第一章 常见编译器的编译过程理解：优化器

1.1 实验目的

通过 clang 编译器进行自动并行化，理解编译过程的代码优化所起的重要作用。

1.2 实验任务

在 Ubuntu 系统下利用 openmp、SIMD 以及常规优化-O2 开启时查看编译器优化的效果，并进行数据分析。

1.3 实验内容

例子代码位置:
https://github.com/NPUCompiler/exp02-understand

1.3.1 SIMD 与自动矢量化优化

SIMD (Single Instruction Multiple Data) ，单指令流多数据流，是一种采用一个控制器来控制多个处理器，同时对一组数据（又称“数据向量”）中的每一个分别执行相同的操作从而实现空间上的并行性的技术。简单来说就是一个指令能够同时处理多个数据，属于数据级并行（data level parallelism）。

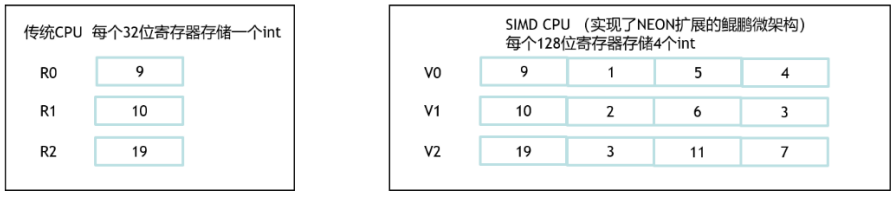

图 1 硬件基础：向量指令集扩展

在并行计算中，自动矢量化是自动并行化的一种特殊情况。在编译过程中，可以自动将计算机程序中一次计算操作处理一个数据的标量实现，转换为一个操作同时处理多个数据的矢量实现，以实现自动矢量优化。这种自动矢量优化在一些具有循环计算的场景下提升显著。

使用 clang 编译器时添加-O2 或-O3 选项可以开启自动矢量优化，使用#pragma clang loop vectorize(disable)可以强制关闭相应矢量优化。

例子程序见 src/example02

编译运行：

（1）直接编译运行，不使用优化，查看运行时间。

```
clang -o test1 test1.c && ./test1
```

（2）直接编译运行，使用优化-O2，查看运行时间。

```
clang -O2 -o test1 test1.c && ./test1
```

（2）把 pragma 语句中的 disable 修改为 enable 后编译运行，查看运行时间。

```
clang -O2 -o test1 test1.c && ./test1
```

（3）去掉 pragma 语句编译运行，查看运行时间。

```
clang -O2 -o test1 test1.c && ./test1
```

请测试多组（至少 5 组），用表格记录原始数据，绘制变化曲线看变化趋势，并且通过反汇编代码分析优化开启前后所使用指令的不同，给出分析的证据。

1.3.2 OpenMP 与线程并行

在并行计算中，线程级并行是除 SIMD 之外的另一种并行模式。对于没有依赖的任务，可以利用计算机的多个 CPU 核来并行执行任务，实现任务的并行执行。多线程编程有多种编程模型，一种是直接借助操作系统提供的 API 或者 pthread 库等实现任务的控制，一种是针对单主机多核或多 CPU 的多线程并行模型 OpenMP。

OpenMP 是在节点内（多核 SMP）执行的基于共享内存的编程模型。OpenMP 是针对单主机上多核/多 CPU 并行计算而设计的工具，换句话

说，OpenMP 更适合单台计算机共享内存结构上的并行计算。由于使用线程间共享内存的方式协调并行计算，它在多核/多 CPU 结构上的效率很高、内存开销小、编程语句简洁直观，现在最新版的 C、C++、Fortran 编译器基本上都内置 OpenMP 支持。不过 OpenMP 最大的缺点是只能在单台主机上工作，不能用于多台主机间的并行计算。对于程序员，OpenMP 是一个可移植、线程化、共享内存的编程规范，它允许将程序分为并行部分与串行部分，隐藏堆栈管理，提供同步化结构。OpenMP 并不能自动并行、保证加速、避免数据冲突。

例子程序见 src/example03。

编译运行：

（1）添加-fopenmp 参数编译运行，查看运行时间；

```
clang -fopenmp -O2 -o test2 test2.c && ./test2
```

（2）与只开启矢量化的例子比较运行时间差异。

```
clang -O2 -o test2 test2.c && ./test2
```

请测试多组（至少 5 组），用表格记录原始数据，绘制变化曲线看变化趋势，并且通过反汇编代码分析优化开启前后所使用指令的不同，给出分析的证据。

另外，请查阅 openmp 的相关资料，实现如果通过#openmp omp sections 与#openmp omp section 配合实现多个任务的并发执行。

1.4 课后作业

（1）学习 OpenMP 通过宏指令进行多线程处理的方式，如果在并行执行的处理中存在数据的互斥方法，如列表中添加元素，OpenMP 如何处理。

（2）尝试编写程序利用#pragma clang loop unroll(enable)进行循环展开进行性能优化验证。例如针对如下的函数进行循环展开、向量化以及 openmp 多线程看优化的效果，并做实验对比。既然是实验，对运行的时间性能进行多组测试并进行比对与分析。

```
1 void array_add(float *arr1, float *arr2, float *result)
2 {
3     for(int i = 0; i < 100000; i++) {
4         result[i] = arr1[i] + arr2[i];
5     }
6 }
```