Highway库介绍

——一个提供高性能便捷式SIMD/vector服务的开源C++库

PLCT实验室：史玉龙

Email：yulong@nj.iscas.ac.cn

Highway是什么
为什么要做highway库
怎么用highway库
当前实现进展

Highway是什么？

- Highway是一个通过提供可移植内置函数的形式来满足SIMD/vector服务的C++库。其函数能够直接映射到CPU的SIMD指令，无需进行大量的编译转换。
- SIMD即Single Instruction, Multiple Data，一条指令操作多条数据，能够高效完成并行计算。
- Highway仓库地址： https://github.com/google/highway
- Highway开发人员：Google公司janwas，邮箱： janwas@google.com

为什么要做Highway库？

- Google公司的开发人员认为像服务器、移动终端、台式机的CPU还具有很大的，未被开发的前置，CPU的计算效率还能够进一步提升。
- Highway能够使得SIMD/vector编程切实可行。
- 适用于多种目标机器，目前支持四种架构；相同的应用程序可以应用于8个指令集。
- 部署灵活，应用程序可以生成多个目标代码，运行时选择一个最优的；也可以选择单个指令集生成目标代码，减少一定的运行开销。
- 广泛用于各种领域，如图像、视频、音频、线性代数、排序和随机数生成

怎么用highway库?

- 在线工具： https://github.com/google/highway的readme文件中，有一个在线编辑代码且可编译的工具

Online demos using Compiler Explorer:

- generating code for multiple targets (recommended)
- single target using -m flags

怎么用highway库?

- 本地安装

1.该项目是使用Cmake生成和构建，在基于Debian系统执行以下命令完成安装：

sudo apt install cmake

2.highway的单元测试使用googletest. Cmake在配置时会下载依赖项，可以设置Cmake变量为NO且安装gtest来禁用该红能。

sudo apt install libgtest-dev

3.将highway构建为静态库或共享库

mkdir -p build && cd build

cmake ..

make -j && make test

怎么用highway库?

- 创建向量时代码移植

```
// Defined by Highway:
template <typename Lane, size_t kLanes>
struct Simd {  // Empty tag type
  using T = Lane;
};
Type128 Zero(Simd<float, 4> /*tag*/);
Type256 Zero(Simd<float, 8> /*tag*/);


// Your code
const HWY_FULL(float) d;  // = Simd<float, ??>
const auto zero = Zero(d);
const auto one = Set(d, 1.0f);
```

怎么用highway库?

- 涉及loops和memory的代码移植

```
for (size_t x = 0; x< xsize; ++x)
{
    const float xval = rowx[x];
    const float yval = rowy[x];
    const float scaler = s + (yw*(1.0f - s)) /
(yw + yval*yval);
    rownew[x] = scaler * xval;
}
```

```
for (size_t x = 0; x< xsize; x += Lanes(d))
{
    const auto xval = Load(d, rowx + x);
    const auto yval = Load(d, rowy + x);
    const auto scaler = s + (yw*(one - s)) /
MulAdd(yval, yval, yw);
    Store(scaler * xval, d, rownew + x);
}
```

怎么用highway库？

- 涉及Alignment的代码移植

```
std::vector<float> rowx(128);
for (size_t x = 0; x < xsize; x += Lanes(d))
{
    const auto xval = Load(d, rowx.data() + x);
    …
}
//这种要求内存对齐
    const auto xval = LoadU(d, rowx.data() + x);
//这种不要求内存对齐，但是效率很低
```

```
HWY_ALIGN float rowx[128];
//对于成员可变和大向量来说是不安全的


hwy::AlignedFreeUniquePtr<float[]> rowx =
hwy::AllocateAligned<float>(128);
//可用于成员可变和大向量的情况
```

怎么用highway库?

- 涉及data layout的代码移植

```
struct Point {
  float x;
  float y;
};
hwy::AlignedFreeUniquePtr<Point[]> points =
hwy::AllocateAligned<Point>(N);

const HWY_FULL(float) d;

// mixes x and y in vector
auto mixed = Load(d, &points.data().x);

hwy::AlignedFreeUniquePtr<float[]> all_x_then_y =
hwy::AllocateAligned<float>(N * 2);
auto only_x = Load(d, all_x_then_y.data());
auto only_y = Load(d, all_x_then_y.data() + N);
```

怎么用highway库?

- 涉及branches的代码移植

```
float RemoveRangeAroundZero(float w, float x) {
  return           x > w ? x - w :
                        x < -w ?    x + w : 0.0f;
}
template<class V>
V RemoveRangeAroundZero(V w, V x) {
  return IfThenElse(x > w, x - w,
         IfThenElseZero(x < Neg(w), x + w));
}

bool AllPositiveIntegers(int v) {
  return v >= 0;
}
template<class V>
bool AllPositiveIntegers(V v) {
  // avoids/hides 'zero'/'sign bit' constant
  return AllTrue(Abs(v) == v);
}
```

应用举例——RNG(random number generator)

```
for (size_t i = 0; i < N; ++i) {
  auto s1 = s0_[i];
  const auto s0 = s1_[i];
  const auto bits = s1 + s0;
  s0_[i] = s0;
  s1 ^= s1 << 23;
  random_bits[i] = bits;
  s1 ^= s0 ^ (s1 >> 18) ^
        (s0 >> 5);
  s1_[i] = s1;
}
```

```
const HWY_FULL(uint64_t) d;  // assumes <= 512bit
for (size_t i = 0; i < N; i += Lanes(d)) {
  auto s1 = Load(d, s0_ + i);
  const auto s0 = Load(d, s1_ + i);
  const auto bits = s1 + s0;
  Store(s0, d, s0_ + i);
  s1 ^= ShiftLeft<23>(s1);
  Store(bits, d, random_bits + i);
  s1 ^= s0 ^ ShiftRight<18>(s1) ^
        ShiftRight<5>(s0);
  Store(s1, d, s1_ + i);
}
```

当前实现进展

- Supported targets：scalar, S-SSE3, SSE4, AVX2, AVX-512, AVX3_DL(~Icelake, requires opt-in by defining), NEON(ARMv7 and v8), SVE, SVE2, WASM SIMD.
- 0.11版本已经可以稳定的适用于一些项目中。
- RVV的支持计划2022年上半年完成。当前实现了一个试验性质的版本，用的是clang/llvm编译器，gcc没有相应的支持。
- Google开发者的issue讨论： https://github.com/google/highway 的issue记录中能够看到关于RVV的实现进展。