全国先进技术创新大赛-A3 技术报告

1. 背景

1.1 SIMD 编程

通常在 CPU 的编程中，程序员处理的是标量数据，这对于通用计算领域而言，往往是足够的。而在一些特定的计算领域，如图像处理、信号处理、科学计算，以及人工智能等，程序员通常需要操作海量的数据，并做相同或者类似的计算。在这些领域，使用循环对不同数据做标量运算往往无法达到性能的要求。因此普遍需要程序员对数据进行并行操作。一种结合指令集设计的方式就是进行 SIMD 编程。SIMD 是 Single Instrustion, Multiple Data 的缩写，通常也称为“单指令多数据”，是一种较为常见的并行计算技术。它能够同时对多个数据元素执行相同的操作，从而提高程序的执行效率。而本次竞赛提供的 GCU 便支持 SIMD 编程，并且提供了多个 SIMD 内置函数接口，通过使用 SIMD 编程能够同时对多个数据进行操作，这显著提高了程序运行速度和计算效率。

1.2 向量化方式

除了手工向量化，即通过内嵌汇编或编译器提供的 intrinsic 函数来添加 SIMD 指令，也可以通过自动向量化，即利用编译器分析串行程序中控制流和数据流的特征，识别程序中可以向量执行的部分，将标量语句自动转换为相应的 SIMD 向量语句。默认情况下，GCC、ICC 或 AOCC/LLVM 编译器中都启用了部分自动向量化功能，可以对一些简单循环体(无较多条件语句、函数调用，无嵌套循环)实现自动向量化操作。

1.3 Intel 高级向量扩展

Intel CPU 支持多种向量扩展指令，位宽从 64bit、128bit、256bit 到 512bit，分别对应于 MMX、SSE、AVX/AVX2 和 AVX512。位宽的增加可以提升向量化指令的计算效率，但同时也对数据排布提出了更高的要求。

2. 问题分析和解决方案

决赛一共有 4 道题，Softmax 算子、Maxpooling2d 算子、Sort 算子和 StringMatch 算子，分别占线上打榜成绩的 20%、20%、20%和 40%。在最终的线上打榜成绩排名第三（郭天宇，86.97 分）。

排名	用户名	分数
1	张力中	95.98 分
2	申邦渝	94.24 分
3	郭天宇	86.97 分
4	王鑫	83.95 分
5	赵文新	81.37 分
6	胡鹏	79.66 分
7	赵全军	65.07 分
8	林玮臻	59.03 分

共 8 条 1 10 条/页

2.1 Softmax 算子（90.95pts）

Softmax 算子是一种常用于机器学习和深度学习中的数学操作，通常用于将一组数值转换成概率分布。它的作用是将输入的一组实数值转换为概率分布，其中每个实数值对应一个概率值，这些概率值之和等于 1。 在深度学习中，Softmax 通常作为神经网络的输出层的一部分，用于多类别分类问题，如图像分类或自然语言处理中的词语分类。它帮助模型将原始输出转化为概率分布，以便进行分类和决策。Softmax 具体的计算公式如下：

$$\mathrm{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{c=1}^{C} e^{x_c}}$$

其中x_i代表第 i 个元素，C 代表这组数值的元素个数。通过观察上述公式，可以将整个 Softmax 计算流程分为三个步骤：1. 对所有输入数值取 e 的指数次方；2. 完成对步骤 1 得到的中间结果累加求和；3. 将步骤 1 得到中间结果除以步骤 2

得到的中间结果。（具体计算流程图 1 所示）

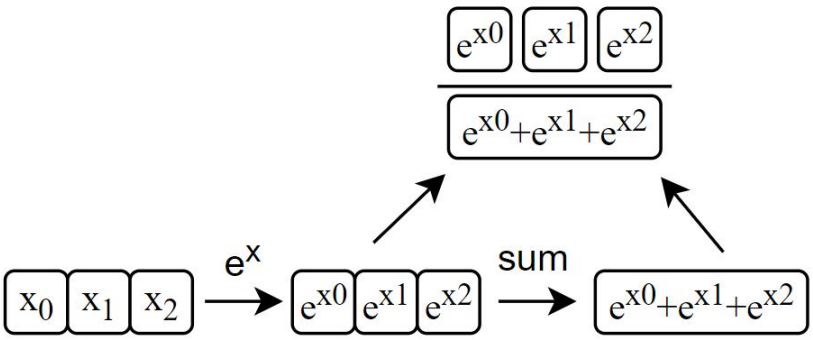

图 1 Softmax 计算流程

根据 Softmax 的计算流程，可以实现一个朴素的版本(baseline)，之后基于朴素的 baseline 版本进行了如下优化：

1. 步骤 1 实现的操作其实和激活函数的 Elementwise 操作基本一致，所以可以很简单的通过向量化指令完成加速。（代码见图 2）

```
for (int i = 0; i < height * width; i++) {
  input[i] = expf(input[i]);
}
-----------------------------------------------------------------
int k;
for (k = 0; k + 32 <= height * width; k += 32) {
  reinterpret_cast<v16f32 *>(&input[k])[0] =
    __s20v_exp_f32__(reinterpret_cast<v16f32 *>(&input[k])[0]);
}
for (; k < height * width; k++) {
  input[k] = expf(input[k]);
}
```

图 2 Softmax 优化 1（上：优化前 下：优化后）

2. 步骤 2 和步骤 3 分别实现了元素的累加求和以及 Elementwise 除法，由于这阶段数据访问可能会存在不对齐的问题，选择使用自动向量化方式进行优化。考虑到 GCU 平台的向量化指令位宽为 1024bit，可以一次处理 32 个 float 类型的数据，所以将循环分成内外两层，外层循环步长为 32 个元素，内层循环步长为 1，依次处理 32 个元素。（代码见图 3）

```
for (int h = 0; h < height; h++) {
  float acc = 0;
  for (int w = 0; w < width; w++) {
    acc += input[h * width + w];
  }
  for (int w = 0; w < width; w++) {
    output[h * width + w] =
      input[h * width + w] / acc;
  }
}
```

```
for (int h = 0; h < height; h++) {
  float acc = 0;
  int w;
  for (w = 0; w + 32 <= width; w += 32) {
    for (int j = 0; j < 32; j++) {
      acc += input[h * width + w + j];
    }
  }
  for (; w < width; w++) {
    acc += input[h * width + w];
  }
  float t = 1 / acc;
  for (w = 0; w + 32 <= width; w += 32) {
    for (int j = 0; j < 32; j++) {
      output[h * width + w + j] =
        input[h * width + w + j] * t;
    }
  }
  for (; w < width; w++) {
    output[h * width + w] =
      input[h * width + w] * t;
  }
}
```

图 3 Softmax 优化 2（左：优化前 右：优化后）

图 4 展示了 Softmax 算子分别在优化 1（opt1）以及优化 1 加优化 2（opt1+opt2）版本相对于 baseline 版本的归一化运行时间的变化，opt1 平均实现 1.30 倍加速比，opt1+opt2 平均实现了 7.30 倍加速比，由此可见自动向量化有着很大的优化潜力。

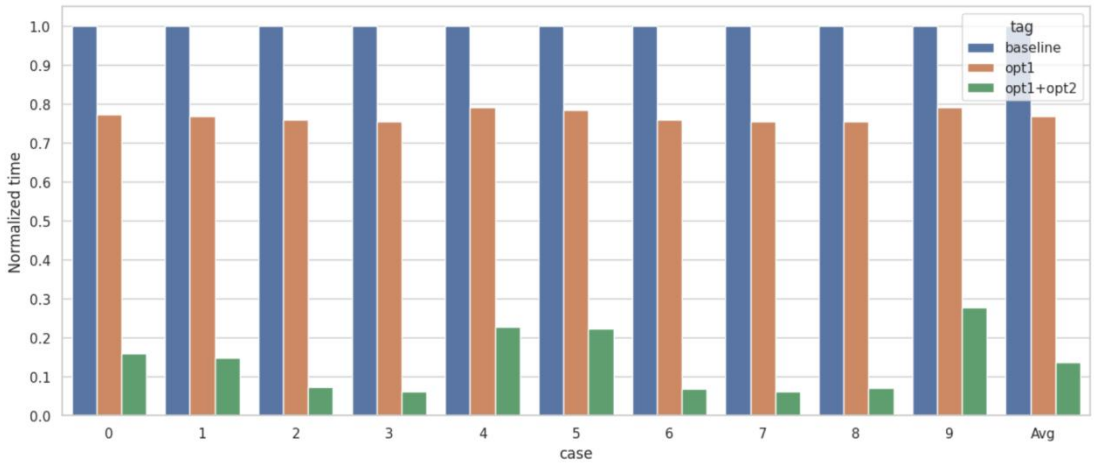

图 4 Softmax 算子优化效果展示

2.2 Maxpooling2d 算子（99.76pts）

池化层是当前卷积神经网络中常用组件之一，用于降低信息冗余,提升模型的尺度不变性、旋转不变性以及防止过拟合等。 最大值池化是最常见、也是用的最多的池化操作。图 5 展示了 Maxpooling2d 的操作示意图，其中输入为 4x4，

步长为 2x2，输出为 2x2。

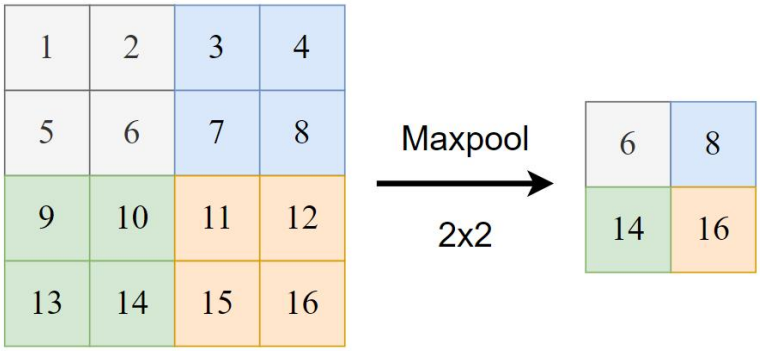

图 5 Maxpooling2d 计算示意图

根据 Maxpooling2d 的计算流程，我们可以构建一个朴素版本（baseline）的实现，之后在 baseline 上进行如下优化：由于输入数据的维度是（height，width，channel），所以不同 channel 的相同 height 和 width 的数据会聚集在一起，而它们都需要和其他 height 和 width 上的数据做比较以取得最大值，所以可以通过数据打包比较来实现最大值的计算。图 6 展示了通过打包操作实现 Maxpooling2d 的示意图，其中输入为（2,2,4），虚线上展示了输入的逻辑视角，虚线下则展示了输入的物理视角，我们可以通过将不同 channel 的相同 height 和 width 数据打包在一起进行最大值的寻找，例如将 Pack1、Pack2、Pack3 和 Pack4 分别对应元素的最大值求出便是最终结果。针对图 6 的优化，分别实现了标量(opt-s)和向量化两个版本(opt-v)，其中标量版本 Pack 长度为 16 和 4，向量版本 Pack 长度为 32 和 4。

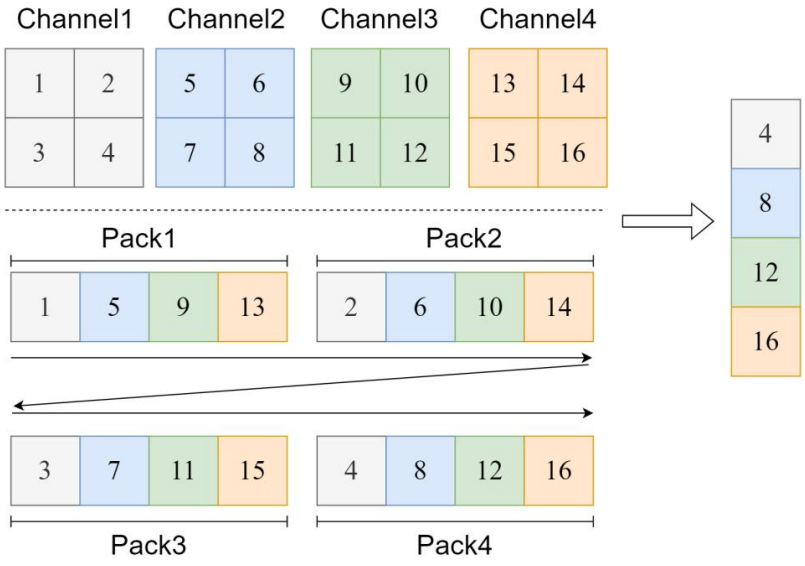

图 6 打包操作实现 Maxpooling2d 示意图（虚线上：逻辑视角 虚线下：物理视角）

图 7 展示了 Maxpooling2d 算子分别在标量(opt-s)和向量(opt-v)优化版本相对于 baseline 版本的归一化运行时间，可以看到 opt-s 平均实现了 1.43 倍的加速比，而 opt-v 平均实现了 10.87 倍加速比，所以手动向量化可以极大地提升处理数据的效率。

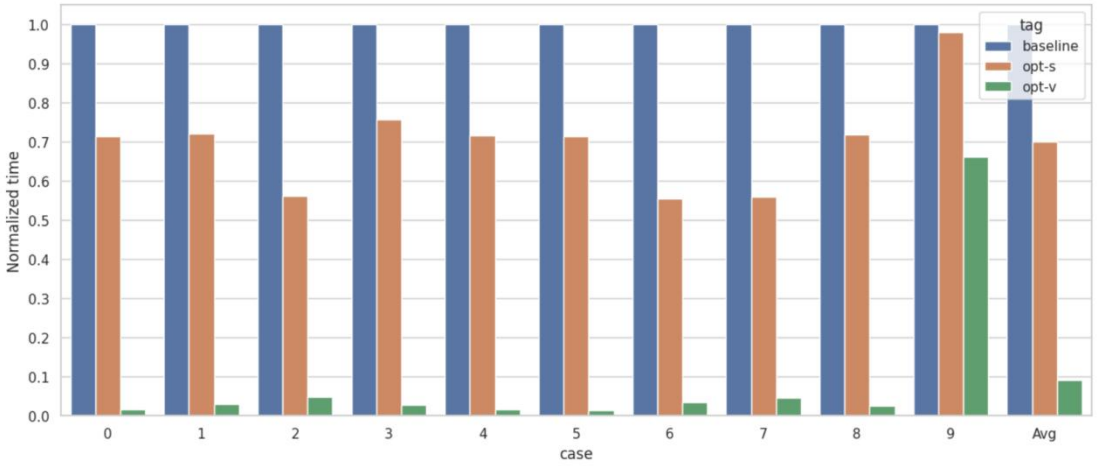

图 7 Maxpooling2d 算子优化效果展示

2.3 Sort 算子（51.53pts）

排序是一个常见的计算机科学和数据处理操作，它在 AI 中有各种应用。例如在强化学习中，排序算法可以用来选择动作或策略，从而选择最佳动作以最大化奖励。在计算机视觉的图像检测中，可以使用排序算法对检测到的对象按其置信度进行排序。

尽管很多排序算法的平均时间复杂度均为 O(nlog(n))，比如归并排序、堆排序和快速排序等，但是快速排序往往在现实中有着很优秀的表现，这源于其 O（1）的空间复杂度以及缓存友好的局部性操作。这一点也在 C++标准模版库得到了印证，其中的 sort 方法使用的是内省排序，可以理解为快速排序+插入排序+堆排序，排序的主体部分采用的是快速排序，对于小数组则采用插入排序，当递归深度超出限制后采用堆排序。因此，首先实现了一个朴素的快速排序版本作为 baseline，之后在 baseline 上做如下优化：1. 当递归遇到的数据长度小于某一阈值后，直接返回，最后采用一遍插入排序完成（插入排序对于相对有序的数组开销较小）；2. 锚点（pivot）的选择对于快速排序至关重要，所以可以采取三数取中法来获得离数组中位数较接近的数字。图 8 展示了优化后的快速排序（opt）相比 baseline

的归一化运行时间，opt 版本平均实现了 1.19 倍加速比。

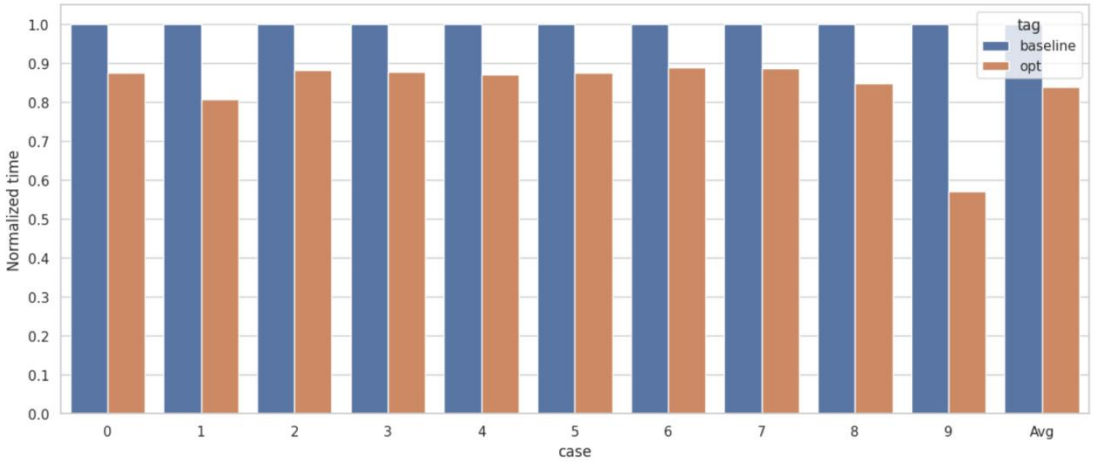

图 8 Sort 算子优化效果

关于向量化排序，之前有工作提出可以基于 Intel 的向量扩展 AVX-512[1]或 AVX-2[2]实现基本数组的双调排序和元素划分。但是上述方法会使用到基于掩码的操作来实现数据 shuffle/permute，而本次实验平台暂不支持这个操作，替代操作开销会较大，故将排序算子的向量化工作留到以后。

2.4 StringMatch 算子（96.3pts）

StringMatch 算子要求实现多匹配串和多模式串的字符串匹配查询。

针对 StringMatch 算子，我们使用如下算法进行优化：

1. 使用 KMP 算法。KMP 是一种改进的字符串匹配算法，由 D.E.Knuth，J.H.Morris 和 V.R.Pratt 提出，它利用模式串的最长的相同的前后缀进行移动，而主串不需要回溯，从而达到快速匹配的目的。

2. 使用 Aho-Corasick automaton（AC 自动机）。AC 自动机是以字典树结构为基础，结合 KMP 思想建立的自动机，用于解决多模式匹配等任务。

3. 使用 Boyer-Moore（BM）算法。BM 算法是一种高效的字符串匹配算法，它的核心思想是将模式串从右往左进行比较，同时用到了两种规则——坏字符规则和好后缀规则，来计算移动的偏移量。

图 9 展示了 AC 和 BM 算法相对于 KMP 算法的归一化运行时间，其中 AC 算法实现了 1.99 倍加速比，BM 算法 21.42 倍加速比，可以看到 BM 算法相比其他字符串匹配算法有着巨大的优势。

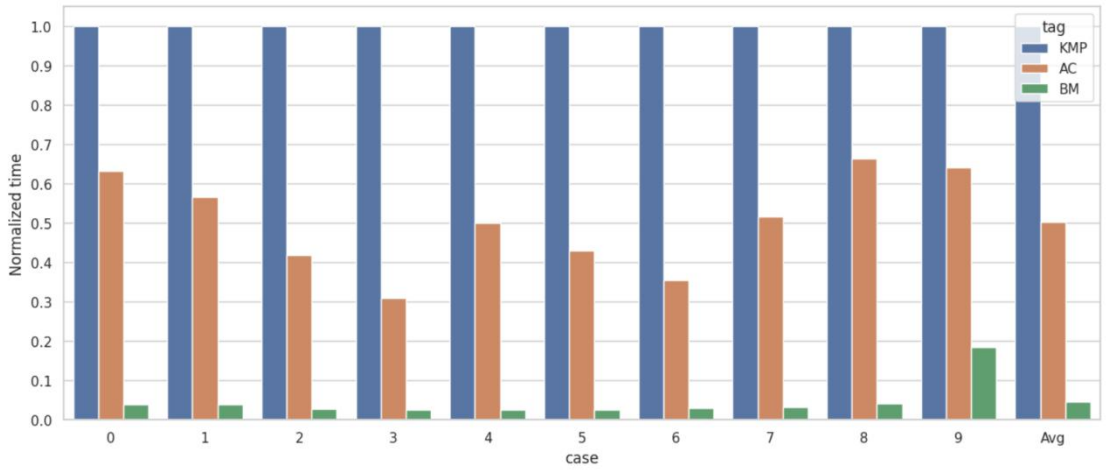

图 9 StringMatch 算子优化效果

3. 引用

[1] Bramas, B. (2017). A Novel Hybrid Quicksort Algorithm Vectorized using AVX-512 on Intel Skylake. International Journal of Advanced Computer Science and Applications, 8(10). https://doi.org/10.14569/ijacsa.2017.081044

[2] Blacher, M., Giesen, J., & Kühne, L. (2021). Fast and Robust Vectorized In-Place Sorting of Primitive Types. In D. Coudert & E. Natale (Eds.), 19th International Symposium on Experimental Algorithms (SEA 2021) (Vol. 190, p. 3:1-3:16). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.SEA.2021.3