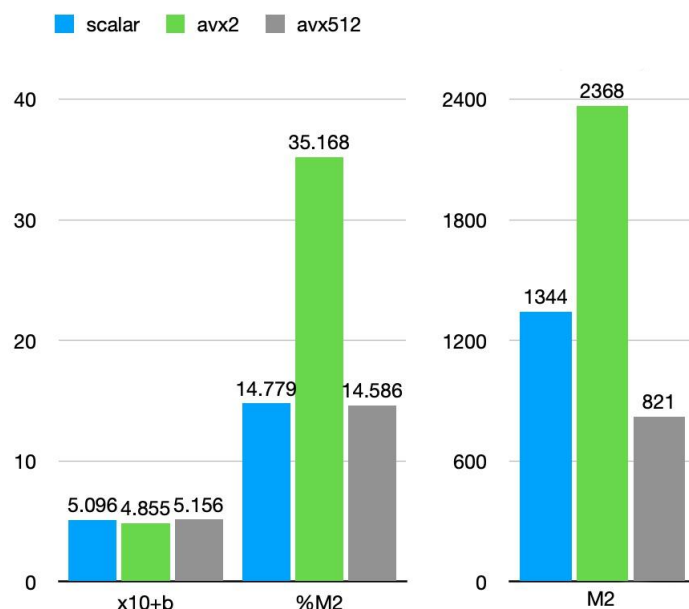


图 4.3: 性能测试时间效果

可以看到在一定时间规模下，avx512 指令集相对于标量运算缩短了 30%，具有相对显著的效果。但是 avx2 的效果不如标量，查阅资料分析可知，一方面因为标量版本已经非常优化了，一次 u128 的取模只需要 4 次 u64 比较和 1 次 u128 减法；另一方面向量版本由于无法利用分支跳转，所以只能靠大量的计算，一次 u128 取模需要 12 次 u64 比较和 4 次 u128 减法，还有不少 select 操作。整体来说 SIMD 算力的优势难以抵消计算量的劣势，所以性能还不如标量。

5 实验总结

本实验验证了 SIMD 技术在处理 128 位整数运算上的效率提升。基于 u128 类型，主要关注加法、减法、乘法以及模运算的性能表现。通过将常规的标量运算与 SIMD 运算进行比较，旨在展示 SIMD 在并行处理大规模数据集时的性能优势。Rust SIMD 模块能够生成接近最优的本机向量指令。AVX512 相比 AVX2 有不小的改进。SIMD 优化为 u128 类型的运算带来了显著的性能提升，特别是在处理大量数据时的效率极为突出。这证明了在需要大规模数值计算的应用中，采用 SIMD 技术可以显著提高应用程序的性能。未来工作可以探索更多的 SIMD 优化技术，以进一步提升处理速度和效率。本次实验的相关代码和文档已经上传 github: [Github](#)

6 代码

6.1 伪代码

伪代码

Algorithm 1 初始化 obj 文件信息——对应 MeshSimplify 类中 readfile 函数, Face 类 calMatrix 函数

Input: obj 文件, 顶点、边、面列表

Output: 是否读取成功

```

1: function CALMATRIX(Face)
2:    $normal \leftarrow e1 \times e2$ 
3:    $normal \leftarrow normal / normal.length$ 
4:    $temp[] \leftarrow normal.x, normal.y, normal.z, normal \cdot Face.v1$ 
5:    $Matrix[i][j] = temp[i] * temp[j]$ 
6:   return Matrix
7: end function
8: 根据 obj 的 v 和 f 区分点面信息, 读取并加入列表
9:  $scale \leftarrow$  记录点坐标中距离原点最远的分量, 以便后续 OpenGL 进行显示
10:  $ori \leftarrow$  记录中心点, 便于 OpenGL 显示在中心位置, 避免有的 obj 偏移原点较多
11: 根据三角面片信息, 计算一个面的三条边
12: 计算每个面的矩阵  $\leftarrow calMatrix$ 
13: 将每个面的矩阵加到各点, 由点维护
14: return True

```

6.2 代码

代码样式 1

```

1
2  int main() {
3      cout << "Hello World!" << endl;
4      return 0;
5  }

```

代码样式 2

逐列访问平凡算法

```

1  void ord()
2  {
3      double head, tail, freq, head1, tail1, times=0; // timers
4      init(N);
5      QueryPerformanceFrequency((LARGE_INTEGER *)&freq);
6      QueryPerformanceCounter((LARGE_INTEGER *)&head);
7      for (int i=0; i<NN; i++)
8          for (int j=0; j<NN; j++)
9              col_sum[i] += (b[j][i]*a[j]);
10     QueryPerformanceCounter((LARGE_INTEGER *)&tail);
11     cout << "\nordCol:" << (tail-head)*1000.0 / freq << "ms" << endl;
12 }

```

7 图和表格

7.1 图

如图7.4所示



图 7.4: Caption

如图7.5所示。如图5(a)所示。

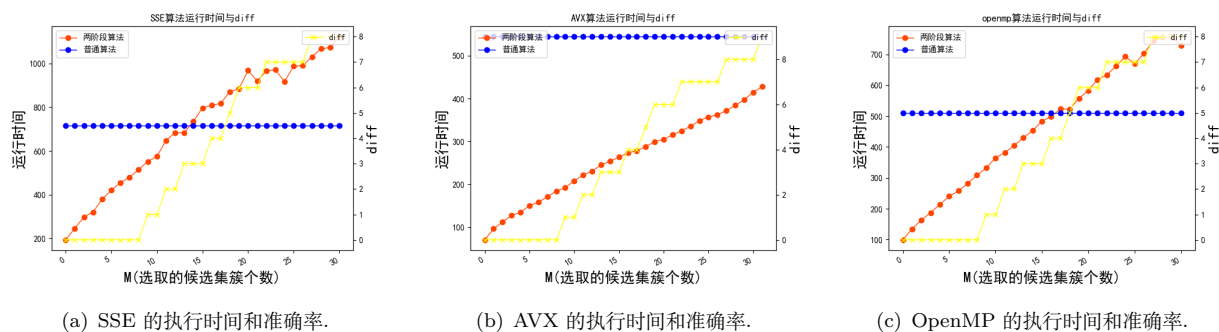


图 7.5: 不同并行优化算法的执行时间与准确率对比

表, 如表2所示。

N/n\Algo	naive-conv	naive-pool	omp-conv	omp-pool
64/2	0.0167	0.01255	0.04142	0.03799
64/4	0.03599	0.0394	0.0458	0.0421

表 2: 性能测试结果 (4 线程)(单位:ms)

带单元格表格, 如表3所示。

Cost		To				
		A	B	C	D	E
From	B	7	0	1	3	8
	C	8	1	0	2	7
	D	8	3	2	0	5

表 3: 结点 C 距离向量表 (无毒性逆转)

推荐你们使用这个网站, 将数据填进去, 设置好格式之后就可以自动生成表格的 latex 代码。

[TableGenerator](#)

8 其他会用到

8.1 枚举

带标号枚举

1. 1

2. 2

不带标号枚举

- 1

- 2

8.2 公式

多行公式

$$a + b = a + b \tag{1}$$

$$\frac{a + b}{a - b} \tag{2}$$

$$a + b = a + b \frac{a + b}{a - b} \tag{3}$$

单行公式：

$$\sum_{i=1}^N$$

行内公式： $\sum_{i=1}^N$

推荐你们使用 *Mathpix Snip*。是一个公式识别软件，可以自动生成公式对应的 latex 代码。

8.3 链接

参考文献 [3][1][2] 超链接 [YouTube](#)

参考文献

- [1] Gene H Golub and James M Ortega. *Scientific computing: an introduction with parallel computing*. Elsevier, 2014.
- [2] Stephen Bassi Joseph, Emmanuel Gbenga Dada, Sanjay Misra, and Samuel Ajoka. Parallel faces recognition attendance system with anti-spoofing using convolutional neural network. In *Illumination of Artificial Intelligence in Cybersecurity and Forensics*, pages 123–137. Springer, 2022.
- [3] Michael J Quinn. *Parallel computing theory and practice*. McGraw-Hill, Inc., 1994.