

飞腾多核处理器的异构编程模型性能评估与优化

实验整体思路：对C版本和SYCL版本进行基准测试，并把C版本的基准测试作baseline。通过对SYCL源代码进行代码手工调优，使得SYCL基准测试的性能达到与对应测试的C语言版本一致甚至更优的结果，并给出优化有效的原因。

实验分为单核和多核，即是否可以并行执行工作项。

单核基准测试选择

1. SYCL-Bench/runtime/dag_task_throughput_independent.cpp
SYCL-Bench/runtime/dag_task_throughput_independent.cpp 测SYCL实现的调度开销和SYCL实现所使用的后端开销
2. PolyBenchC-4.2.1 测C吞吐量
SYCL-Bench/applications 测SYCL吞吐量
3. STREAM benchmark (Copy、Scale、Add、Traid) 测C带宽
SYCL-Bench/micro/DRAM.cpp 在DRAM.cpp基础上使用SYCL补充 Scale、Add、Traid 测SYCL带宽
4. barrier SYCL使用主机设备时不支持显式使用barrier()，使用single_task改写成功的样例

多核基准测试选择

1. SYCL-Bench/runtime/dag_task_throughput_independent.cpp
SYCL-Bench/runtime/dag_task_throughput_independent.cpp 测SYCL实现的调度开销和SYCL实现所使用的后端开销
2. PolyBenchC-4.2.1 测C吞吐量
SYCL-Bench/applications 测SYCL吞吐量
3. STREAM benchmark (Copy、Scale、Add、Traid) 测C带宽
SYCL-Bench/micro/DRAM.cpp 在DRAM.cpp基础上使用SYCL补充 Scale、Add、Traid 测SYCL带宽

由于是对比实验，上述基准测试C实现和SYCL实现的基准测试问题规模对应且都为float类型。

单核实验环境

- ARM 64核处理器（单核）
- GCC (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
- DPCPP-2022-06

多核实验环境

- ARM 64核处理器
- GCC (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0, _OpenMP 201511
- DPCPP-2023-0413, 后端为POCL4.0

单核性能评估与优化

单核基准测试代码、数据都放在

```
~/dcc/sycl-bench/GraduationProject/SingleCore
```

第一类基准测试 测SYCL实现的调度开销和SYCL实现所使用的后端开销

```
代码目录  ~/dcc/sycl-bench/GraduationProject/SingleCore/dag_exp1
运行脚本  ~/dcc/sycl-bench/GraduationProject/SingleCore/run-time-dag-global
          ~/dcc/sycl-bench/GraduationProject/SingleCore//run-time-dag-
          local
```

~/dcc/sycl-bench/GraduationProject/SingleCore/sycl-bench-dag-global-singlecore.csv

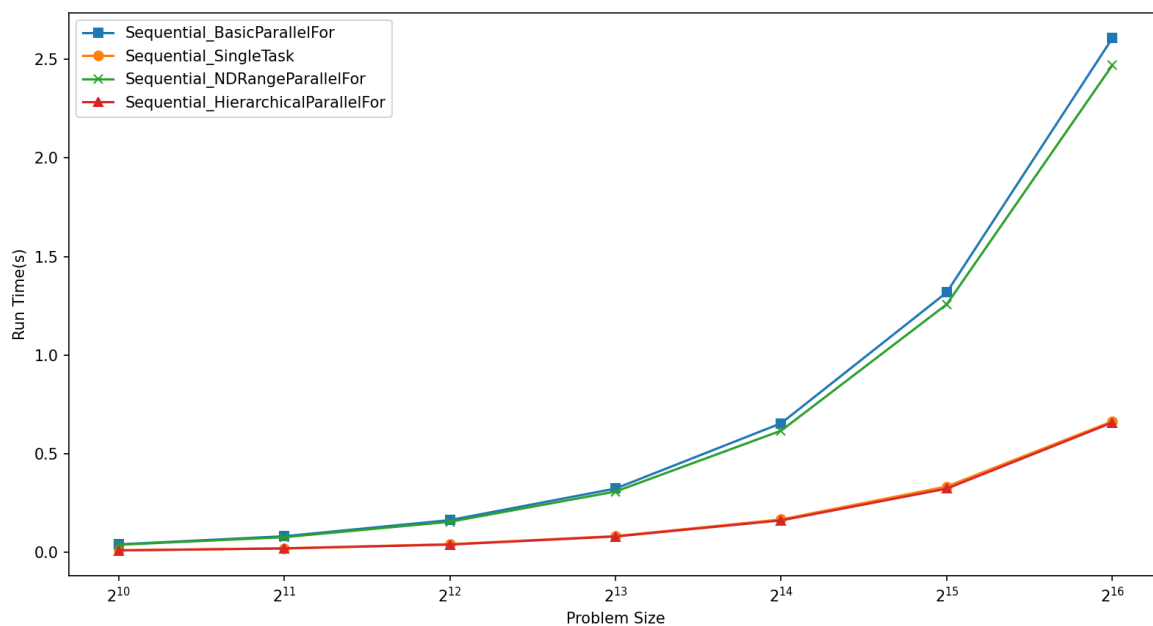
固定local_size为256，分析global_size从1024到65536的影响

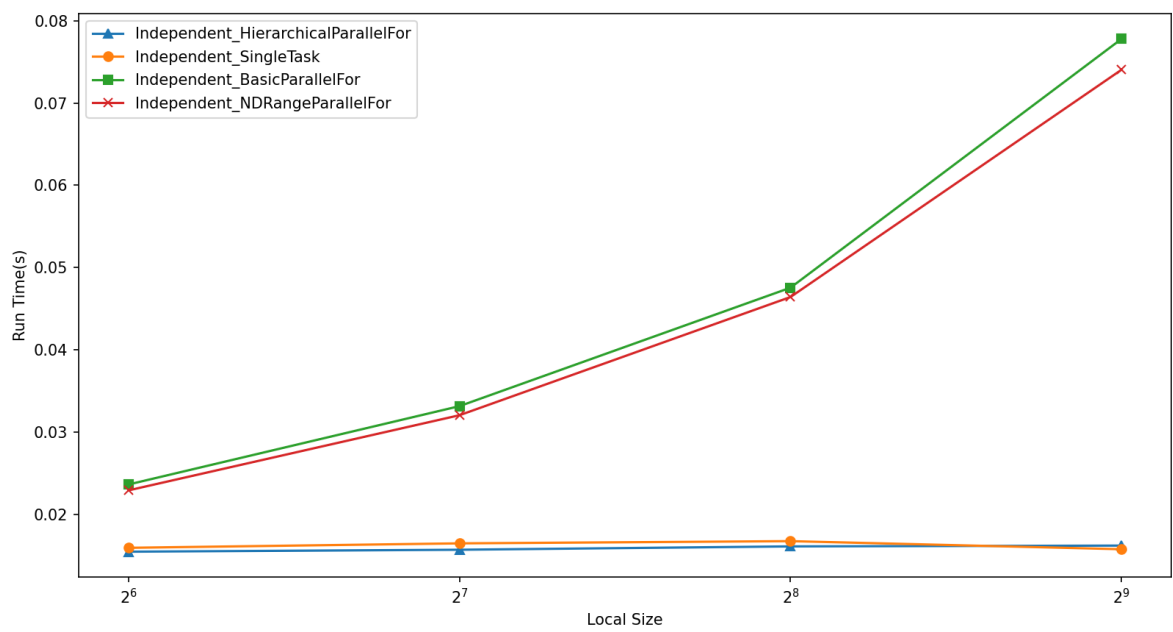
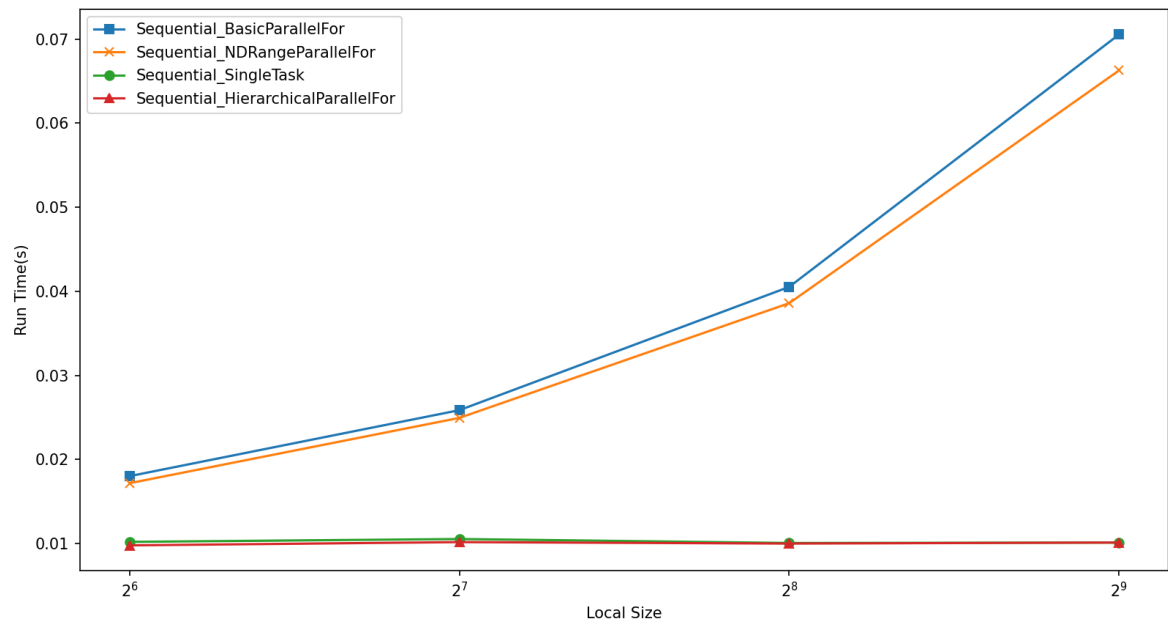
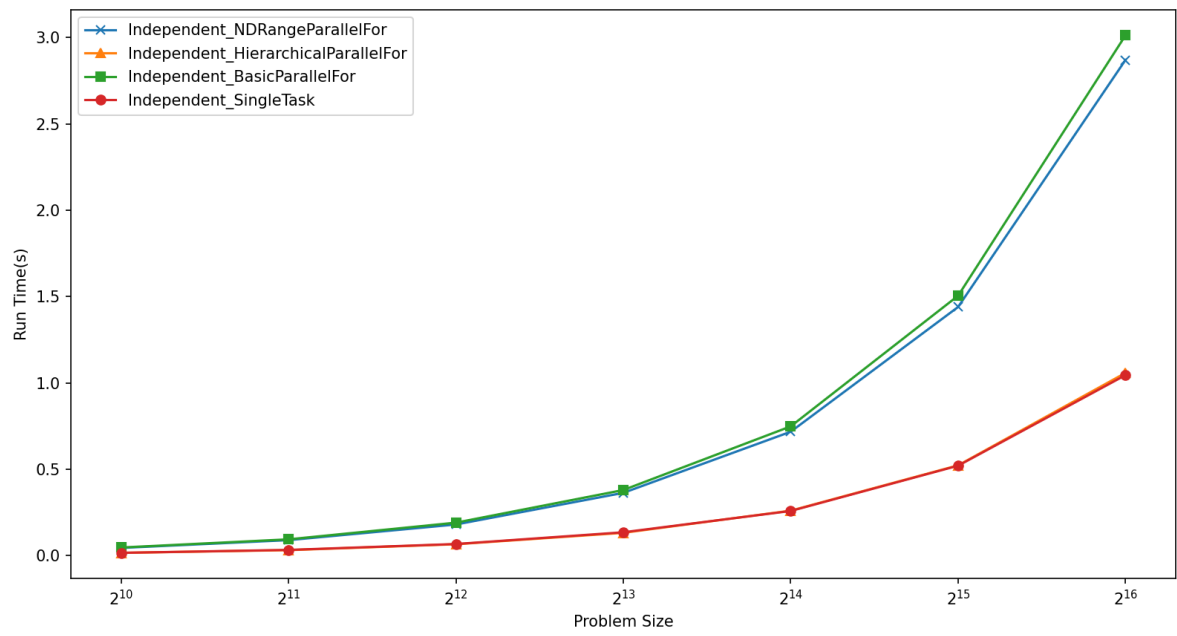
~/dcc/sycl-bench/GraduationProject/SingleCore/sycl-bench-dag-local-singlecore.csv

固定global_size为1024，分析local_size从64到512的影响

未来：无

SYCL实现调度开销

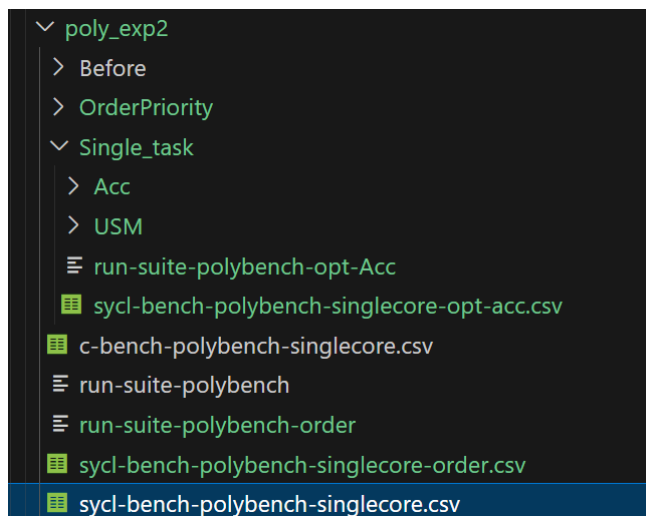




第二类基准测试 测C和SYCL吞吐量

SYCL第二类基准测试目录 ~/dcc/sycl-bench/GraduationProject/SingleCore/poly_exp2

C第二类基准测试目录 ~/dcc/sycl-bench/BenchC/PolyBenchC-4.2.1-master/Test



Before: SYCL/applications原始基准测试代码。

OrderPriority: 通过对SYCL/applications部分基准调整行列顺序后的基准，原因在于PolyBenchC-4.2.1部分基准采取行优先，而SYCL/applications却采取列优先，致使SYCL版本性能不如C版本。（可提高性能）

Single_task/USM: 只修改了gemm.cpp，性能没多大差别，放弃。

Single_task/Acc（**最终优化版**）：在OrderPriority的基础上，通过将basic_parallel_for转换为single_task、减少event.push_back()数量、合并for循环等措施来提高SYCL实现基准测试性能。（可提高性能）

run-suite-polybench-opt-Acc: 自动化执行单核SYCL优化后的基准测试Python脚本。

sycl-bench-polybench-singlecore-opt-acc.csv: Single_task/Acc（最终优化版）的数据。

c-bench-polybench-singlecore.csv: C版本Polybench实验数据。

run-suite-polybench: 自动化执行单核SYCL原基准测试Python脚本

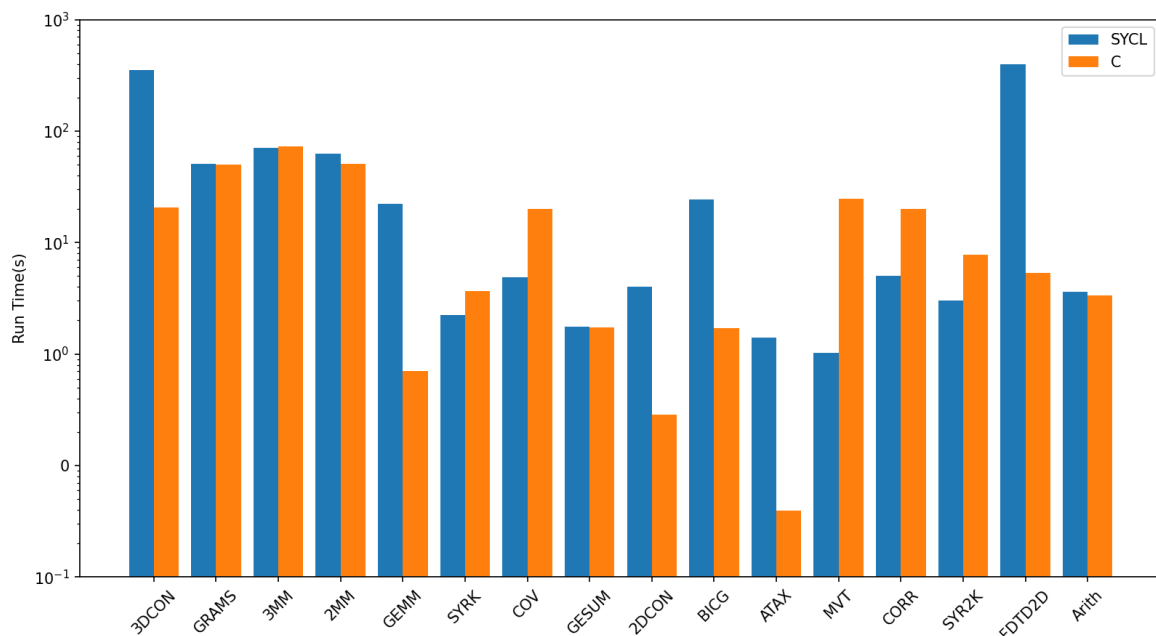
run-suite-polybench-order: 自动化执行单核SYCL行列优化后Python脚本

sycl-bench-polybench-singlecore-order.csv: 单核SYCL行列优化后的数据。

sycl-bench-polybench-singlecore.csv: Before的单核实验数据。

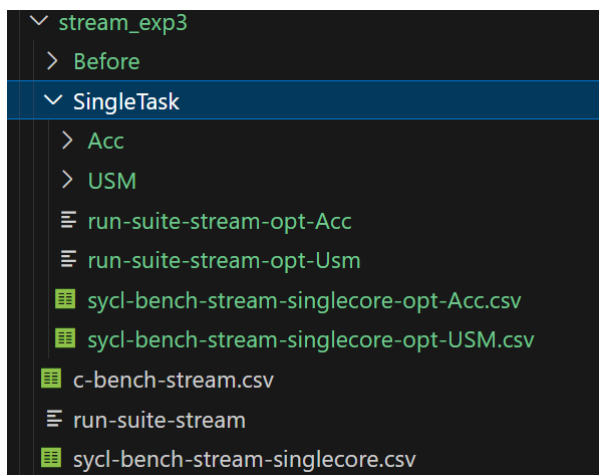
未来：由于SYCL单核优化的吞吐量都达到甚至超过C版本，后续无需在优化。

单核优化前 C和SYCL吞吐量对比



第三类基准测试 测C和SYCL带宽

SYCL第三类基准测试目录 `~/dcc/sycl-bench/GraduationProject/SingleCore/stream_exp3`
 C第三类基准测试目录 `~/dcc/sycl-bench/BenchC/STREAM-master`



Before: SYCL/DRAM Add Scale Traid原始基准测试代码。

Single_task/Acc: 通过将basic_parallel for转换为single_task来提高SYCL实现性能。（可提高性能）

Single_task/USM（最终优化版）：在Single_task/Acc的基础上，将Accessor转换为USM模式。（可提高性能）

run-suite-stream-singlecore-opt-Acc: Single_task/Acc实验数据。

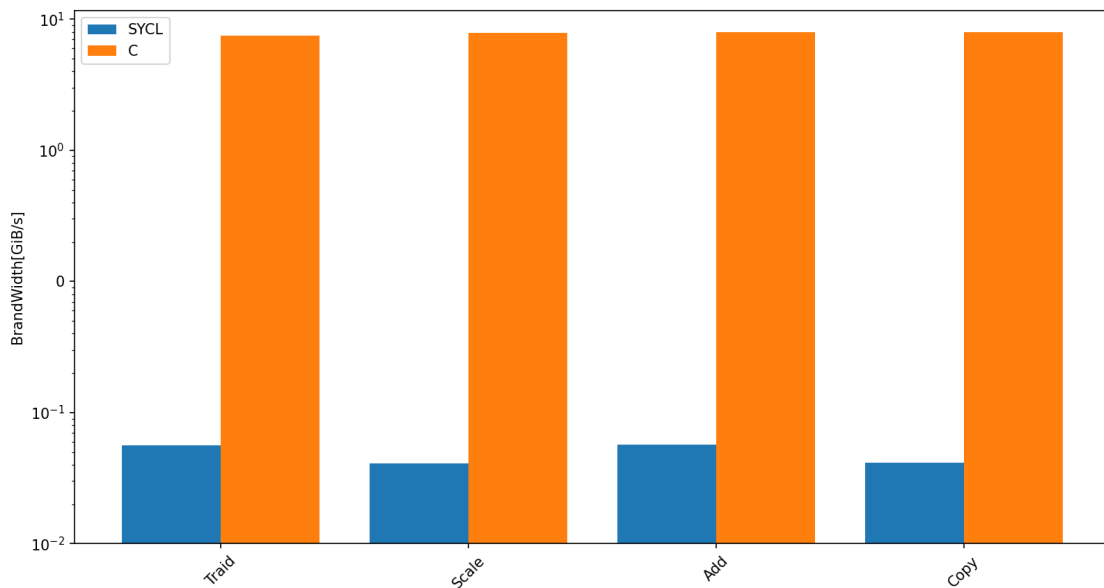
run-suite-stream-singlecore-opt-USM: Single_task/USM（最终优化版）实验数据。

c-bench-stream.csv: C版本stream实验数据。

sycl-bench-stream-singlecore.csv: Before的单核实验数据。

未来：由于SYCL单核优化的带宽还未达到C版本，后续还要进行优化。具体如何优化未知。

单核优化前 C和SYCL带宽对比

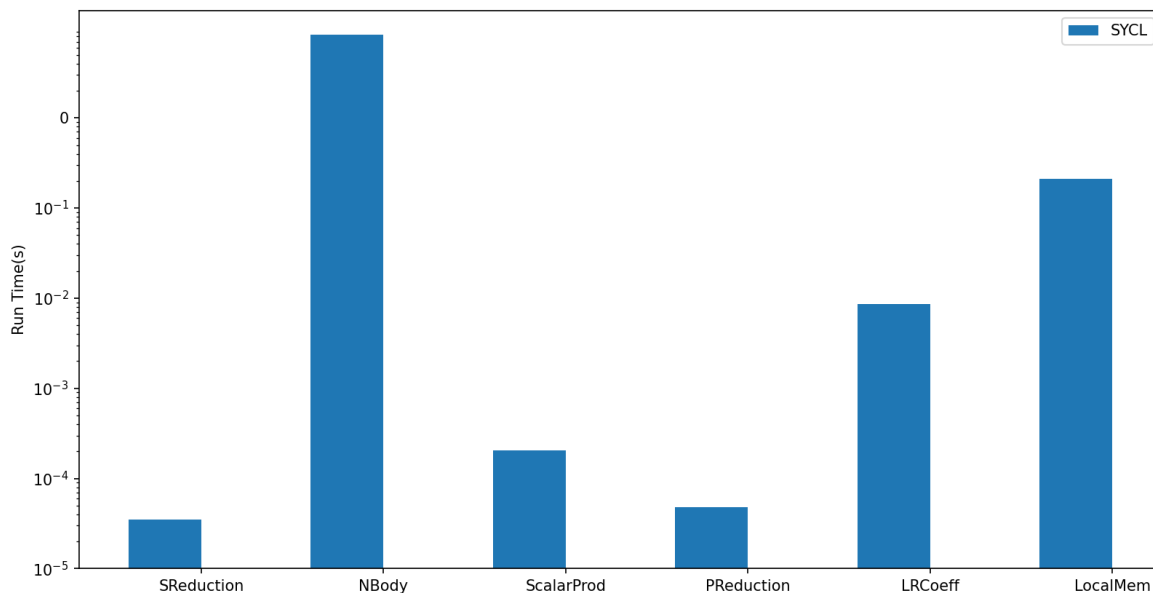


第四类基准测试 SYCL 使用single_task+并行循环分裂+并行循环交换 克服显示使用barrier()问题

SYCL第四类基准测试目录 ~/dcc/sycl-bench/GraduationProject/SingleCore/barrier_exp4
~/dcc/sycl-bench/GraduationProject/SingleCore/run-time-barrier 执行脚本
~/dcc/sycl-bench/GraduationProject/SingleCore/run-suite-barrier barrier实验数据

未来：可以加上Hierarchical parallel_for 对比，因为Hierarchical parallel_for 隐式使用barrier

5个显示使用barrier()基准测试执行情况



多核性能评估与优化

多核基准测试代码、数据都放在

```
~/dcc/sycl-bench/GraduationProject/MultiCore
```

第一类基准测试 测SYCL实现的调度开销和SYCL实现所使用的后端开销

```
代码目录 ~/dcc/sycl-bench/GraduationProject/MultiCore/dag_exp1
运行脚本 ~/dcc/sycl-bench/GraduationProject/MultiCore/run-time-dag-global
          ~/dcc/sycl-bench/GraduationProject/MultiCore/run-time-dag-local
```

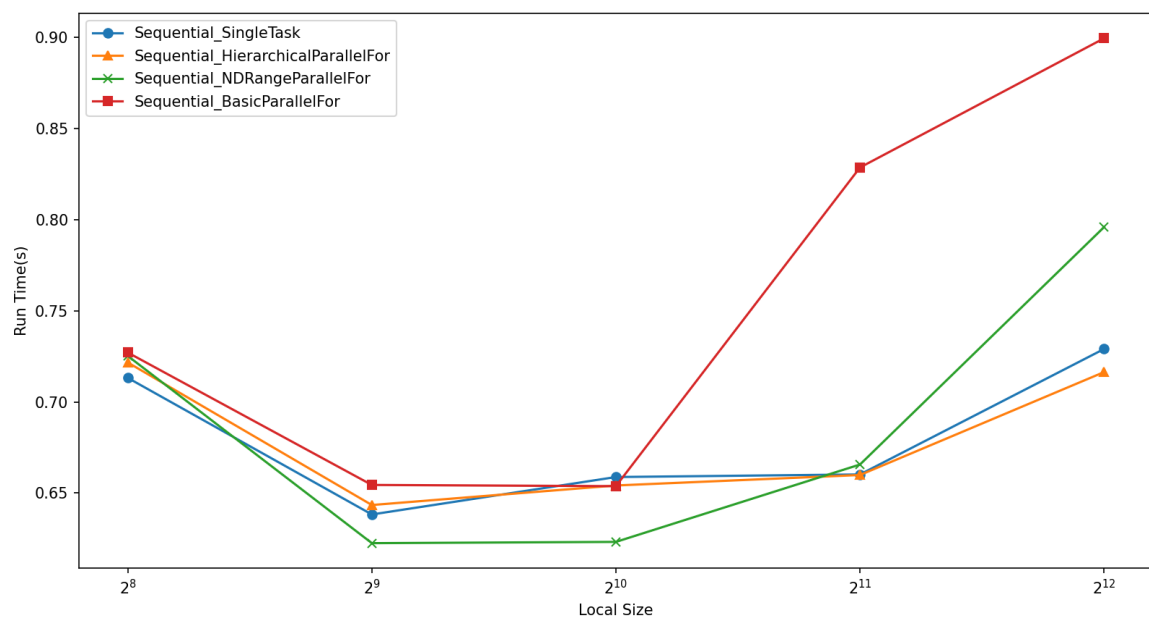
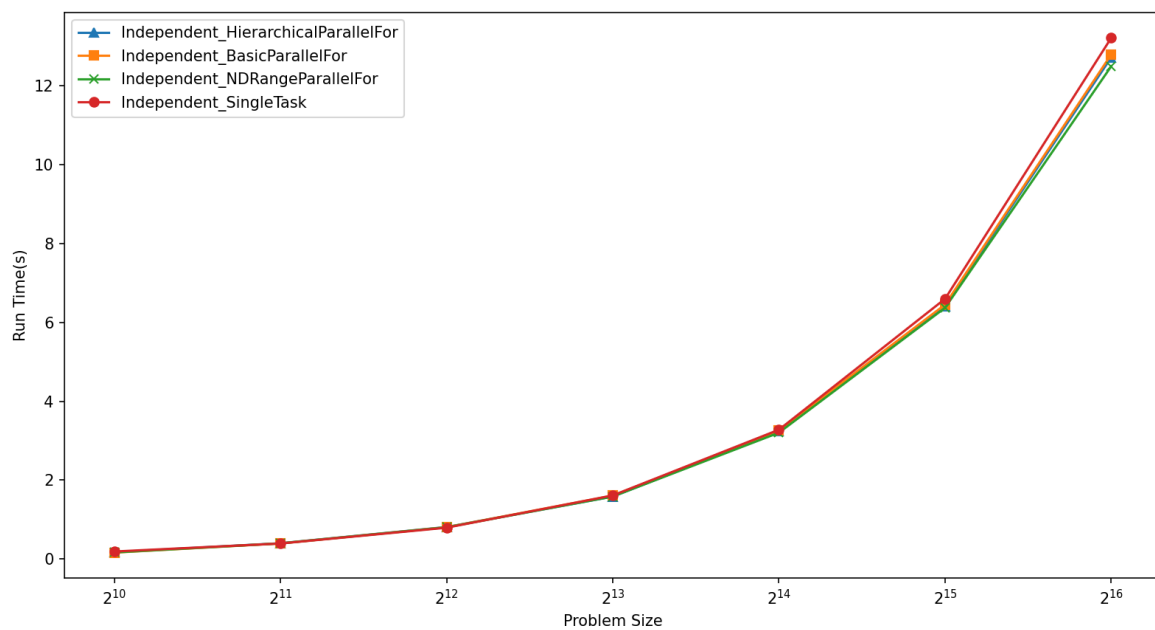
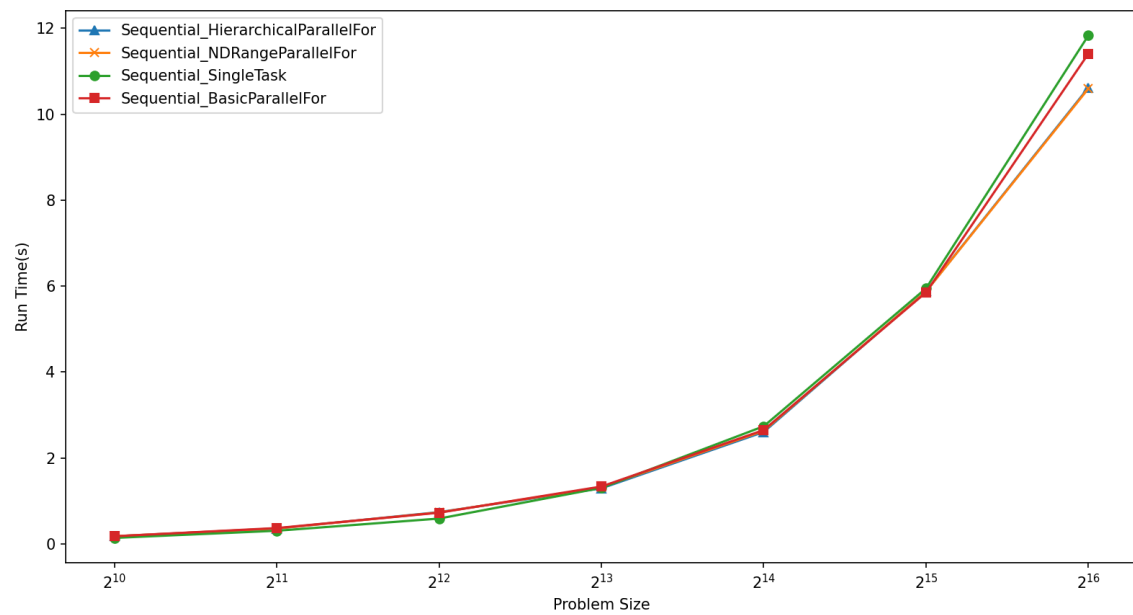
~/dcc/sycl-bench/GraduationProject/MultiCore/sycl-bench-dag-global-singlecore.csv
未绑核时，固定local_size为256，分析global_size从1024到65536的影响

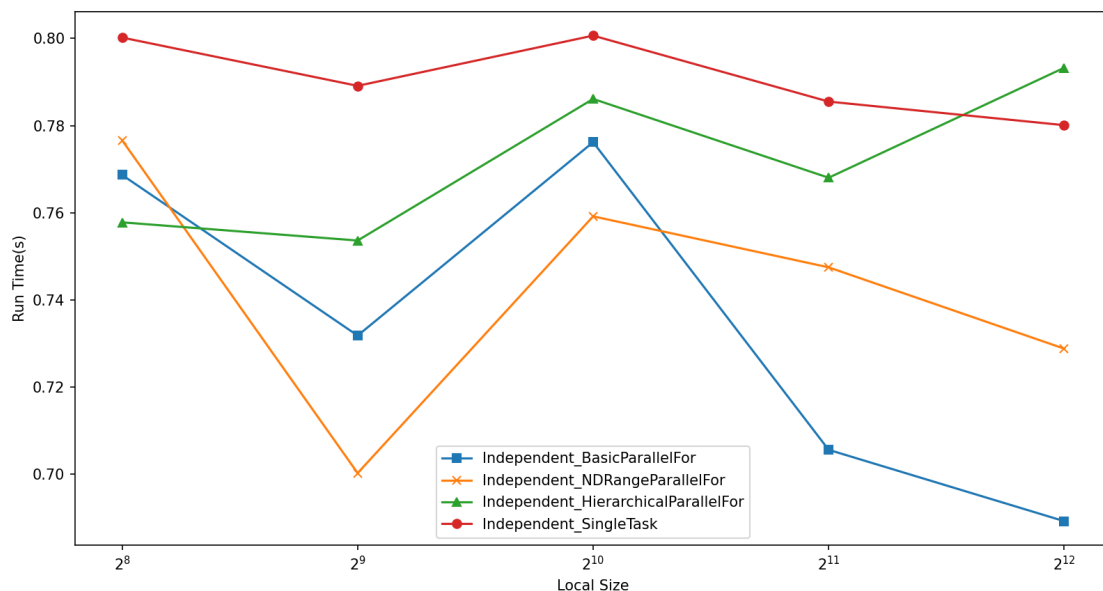
~/dcc/sycl-bench/GraduationProject/MultiCore/sycl-bench-dag-local-singlecore.csv
未绑核时，固定global_size为1024，分析local_size从256到4096的影响

之所以将local_size 最大调整为4096 是因为设备最大支持工作组大小为4096

未来：有必要改为从64到4096吗？感觉没必要

SYCL实现调度开销和SYCL实现所使用的后端开销





第二类基准测试 测C和SYCL吞吐量

SYCL第二类基准测试目录 ~/dcc/sycl-bench/GraduationProject/Multicore/poly_exp2

C第二类基准测试目录 ~/dcc/sycl-bench/BenchC/PolyBenchC-4.2.1-master/Test-fopenmp

```

└─ poly_exp2
   └─ > Before
      └─ > Order-opt
         └─ > USM-opt
            ├── C-bench-polybench-fopenmp.csv
            ├── run-suite-polybench-before
            └── sycl-bench-polybench-multicore.csv
  
```

Before: SYCL/applications原始基准测试代码。

Order-opt: 存放通过解决行列优先（工作项复用/行优先+原子指令）带来的缓存缺失问题可提升性能的基准测试。

USM-opt: 存放通过线程绑核，将Accessor转换为USM，初始化数据采取并行化方式parallel for三种方式可提升性能的基准测试。

C-bench-polybench-fopenmp.csv: 未绑核时 C吞吐量实验数据。

sycl-bench-polybench-multicore.csv: 未绑核时 优化前的SYCL吞吐量实验数据。

暂未测优化后SYCL实现实验数据。（测优化后的数据就是测 Order-opt+USM-opt）

从下图C和SYCL吞吐量对比可知，3DCON、GRAMS、3MM、GEMM、COV、GESUM、2DCON、BICG、ATAX、CORR、FDTD2D需要进行优化。通过分析代码，发现2MM、3MM、GEMM可通过工作项复用来减少缓存缺失带来的性能影响，ATAX和BICG可通过调整行列顺序（注意：这里需要使用原子指令以避免数据竞争）减少缓存缺失带来的影响（这两个基准测试在C中是行优先，但在SYCL中是列优先）。因此，这5个SYCL基准都可认为是行列优先是性能瓶颈，优化后的代码放在**Order-opt**。

注意：分别尝试了Hierarchical parallel for 和 NDRange parallel for，没有很大的区别。

工作项复用方法可见《DPC++》。举例gemm.cpp

```
events.push_back(args.device_queue.submit([&](handler& cgh) {
    auto A = A_buffer.get_access<access::mode::read>(cgh);
    auto B = B_buffer.get_access<access::mode::read>(cgh);
    auto C = C_buffer.get_access<access::mode::read_write>(cgh);

    cgh.parallel_for<Gemm>(C_buffer.get_range(), [=, NK_ = size](item<2>
item) {
        const auto i = item[0];
        const auto j = item[1];

        C[item] *= BETA;

        for(size_t k = 0; k < NK_; k++) {
            C[item] += ALPHA * A[{i, k}] * B[{k, j}];
        }
    });
});
```

```
events.push_back(args.device_queue.submit([&](handler& cgh) {
    auto A = A_buffer.get_access<access::mode::read>(cgh);
    auto B = B_buffer.get_access<access::mode::read>(cgh);
    auto C = C_buffer.get_access<access::mode::read_write>(cgh);

    //local_accessor
    size_t tile_size = args.local_size;
    local_accessor<DATA_TYPE> tileA{tile_size,cgh};

    //sycl::nd_range<2> execution_range{sycl::range<2>{size,size},
sycl::range<2>{args.local_size,args.local_size}};
    //problem_size local_size
    cgh.parallel_for<Gemm>(nd_range<2>{{size,size},{1,tile_size}},
[=,NK_=size](nd_item<2> item){
        size_t m = item.get_global_id()[0];
        size_t n = item.get_global_id()[1];

        size_t i = item.get_local_id()[1];

        DATA_TYPE sum = 0;
        for(size_t kk=0; kk<NK_; kk+=tile_size)
        {
            //load the matrix tile from matrix A and synchronize to
ensure all work-items have a consistent view
            tileA[i] = A[{m, kk+i}];
            item.barrier();

            //perform computation using the local memory tile, and matrix
B in global memory
            for(size_t k=0; k<tile_size; k++)
            {
                sum += ALPHA * tileA[k] * B[{kk+k,n}];
            }
            item.barrier();
        }
    });
});
```

```

    }

    C[{m,n}] *=BETA;
    C[{m,n}] += sum;
  });

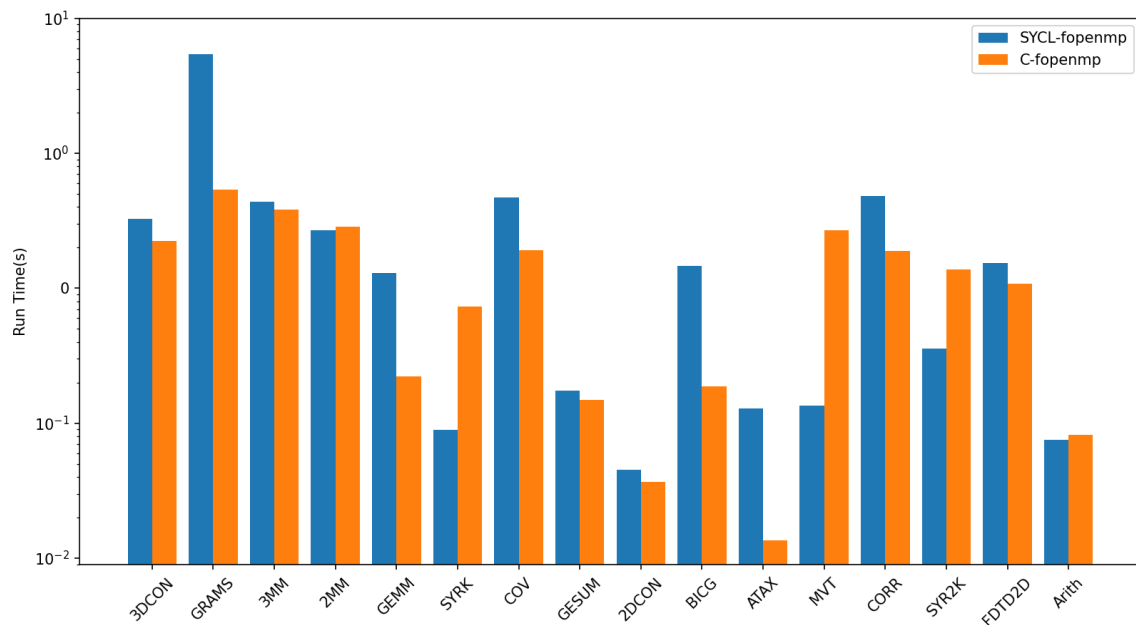
});

```

剩下的6个SYCL基准中都放在**USM-opt**中。2DCON、GESUM、GRAMS可通过线程绑核、将Accessor转换为USM、初始化数据采取并行化方式parallel_for三种方式大大提升性能。虽然GRAMS从开始的55s+降低到10s+，但是还是没达到C版本5s+的要求，存在优化空间。3DCON采取上述三种方式优化会因设备内存不够也无法运行、FDTD2D采取上述三种方式优化性能反而降低，CORR和COV采取上述三种方式优化会无法通过verify（猜测是存在竞争）。

未来：对GRAMS、3DCON、CORR、COV、FDTD2D采取新的优化测量

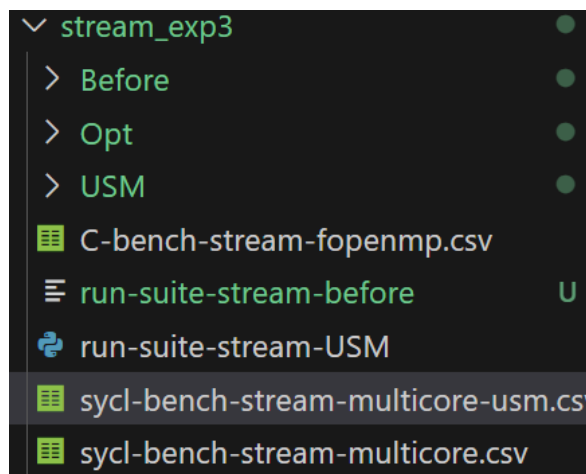
多核优化前，SYCL和C吞吐量对比



第三类基准测试 测C和SYCL带宽

SYCL第三类基准测试目录 ~/dcc/sycl-bench/GraduationProject/Multicore/poly_exp2

C第三类基准测试目录 ~/dcc/sycl-bench/BenchC/STREAM-master



Before: SYCL/DRAM Add Scale Traid原始基准测试代码。

Opt: 分析Hierarchical parallel for 和 NDRange parallel for对带宽的影响，没什么影响，放弃。

USM: 存放通过线程绑核，将Accessor转换为USM，初始化数据采取并行化方式parallel for三种方式可提升性能的基准测试。

sycl-bench-stream-multicore-usm.csv: 绑核后 即优化后的SYCL带宽实验数据。

C-bench-stream-fopenmp.csv: 未绑核时 C带宽实验数据。

sycl-bench-stream-multicore.csv: 未绑核时 优化前的SYCL带宽实验数据。

未来: 由于SYCL多核优化后的带宽还未达到C版本，后续还要进行优化。具体如何优化未知。

多核优化前，SYCL和C带宽对比

