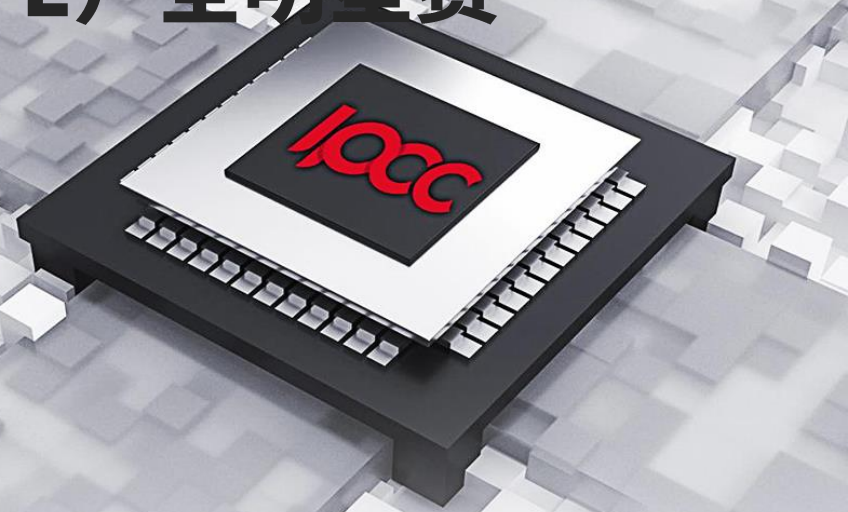
Association for Computing Machinery

ACM中国 国际并行计算挑战赛

AMD

全国大学生高性能计算超级联赛（HPC-PL）全明星赛

暨第三届ACM中国-国际并行计算挑战赛开幕赛



汇报人：黄业琦　2022年5月22日

目录 CONTENTS

01.相关背景
02.优化思考
03.旁门左道
04.总结思考

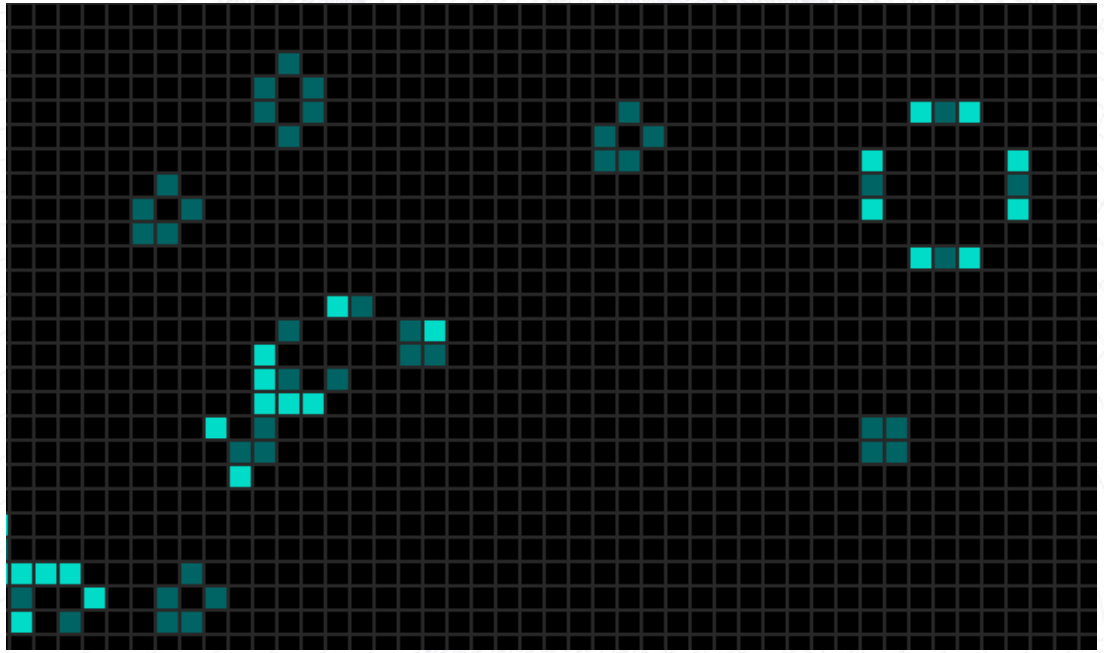
背景：康威生命游戏

生命游戏中，对于任意细胞，规则如下：

每个细胞有两种状态: 存活或死亡

当前细胞为存活状态时:
- 当周围的存活细胞低于2个时，该细胞变成死亡状态。
- 当周围有2个或3个存活细胞时，该细胞保持原样。
- 当周围有超过3个存活细胞时，该细胞变成死亡状态。

当前细胞为死亡状态时:
- 当周围有3个存活细胞时，该细胞变成存活状态。



原始代码分析

Stencil 计算模式：
对周围网格进行访存，可能会破坏空间局部性

在图形图像处理
微分方程计算
都有普遍的应用

```
for (int iter = 0; iter < max_iter; ++iter) {
  printf("Iter %d...\n", iter);
  for (int i = 1; i < N - 1; ++i) {
    for (int j = 1; j < N - 1; ++j) {
      int cnt = 0;
      for (int k = 0; k < 8; ++k) {
        int ii = i + dy[k];
        int jj = j + dx[k];
        if (a[ii][jj] == 1) {
          cnt++;
        }
      }
      if (a[i][j] == 1) {
        if (cnt == 2 || cnt == 3) {
          tmp[i][j] = 1;
        } else {
          tmp[i][j] = 0;
        }
      } else {  // a[i][j] == 0
        if (cnt == 3) {
          tmp[i][j] = 1;
        } else {
          tmp[i][j] = 0;
        }
      }
    }
  }
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      a[i][j] = tmp[i][j];
    }
  }
  #ifdef DEBUG
  char filename[10];
  sprintf(filename, "./output/iter%d", iter);
  FILE* f = fopen(filename, "w");
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < N; ++j) {
      fprintf(f, "%d ", tmp[i][j]);
    }
    fprintf(f, "\n");
  }
  fclose(f);
  #endif
}
```

优化1: INTEL TBB

```
constexpr int blockSize = 66;

tbb::parallel_for(
    tbb::blocked_range2d<size_t>(1, N - 1, blockSize, 1, N - 1, blockSize),
    [&](tbb::blocked_range2d<size_t> const &r) {
      for (int i = r.rows().begin(); i < r.rows().end(); ++i) {
        bool tmp_j[blockSize];
        for (int j = r.cols().begin(); j < r.cols().end(); ++j) {

          ...

        }
      }
    },
    tbb::simple_partitioner{});
```

负优化?

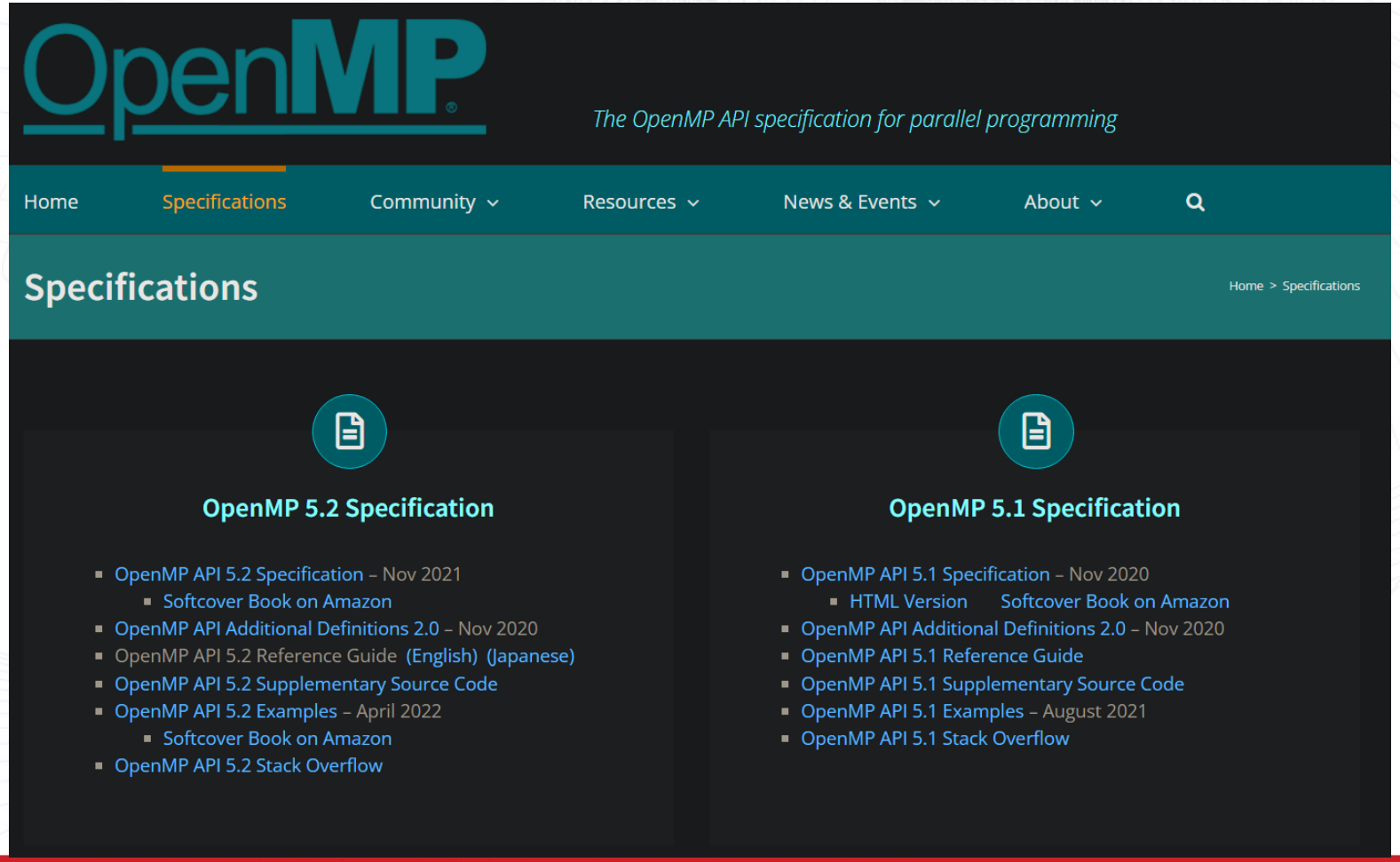
优化2: OpenMP

```
#pragma omp parallel for collapse(2) private(cnt)
```

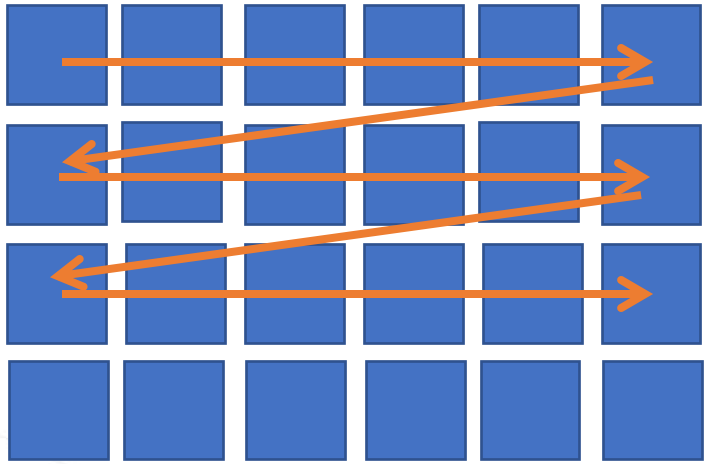
最简单的往往最有效

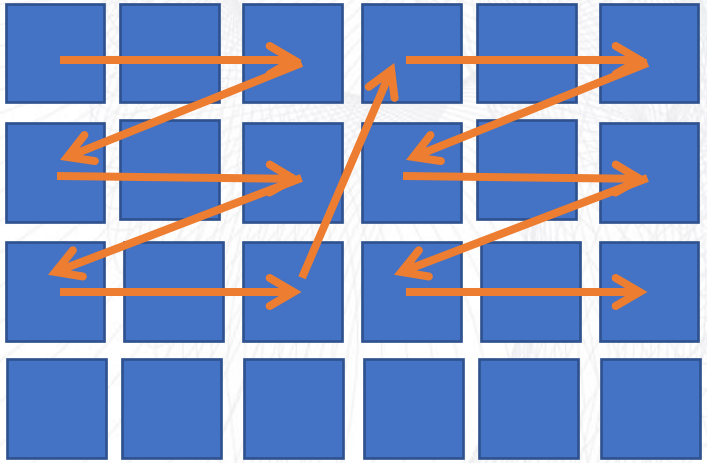
其他尝试：
- OMP SIMD
- 只在外层循环设置omp
- 只在内层循环设置omp

其他潜在的尝试机会：
- OMP TASK



优化3: TILING 充分利用 cache

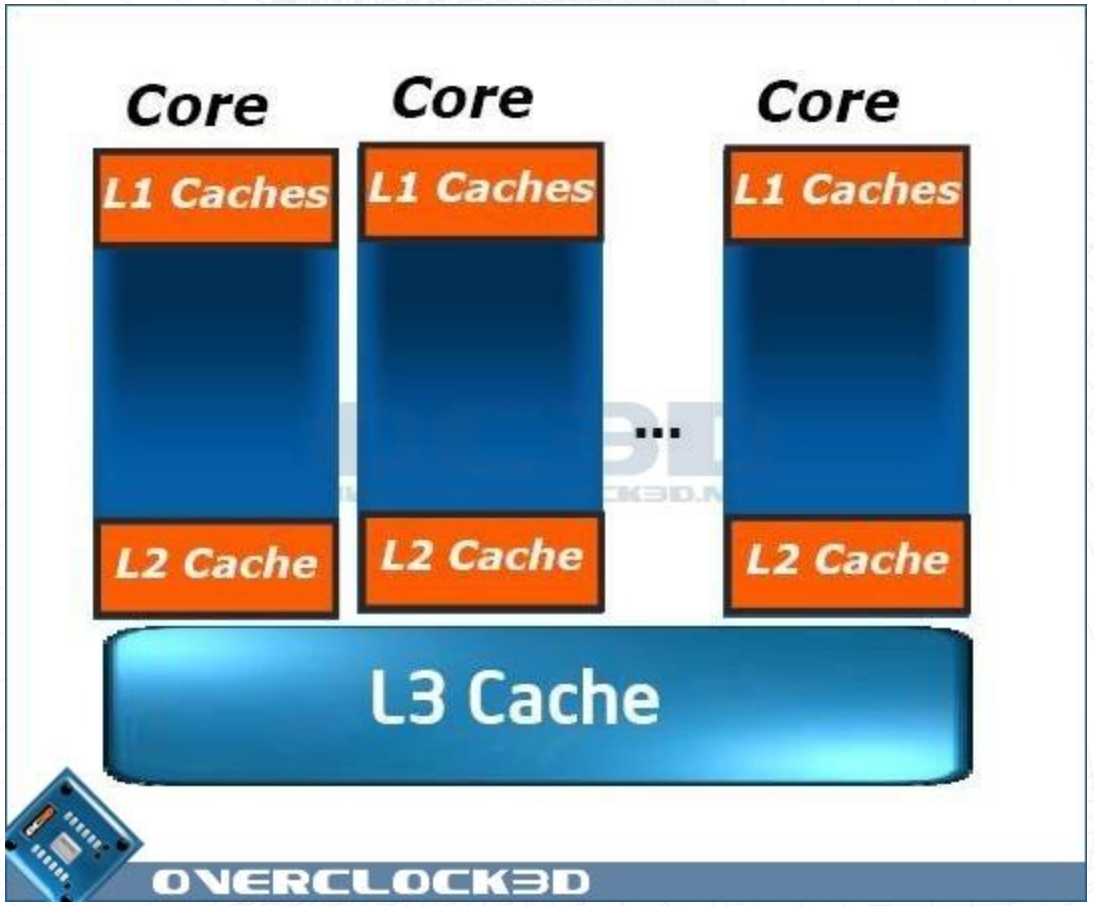




```
#pragma omp parallel for collapse(2) private(cnt)
    for (int jj = 1; jj < N - 1; jj += blockSize) {
      for (int i = 1; i < N - 1; ++i) {
        bool tmp_j[blockSize];
        int sub_j_index = 0;
#pragma GCC unroll 4
        for (int j = jj; j < jj + blockSize; ++j) {
```

优化4: 数据预取和手动代码展开

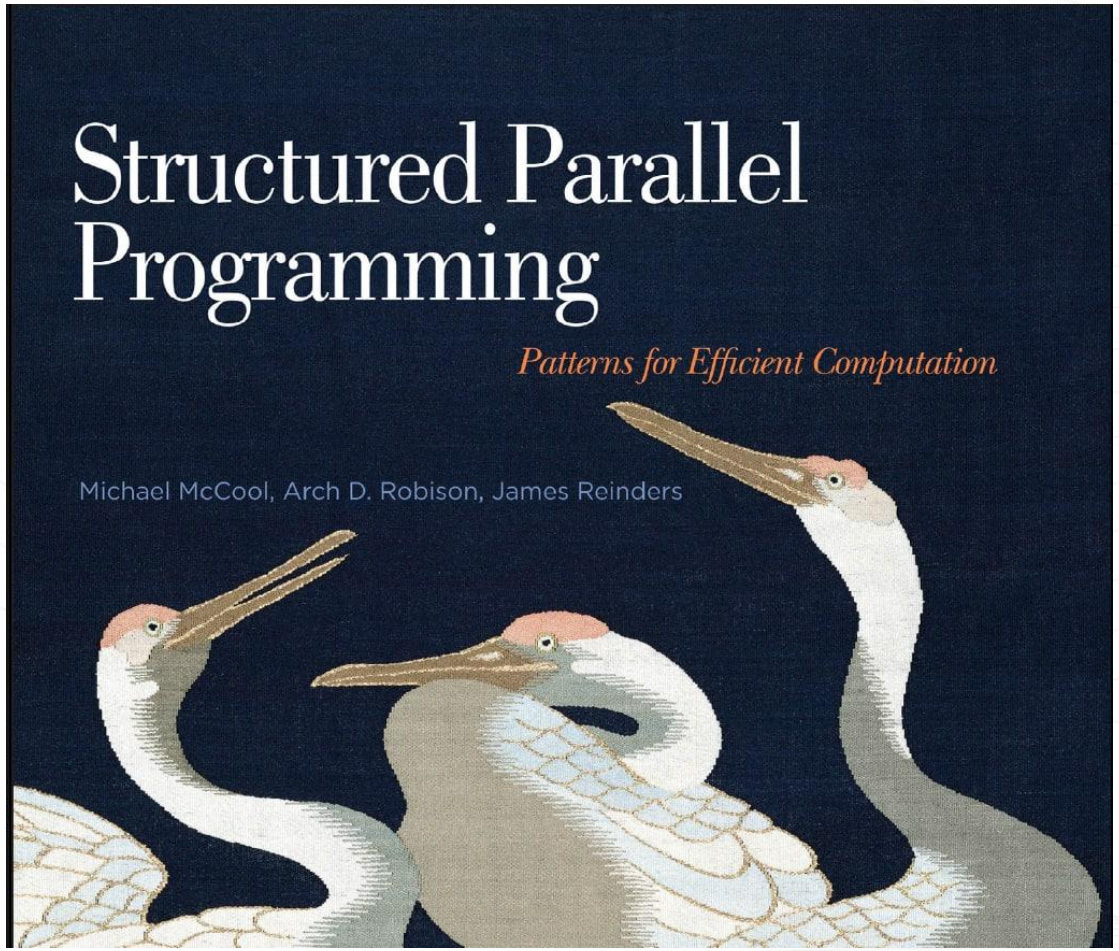
```
for (int j = jj; j < jj + blockSize; ++j) {
  _mm_prefetch(&a[i + 1][j], _MM_HINT_T0);
  _mm_prefetch(&a[i + 2][j], _MM_HINT_T0);
  _mm_prefetch(&a[i + 3][j], _MM_HINT_T1);

  cnt = a[i + dy[0]][j + dx[0]];
  cnt += a[i + dy[1]][j + dx[1]];
  cnt += a[i + dy[2]][j + dx[2]];
  cnt += a[i + dy[3]][j + dx[3]];
  cnt += a[i + dy[4]][j + dx[4]];
  cnt += a[i + dy[5]][j + dx[5]];
  cnt += a[i + dy[6]][j + dx[6]];
  cnt += a[i + dy[7]][j + dx[7]];
```



优化5: 拷贝整合

```
      if (a[i][j] == 1) {
        if (cnt == 2 || cnt == 3) {
          tmp_j[j - jj] = 1;
        } else {
          tmp_j[j - jj] = 0;
        }
      } else { // a[i][j] == 0
        if (cnt == 3) {
          tmp_j[j - jj] = 1;
        } else {
          tmp_j[j - jj] = 0;
        }
      }
    }// j loop
    memcpy(&tmp[i][jj], tmp_j, sizeof(bool) * blockSize);
  }// i loop
}// jj loop
```

好书推荐：



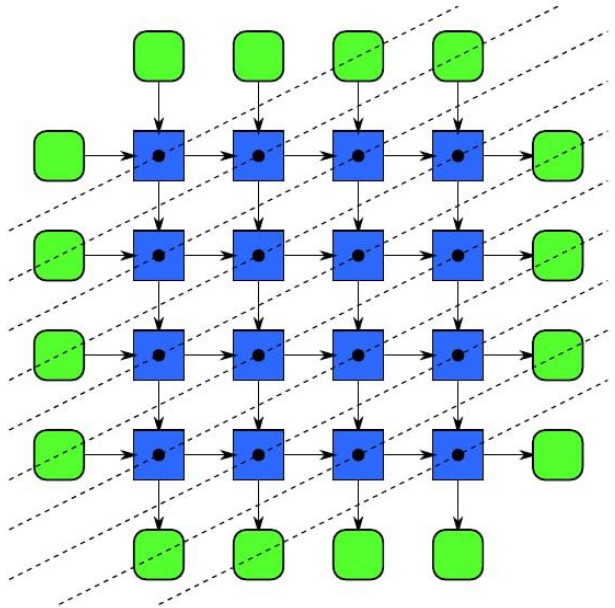


FIGURE 7.4

Recurrence pattern, definition. A multiply nested loop can be parallelized if the data dependencies are regular by finding a separating hyperplane and sweeping it through the lattice. Here, one possible separating hyperplane sweep is shown using a sequence of dotted lines.

```
1  void my_recurrence(
2      size_t v,          // number of elements vertically
3      size_t h,          // number of elements horizontally
4      const float a[v][h], // input 2D array
5      float b[v][h]      // output 2D array (boundaries already initialized )
6  ) {
7      for (int i=1; i<v; ++i)
8          for (int j=1; j<h; ++j)
9              b[i][j] = f(b[i-1][j], b[i][j-1], a[i][j]);
10 }
```

LISTING 7.2

Serial 2D recurrence. For syntactic simplicity, the code relies on the C99 feature of variable-length arrays.

优化5: 编译选项

```
g++ conway.cpp -O3 -fopenmp -o Conway -march=znver1 -mfma -fomit-frame-pointer -mavx2 \
                -mtune=znver1 -m3dnow
```

总结经验（来自我的同一个实验室的同学）：

3.1.1.3. GNU

Table 5. Suggested compiler flags for GNU compilers

Compiler	Suggested flags
gcc compiler	-O3 -march=znver1 -mtune=znver1 -mfma -mavx2 -m3dnow -fomit-frame-pointer
g++ compiler	-O3 -march=znver1 -mtune=znver1 -mfma -mavx2 -m3dnow -fomit-frame-pointer
gfortran compiler	-O3 -march=znver1 -mtune=znver1 -mfma -mavx2 -m3dnow -fomit-frame-pointer

3.1.1.4. AOCC

Table 6. Suggested compiler flags for AOCC compilers

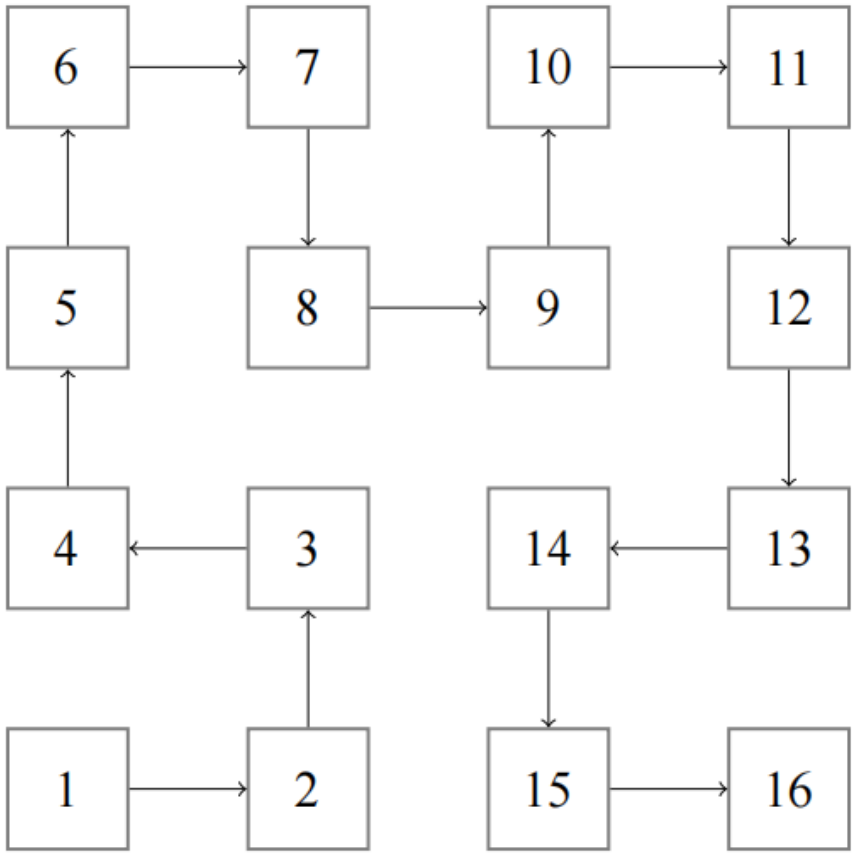
Compiler	Suggested flags
clang compiler	-O3 -march=znver1 -mfma -fvectorize -mfma -mavx2 -m3dnow -floop-unswitch-aggressive -fuse-ld=lld
clang++ compiler	-O3 -march=znver1 -mfma -fvectorize -mfma -mavx2 -m3dnow -fuse-ld=lld
Fortran dragonegg/clang compiler	-O3 -mavx -fplugin-arg-dragonegg-llvm-codegen-optimize=3 -fplugin-arg-dragonegg-llvm-ir-optimize=3

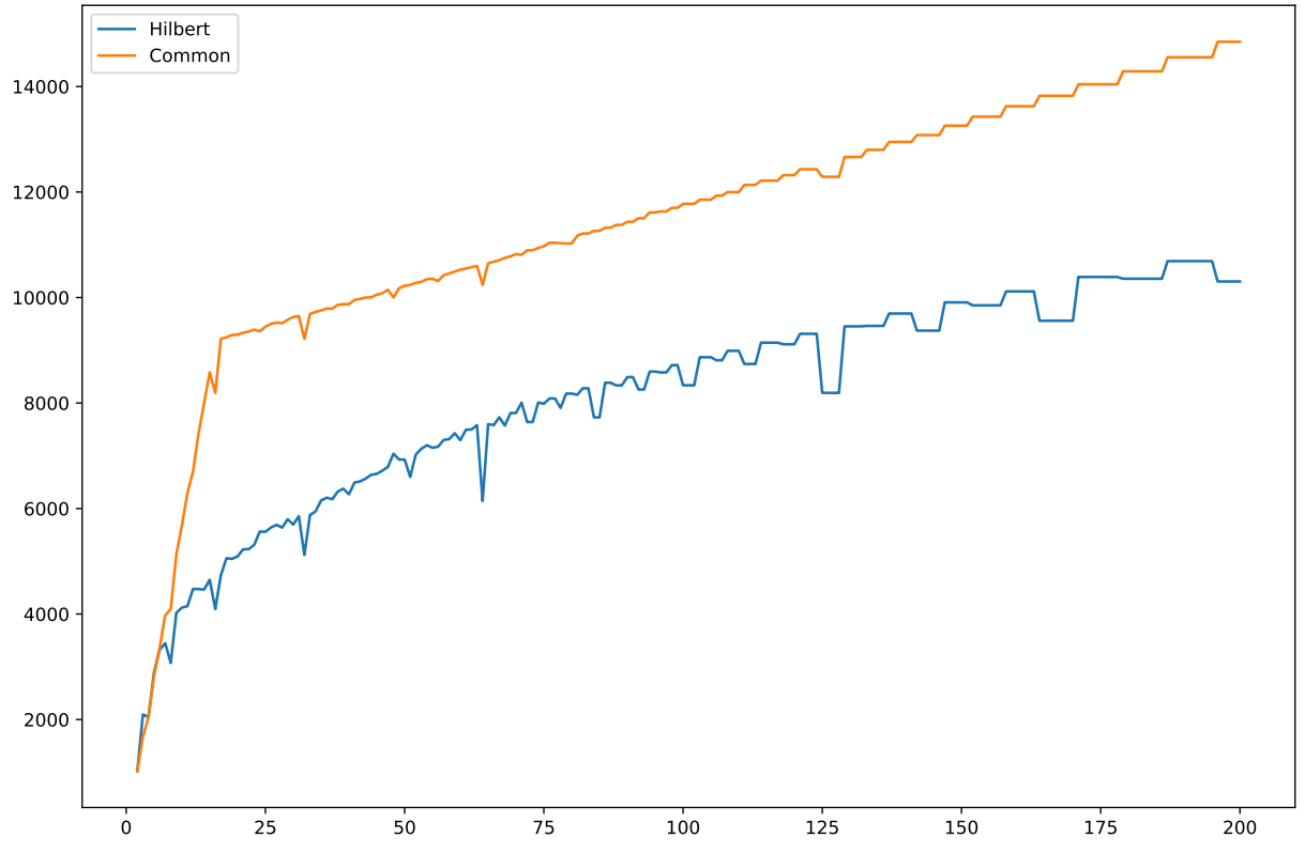
http://home.ustc.edu.cn/~shaojiemike/tags/amd/

旁门左道：

Space Filling Curve：
- 通信上：尽可能减少通信区域面积
- 局部性上：增强空间局部性

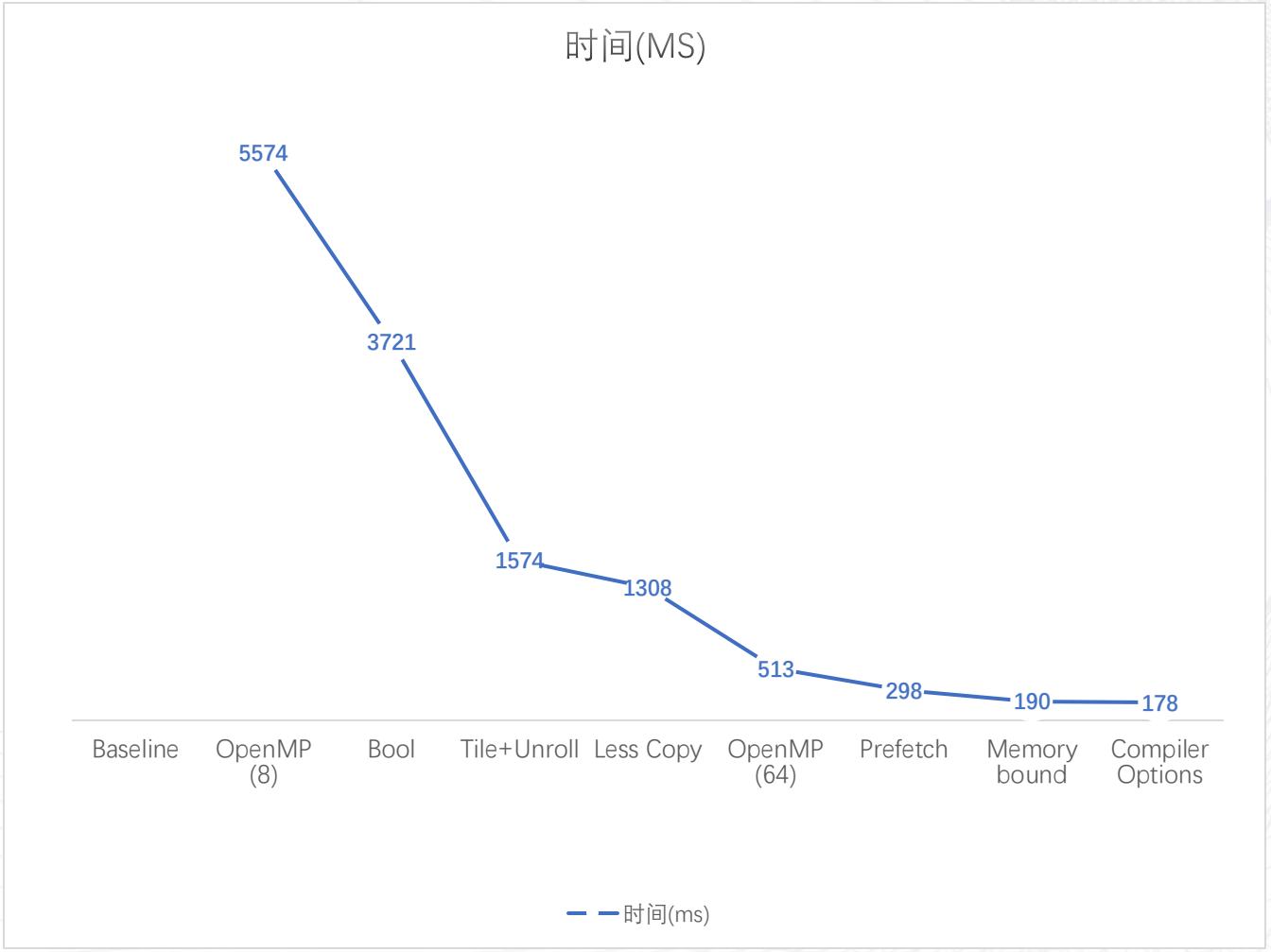
UPCXX：
- PGAS Model
- 减少编码成本，不需要考虑通信
- （代码翻译工具，自动把部分omp变成upcxx）





当然我没有最终采用，是因为我花了30分钟编译UPCXX到并行云上面，然后来不及改代码了qwq

总结


时间(MS)
5574
3721
1574
1308
513
298
190
178
Baseline
OpenMP (8)
Bool
Tile+Unroll
Less Copy
OpenMP (64)
Prefetch
Memory bound
Compiler Options
时间(ms)

总结

1. 团队赛的重要性，优化是一门武功，打一招往往没用
2. 自动化工具不好使，需要大家的努力
3. git 非常重要，我做的乱七八糟的尝试太多，多亏了 git 帮我管理
4. 不要太过相信编译器
5. 经验在这个领域非常重要，希望有更多类似比赛和交流机会

感谢观看

THANKS FOR WATCHING