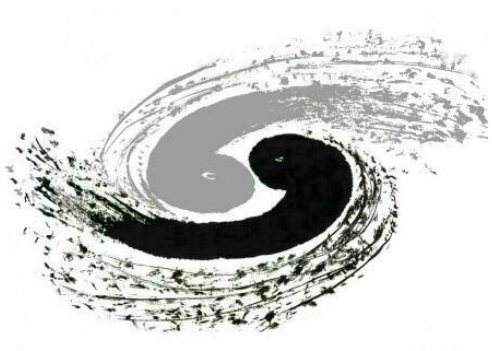


异构计算在径迹重建中的应用

李卫东、林韬、张逸舟

zhangyz@ihep.ac.cn

2023年粒子物理实验计算软件与技术研讨会
山东大学青岛校区

Jun. 11 2023

目录

CONTENT

背景介绍

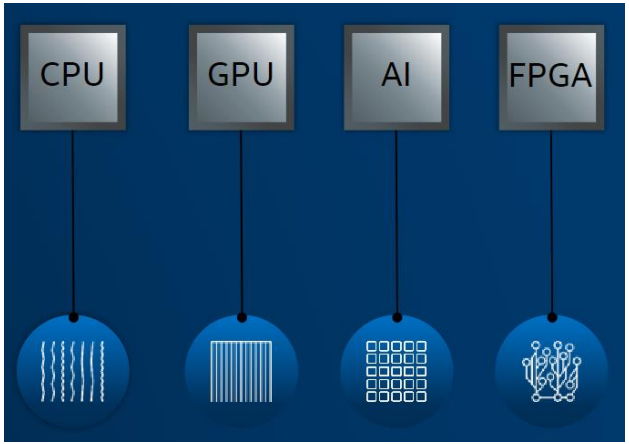
➢ 异构计算

- 异构计算（Heterogeneous Computing），主要指不同类型的指令集和体系架构的计算单元组成的系统的计算方式。
- 传统的单一计算架构面临性能和功耗瓶颈，无法满足日益高涨的算力需求。异构计算是一种解决算力瓶颈问题的思路。
- 相比传统的单一计算架构，异构计算不仅可以提高算力和性能，降低功耗和成本，而且还具备多类型任务的处理能力，发展潜力巨大。

➢ 研究意义

- 高能物理实验软件，例如ROOT、Geant4、DD4hep等，都是经过长期的国际合作在Linux/gcc/x86平台上发展起来的。
- 为了提高性能，高能物理实验（例如HL-LHC实验）正在引入GPU、FPGA等不同的加速卡来加速运算。
- 开发支持异构计算的高性能数据处理软件成为了一个重要的研究方向。

背景介绍

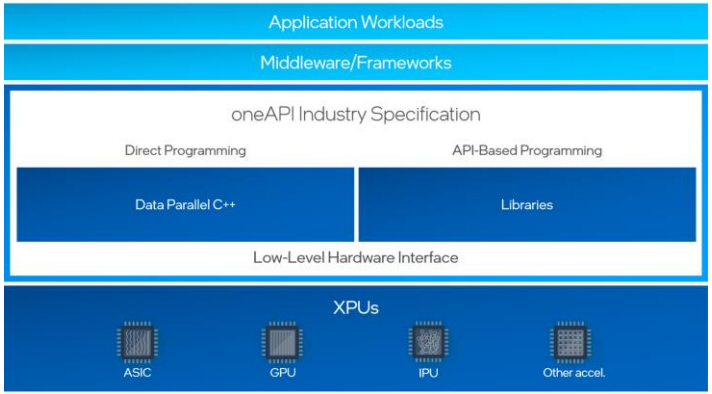


➢ 异构计算面临的困难

- 不同开发框架之间的兼容性差、学习成本高
- 开发环境复杂、框架无法同步更新
- 需要通用的编程语言或API

➢ Intel oneAPI/DPC++

- oneAPI 是统一的软件编程架构，支持多种异构计算单元，包括其他厂商的硬件。
- oneAPI 提供开放、统一的编程语言 DPC++ 和基于 API 的高性能库，能在多种异构平台上运行并提供极高的性能。

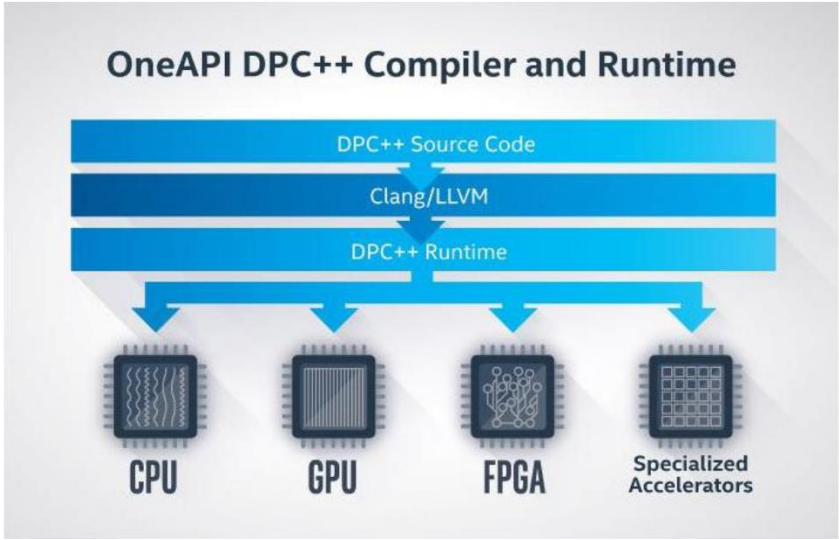


Reference：DPC++ Part 1: An Introduction to the New Programming Model

➢ DPC++(Data Parallel C++)

- DPC++是Intel为了将SYCL引入oneAPI所开发的开源项目。
- SYCL是一种高级别的C++编程模型，是一种跨平台的抽象层，用户不需要关心底层的加速器具体是什么，按照标准编写统一的代码就可以在各种平台上运行。
- 提高了异构计算代码的可移植性和编程效率。

- Data Parallel C++
  = C++ and SYCL* standard and extensions
- Based on modern C++
  C++ productivity benefits and familiar constructs
- Standards-based, cross-architecture
  Incorporates the SYCL standard for data parallelism and heterogeneous programming



Reference：DPC++ Part 1: An Introduction to the New Programming Model

DPC++: 程序示例

Single source

- Host code and heterogeneous accelerator kernels can be mixed in same source files

Familiar C++

- Library constructs add functionality, such as

Construct	Purpose
queue	Work targeting
buffer	Data management
parallel_for	Parallelism

Host code

Accelerator device code

Host code

```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
  auto R = range<1>{ num };
  buffer<int> A{ R };

  queue{}.submit([&](handler& h) {
    auto out =
      A.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });

  auto result =
    A.get_access<access::mode::read>();
  for (int i=0; i<num; ++i)
    std::cout << result[i] << "\n";

  return 0;
}
```

Reference: DPC++ Part 1: An Introduction to the New Programming Model

DPC++: 基于内核的模型

Kernel

- Code that executes on an accelerator, typically many times/instances per kernel invocation (across an ND-range)
- Kernels can be specified using C++ Lambdas

Kernels clearly identifiable in code

- Small number of classes can define a kernel (e.g. parallel_for)

Developer specifies where kernels will run

- Varying levels of control

```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
  auto R = range<1>{ num };
  buffer<int> A{ R };

  queue{}.submit([&](handler& h) {
    auto out =
      A.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });

  auto result =
    A.get_access<access::mode::read>();
  for (int i=0; i<num; ++i)
    std::cout << result[i] << "\n";

  return 0;
}
```

Defines kernel

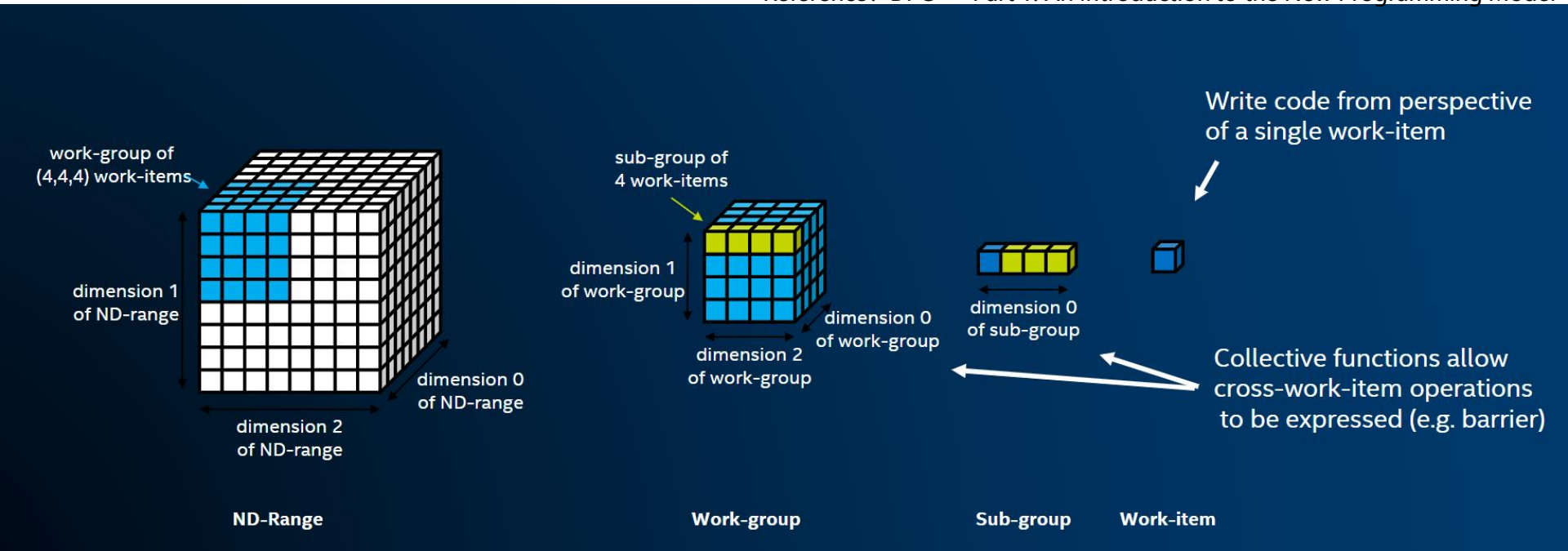
Reference：DPC++ Part 1: An Introduction to the New Programming Model

DPC++: 内核中的执行模型

Data Parallelism is expressed using ND-Ranges

- Total Work = # Work-groups × # Work-items per Work-group
- Bottom-up, hierarchical single program multiple data (SPMD) model

Reference: DPC++ Part 1: An Introduction to the New Programming Model



DPC++: 设备选择

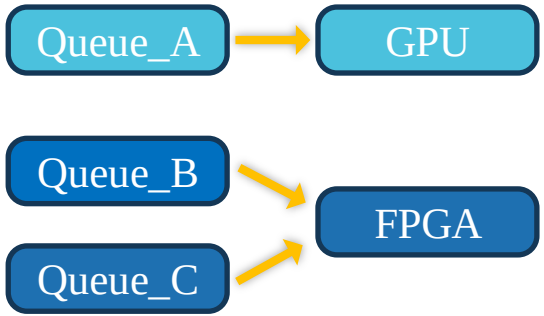
Work is submitted to queues

- Each queue is associated with exactly one device (e.g. a specific GPU or FPGA)
- You can:
  - Decide which device a queue is associated with (if you want)
  - Have as many queues as desired for dispatching work in heterogeneous systems

Create queue targeting any device:	queue();

Create queue targeting a pre-configured classes of devices:	queue(cpu_selector{});
	queue(gpu_selector{});
	queue(intel::fpga_selector{});
	queue(accelerator_selector{});
	queue(host_selector{}); **Always available**

Create queue targeting specific device (custom criteria):	class custom_selector : public device_selector {
	int operator()(..... **// Any logic you want!**
	...
	queue(custom_selector{});

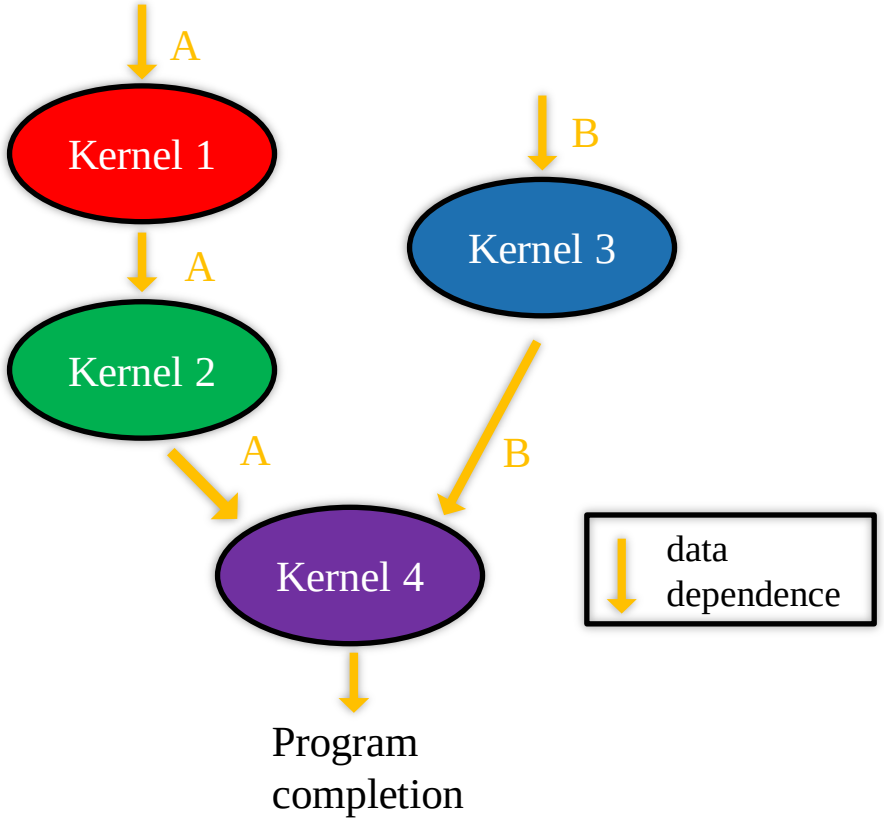


Reference: DPC++ Part 1: An Introduction to the New Programming Model

DPC++:内核执行图

```
int main() {
  auto R = range<1>{ num };
  buffer<int> A{ R }, B{ R };
  queue Q;

  Q.submit([&](handler& h) {
    auto out = A.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });          } Kernel 1

  Q.submit([&](handler& h) {
    auto out = A.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });          } Kernel 2

  Q.submit([&](handler& h) {
    auto out = B.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });          } Kernel 3

  Q.submit([&](handler& h) {
  auto in = A.get_access<access::mode::read>(h);
  auto inout =
    B.get_access<access::mode::read_write>(h);
  h.parallel_for(R, [=](id<1> idx) {
    inout[idx] *= in[idx]; }); });        } Kernel 4
}
```

Automatic data and control dependence resolution!



目录

CONTENT

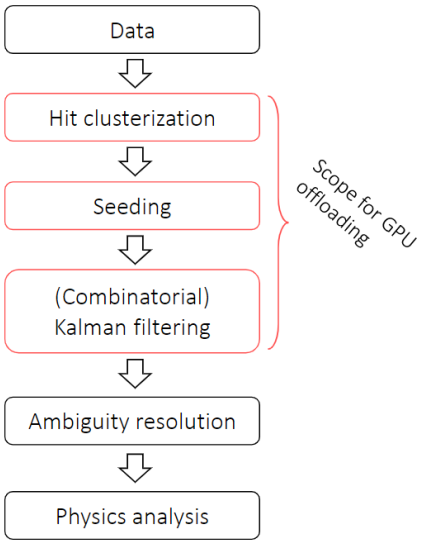
2 异构计算在径迹重建中的应用

Application of heterogeneous computing in track reconstruction

ACTS & TRACCC

ACTS（A Common Tracking Software project）
由CERN主导的开源项目，为未来的探测器设计的通用大型强子对撞机轨道重建软件。

TRACCC为ACTS中正在开发的子项目，目标是设计出能在异构平台上运行的通用径迹重建软件。



径迹重建的主要步骤

Figue from Acts Parallelization R&D

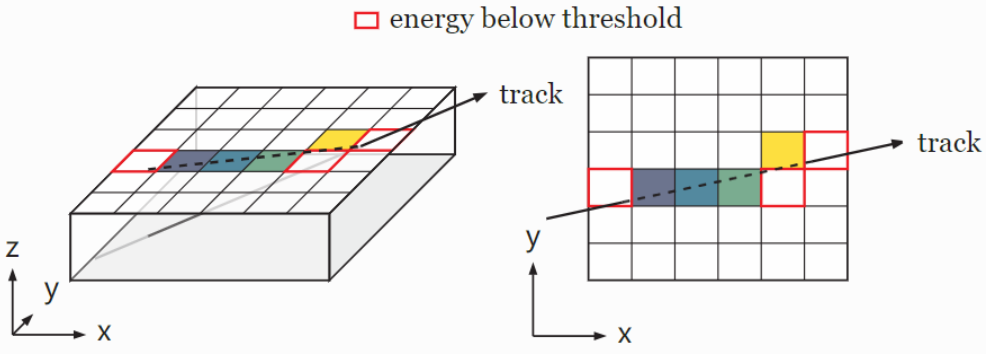
Category	Algorithms	CPU	CUDA	SYCL	Futhark
Clusterization	CCL	✅	✅	✅	✅
	Measurement creation	✅	✅	✅	✅
	Spacepoint formation	✅	✅	✅	○
Track finding	Spacepoint binning	✅	✅	✅	○
	Seed finding	✅	✅	✅	○
	Track param estimation	✅	✅	✅	○
	Combinatorial KF	●	●	○	○
Track fitting	KF	✅	✅	✅	○

✅: exists, ●: work started, ○: work not started yet

TRACCC的目前开发进度

https://github.com/acts-project/traccc

聚类



Figures from ACTS readthedocs

一个像素传感器被带电粒子穿过，从左下方进入，从右上方出来。
红框：能量低于读出阈值
其他四个单元接收到的能量高于阈值，通过聚类形成簇，利用几何信息转换为空间中的三维点。

聚集来自检测器的原始输入，将其转化为可在接下来的步骤中使用的Space Points。

1、读取从检测器中读出的原始输入。
2、基于连通分量标记(Connected-component labeling, CCL)的方法，将cells聚类为簇
3、利用几何信息将簇转换为space points

2 异构计算在径迹重建中的应用

Application of heterogeneous computing in track reconstruction

聚类：TRACCC并行代码

Details::get_queue
获取队列并指定了使用的device
::sycl::local_ccessor
声明的并行计算中共享的变量

h.parallel_for
开始运行device code
cclKernelRange
确定了并行的范围和粒度
[=](::sycl::nd_item<1> item)
将泛型lambda表达式作为对象传入到device上然后进行计算。[=]是捕获列表，它可以捕获当前定义作用域内的变量的值。
(::sycl::nd_item<1> item)是形参列表。

TRACCC具体使用的CCL算法为FastSV算法*，
并行粒度为 8 cells / thread。

Host code

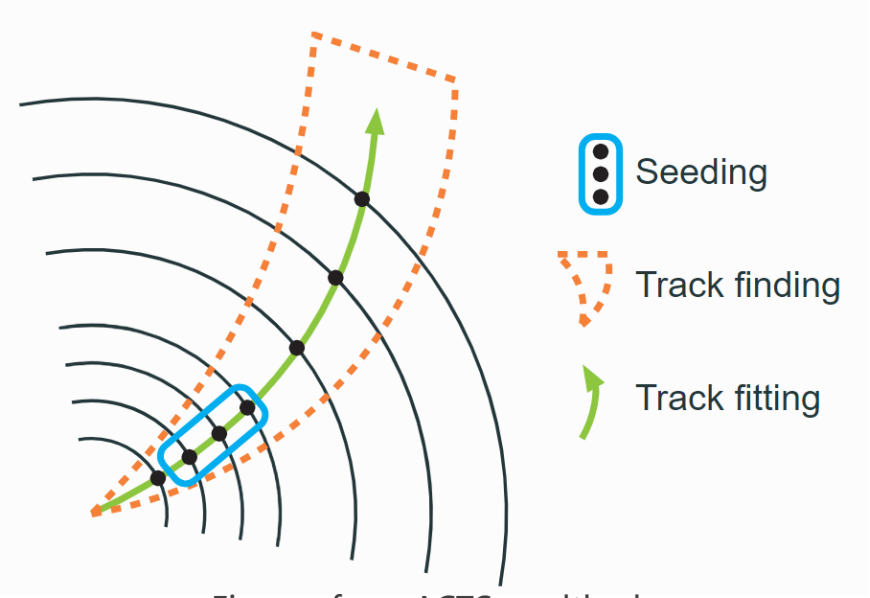
Accelerator device code

Host code

```
// Run ccl kernel
details::get_queue(m_queue)
    .submit([&](::sycl::handler& h) {
        ::sycl::local_accessor<unsigned int> shared_uint(3, h);
        ::sycl::local_accessor<index_t> shared_idx(
            2 * max_cells_per_partition, h);

        h.parallel_for<kernels::ccl_kernel>(
            cclKernelRange, [=](::sycl::nd_item<1> item) {
                index_t* f = &shared_idx[0];
                index_t* f_next = &shared_idx[max_cells_per_partition];
                unsigned int& partition_start = shared_uint[0];
                unsigned int& partition_end = shared_uint[1];
                unsigned int& outi = shared_uint[2];
                traccc::sycl::barrier barry_r(item);

                device::ccl_kernel(
                    item.get_local_linear_id(), item.get_local_range(0),
                    item.get_group_linear_id(), cells, modules,
                    max_cells_per_partition, target_cells_per_partition,
                    partition_start, partition_end, outi, f, f_next,
                    barry_r, measurements_view,
                    *aux_num_measurements_device, cell_links_view);
            });
    })
    .wait_and_throw();
```

*Reference:https://www.sciencedirect.com/science/article/abs/pii/S0743731520302689

seeding



Figures from ACTS readthedocs

1. Space Point组合生成Doublet
2. Doublet扩展成 Triplet
3. 对得到的seed进Confirmation

Space Point组合生成Doublet

Doublet

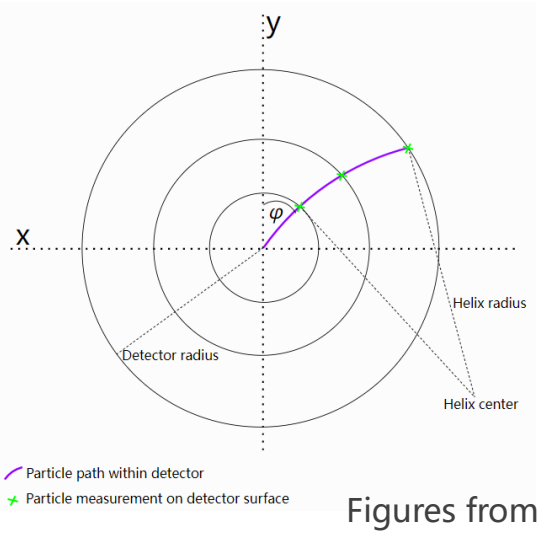
- 并行地遍历每一个Space point
- 遍历该Space point所在的，在φ轴方向上和z轴方向上临近的其他Space points
- 比对两者是否能组成一组middle-top或者middle-bot的doublets

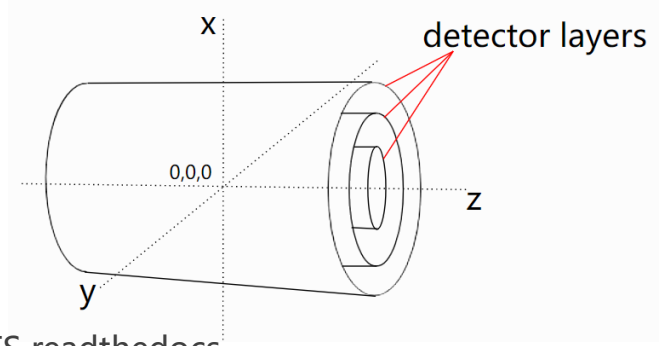
Host code

Accelerator device code

Host code

```
// Find all of the spacepoint doublets.
device::device_doublet_collection_types::view mb_view = doublet_buffer_mb;
device::device_doublet_collection_types::view mt_view = doublet_buffer_mt;
auto find_doublets_kernel =
    details::get_queue(m_queue).submit([&](::sycl::handler& h) {
        h.parallel_for<kernels::find_doublets>(
            doubletFindRange,
            [config = m_seedfinder_config, g2_view, doublet_counter_view,
             mb_view, mt_view](::sycl::nd_item<1> item) {
                device::find_doublets(item.get_global_linear_id(), config,
                                      g2_view, doublet_counter_view,
                                      mb_view, mt_view);
            });
    });
```





Figures from ACTS readthedocs

Doublet扩展成 Triplet

Triplet

- 并行地遍历每一个middle-bot doublet
- 遍历与它共用同一个middle Space point的所有middle-top doublet
- 验证是否能组成一组triplet

Host code

Accelerator device code

Host code

```
// Count the number of triplets that we need to produce.
auto count_triplets_kernel =
    details::get_queue(m_queue).submit([&](::sycl::handler& h) {
        h.parallel_for<kernels::count_triplets>(
            tripletCountRange,
            [config = m_seedfinder_config, g2_view, doublet_counter_view,
             mb_view, mt_view, triplet_counter_spM_view,
             triplet_counter_midBot_view](::sycl::nd_item<1> item) {
                device::count_triplets(
                    item.get_global_linear_id(), config, g2_view,
                    doublet_counter_view, mb_view, mt_view,
                    triplet_counter_spM_view, triplet_counter_midBot_view);
            });
    });
```

```
// Create seeds out of selected triplets
details::get_queue(m_queue)
    .submit([&](::sycl::handler& h) {
        // Array for temporary storage of triplets for comparing within
        // kernel
        ::sycl::local_accessor<triplet> local_mem(
            m_seedfilter_config.max_triplets_per_spM *
                seedSelectingLocalSize,
            h);

        h.parallel_for<kernels::select_seeds>(
            seedSelectingRange,
            [filter_config = m_seedfilter_config, spacepoints_view, g2_view,
             triplet_counter_spM_view, triplet_counter_midBot_view,
             triplet_view, local_mem, seed_view](::sycl::nd_item<1> item) {
                // Each thread uses compatSeedLimit elements of the array
                triplet* dataPos =
                    &local_mem[item.get_local_id() *
                               filter_config.max_triplets_per_spM];

                device::select_seeds(item.get_global_linear_id(),
                                     filter_config, spacepoints_view,
                                     g2_view, triplet_counter_spM_view,
                                     triplet_counter_midBot_view,
                                     triplet_view, dataPos, seed_view);
            });
    })
    .wait_and_throw();
```

Seed select

- 并行地遍历每一个middle space point
- 遍历所有共用该middle space point的所有triplet
- 根据动量和撞击参数进行过滤
- 最后得到seed

径迹参数估计

Track params estimation

- 并行地对每一条seed进行参数估计

Host code

Accelerator device code

Host code

```
details::get_queue(m_queue)
    .submit([&](::sycl::handler& h) {
        h.parallel_for<kernels::estimate_track_params>(
            trackParamsNdRange, [spacepoints_view, seeds_view,
                                 params_view](::sycl::nd_item<1> item) {
                device::estimate_track_params(item.get_global_linear_id(),
                                              spacepoints_view, seeds_view,
                                              params_view);
            });
    })
    .wait_and_throw();
```

2 异构计算在径迹重建中的应用

Application of heterogeneous computing in track reconstruction

使用TRACCC的example程序运行10个event（约200万个cells）

同一份代码在“CPU+GPU”的异构设备上运行，对比了任务分别在CPU和GPU上的运行时间。

运行GPU: NVIDIA Quadro RTX 8000
可以看到GPU在seeding任务中的运行速度远高于CPU。

TRACCC项目地址：
https://github.com/acts-project/traccc

运行结果

```
═══>>> Event 9 <<<═══
Number of spacepoints: 38670 (host), 38670 (device)
  Matching rate(s):
    - 99.3147% at 0.01% uncertainty
    - 99.7414% at 0.1% uncertainty
    - 99.7414% at 1% uncertainty
    - 99.7414% at 5% uncertainty
Number of seeds: 7183 (host), 7183 (device)
  Matching rate(s):
    - 95.2109% at 0.01% uncertainty
    - 99.1369% at 0.1% uncertainty
    - 99.4431% at 1% uncertainty
    - 99.4431% at 5% uncertainty
Number of track parameters: 7183 (host), 7183 (device)
  Matching rate(s):
    - 99.1508% at 0.01% uncertainty
    - 99.819% at 0.1% uncertainty
    - 99.819% at 1% uncertainty
    - 99.8329% at 5% uncertainty
⟹ Statistics ...
- read    380554 spacepoints from 38784 modules
- created        2041344 cells
- created        380554 measurements
- created        380554 spacepoints
- created (sycl) 380554 spacepoints
- created  (cpu) 69313 seeds
- created (sycl) 69313 seeds
⟹Elapsed times...
          File reading  (cpu)  5359 ms
        Clusterization (sycl)  10 ms
        Clusterization  (cpu)  112 ms
   Spacepoint formation  (cpu)  6 ms
               Seeding (sycl)  32 ms
               Seeding  (cpu)  2942 ms
          Track params (sycl)  1 ms
          Track params  (cpu)  20 ms
                     Wall time  8626 ms
[zhangyz@localhost traccc]$
```

目录

CONTENT

3 异构计算的内存优化

Memory for heterogeneous computing

内存拷贝

内存拷贝是异构计算中不得不面对的一大问题。

固然，像径迹重建这类需要频繁并行运行代码的任务，在GPU上运行效率非常高。
但是，当需要运行的数据量非常大，而任务相对较为简单时，“将数据拷贝进显存”这一步骤将占用非常多的时间。

对于一个event（约20万个cell）运行示例：
Copy host->device 90ms
Clusterization 11ms
Seeding 30ms
Track params 1ms

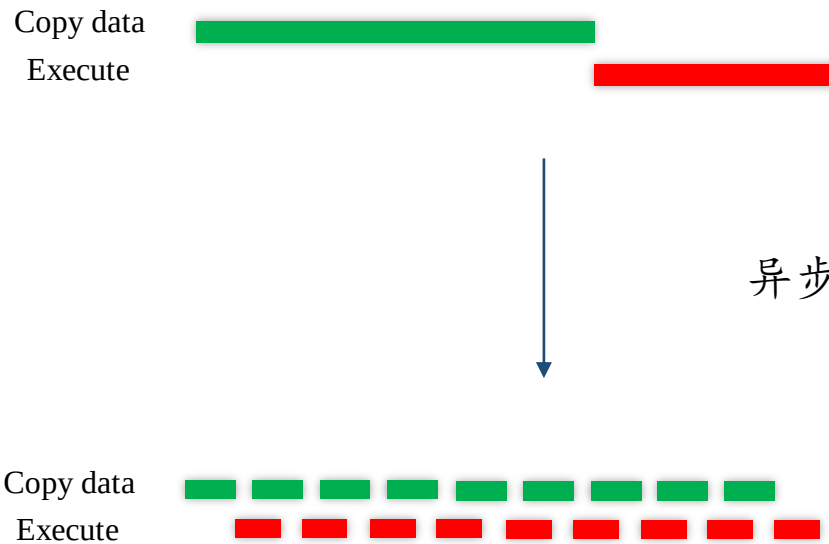
从host拷贝至device占了总运行时间的70%！

```
═══>>> Event 9 <<<═══
Number of spacepoints: 38670 (host), 38670 (device)
  Matching rate(s):
    - 99.3147% at 0.01% uncertainty
    - 99.7414% at 0.1% uncertainty
    - 99.7414% at 1% uncertainty
    - 99.7414% at 5% uncertainty
Number of seeds: 7183 (host), 7183 (device)
  Matching rate(s):
    - 95.2109% at 0.01% uncertainty
    - 99.1369% at 0.1% uncertainty
    - 99.4431% at 1% uncertainty
    - 99.4431% at 5% uncertainty
Number of track parameters: 7183 (host), 7183 (device)
  Matching rate(s):
    - 99.1508% at 0.01% uncertainty
    - 99.819% at 0.1% uncertainty
    - 99.819% at 1% uncertainty
    - 99.8329% at 5% uncertainty
==> Statistics ...
- read    380554 spacepoints from 38784 modules
- created        2041344 cells
- created        380554 measurements
- created        380554 spacepoints
- created (sycl) 380554 spacepoints
- created  (cpu) 69313 seeds
- created (sycl) 69313 seeds
==>Elapsed times...
           File reading  (cpu)  5342 ms
                COPY H->D TIME  90 ms
        Clusterization (sycl)  11 ms
        Clusterization  (cpu)  110 ms
   Spacepoint formation  (cpu)  6 ms
                Seeding (sycl)  30 ms
                Seeding  (cpu)  2950 ms
          Track params (sycl)  1 ms
          Track params  (cpu)  19 ms
                     Wall time  8576 ms
```

3 异构计算的内存优化

Memory for heterogeneous computing

内存拷贝




```
Using CUDA device: NVIDIA GeForce RTX 2060 [id: 0, bus: 1, device: 0]
Reconstructed track parameters: 15976769
Time totals:
                  File reading  1068 ms
            Warm-up processing  753 ms
              Event processing  6392 ms
Throughput:
            Warm-up processing  7.53544 ms/event, 132.706 events/s
              Event processing  6.39269 ms/event, 156.429 events/s
[bash][atspot01]:build-x86_64_ubuntu2004_gcc9 >
```

```
Using SYCL device: NVIDIA GeForce RTX 2060
Reconstructed track parameters: 1659433
Time totals:
                  File reading  821 ms
            Warm-up processing  29 ms
              Event processing  1493 ms
Throughput:
            Warm-up processing  2.95161 ms/event, 338.798 events/s
              Event processing  1.4933 ms/event, 669.659 events/s
```

通过异步拷贝，每个event（约1000个cell）在相同设备下运行速度得到了大幅提升

https://github.com/acts-project/traccc/pull/302

3 异构计算的内存优化

Memory for heterogeneous computing

除了异步向显存拷贝数据之外，我们还希望通过其他方法，修改内存拷贝的方式，以提升异构计算的整体运行效率。

1. 内存访问合并：让并发线程访问的数据尽可能在物理上接近。
2. 自定义内存池来减少系统调用
3. 数据复用：频繁使用的数据放入共享内存中。

…

内存拷贝

```
═══>>> Event 9 <<<═══
Number of spacepoints: 38670 (host), 38670 (device)
  Matching rate(s):
    - 99.3147% at 0.01% uncertainty
    - 99.7414% at 0.1% uncertainty
    - 99.7414% at 1% uncertainty
    - 99.7414% at 5% uncertainty
Number of seeds: 7183 (host), 7183 (device)
  Matching rate(s):
    - 95.2109% at 0.01% uncertainty
    - 99.1369% at 0.1% uncertainty
    - 99.4431% at 1% uncertainty
    - 99.4431% at 5% uncertainty
Number of track parameters: 7183 (host), 7183 (device)
  Matching rate(s):
    - 99.1508% at 0.01% uncertainty
    - 99.819% at 0.1% uncertainty
    - 99.819% at 1% uncertainty
    - 99.8329% at 5% uncertainty
⟹ Statistics ...
- read    380554 spacepoints from 38784 modules
- created        2041344 cells
- created        380554 measurements
- created        380554 spacepoints
- created (sycl) 380554 spacepoints
- created  (cpu) 69313 seeds
- created (sycl) 69313 seeds
⟹Elapsed times...
          File reading  (cpu)  5342 ms
               COPY H→D TIME  90 ms
        Clusterization (sycl)  11 ms
        Clusterization  (cpu)  110 ms
   Spacepoint formation  (cpu)  6 ms
               Seeding (sycl)  30 ms
               Seeding  (cpu)  2950 ms
         Track params (sycl)  1 ms
         Track params  (cpu)  19 ms
                    Wall time  8576 ms
```

Vecmem

VecMem是ACTS项目的一部分

Vecmem提供了支持矢量化的数据模型的内存管理容器。

Vecmem使用std::pmr提供的内存配置类，可以在host端自定义内存分配方案，实现多态的内存管理，可以有效地跨多种设备类型使用。

```
// Create device copy of input collections
traccc::cell_collection_types::buffer cells_buffer(
    cells_per_event.size(), mr.main);
copy(vecmem::get_data(cells_per_event), cells_buffer);
traccc::cell_module_collection_types::buffer modules_buffer(
    modules_per_event.size(), mr.main);
copy(vecmem::get_data(modules_per_event), modules_buffer);
```

← 使用vecmem将cells数据拷入显存

多态内存资源(PMR)

Vecmem所使用的std::pmr提供了一种相当容易使用的方法来支持预定义的和用户定义的内存分配方式。

```
std::pmr::monotonic_buffer_resource pool{buf.data(), buf.size()};
std::pmr::vector<std::string> coll{&pool};
```

不同于std::vector，std::pmr::vector允许传入自定义内存资源，这一特性使得它能够能加轻松地使用异构资源。

我们正在尝试修改PODIO中*Collectiondata的储存方式，希望能将其从std::vector转化为std::pmr::vector。使其能直接在异构设备中运行，从而避免在数据转换过程中出现的大量数据拷贝。

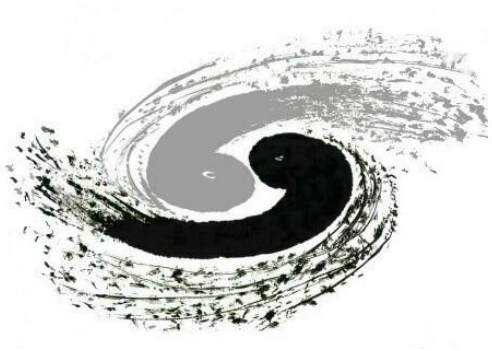
Work in progress: https://github.com/AIDASoft/podio/pull/424

总结和展望

- 开发支持异构计算是高性能数据处理软件的一个重要的研究方向
  - oneAPI 提供开放、统一的编程语言 DPC++
  - 以ACTS的TRACCC项目为例，介绍了异构计算在径迹重建中的代码实例
  - 进一步的工作：

    基于ACTS开发CEPC实验的Pixel探测器 Seeding 算法，结合外部径迹室进行联合径迹重建

- 高能物理实验数据量大，应当尽量避免数据拷贝，提升异构计算的整体运行效率
  - TRACCC近期的工作：使用异步拷贝的方式，提升了整体径迹重建效率
  - std::pmr提供了支持预定义的和用户定义的内存分配方式
  - WIP：尝试修改PODIO中*Collectiondata的储存方式，使其能直接在异构设备中运行



谢谢！

李卫东、林韬、张逸舟

zhangyz@ihep.ac.cn

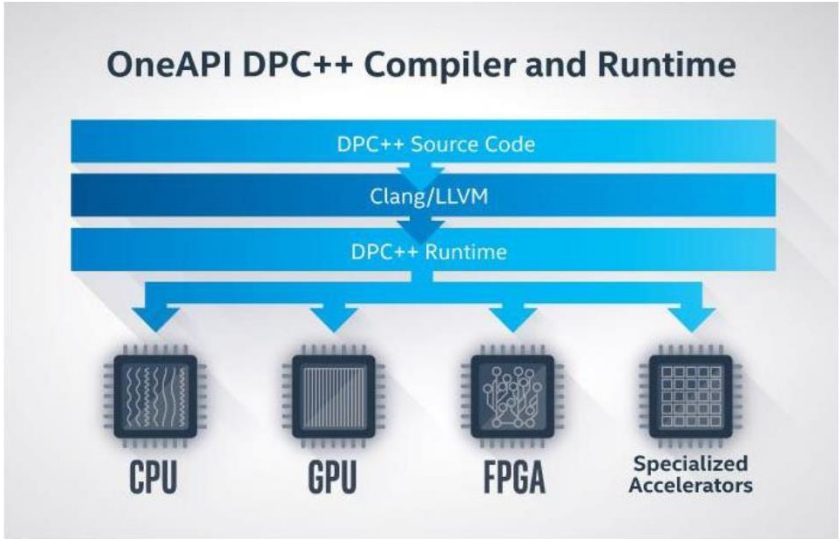
2023年粒子物理实验计算软件与技术研讨会
山东大学青岛校区

Jun. 11 2023

Backup

➢ DPC++(Data Parallel C++)

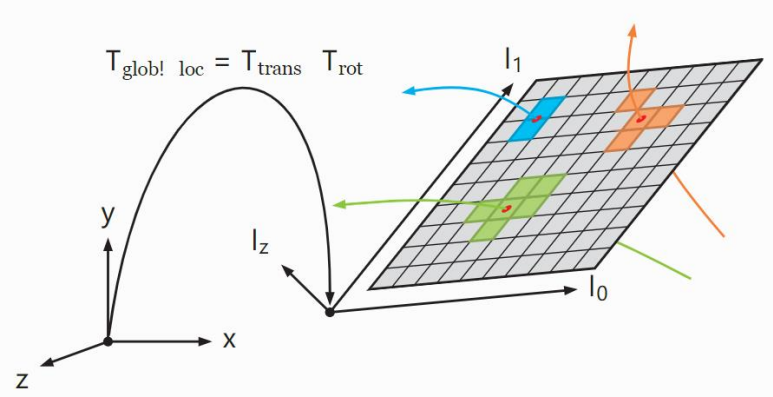
- DPC++是Intel为了将SYCL引入oneAPI所开发的开源项目。
- SYCL是一种 C++ 编程模型，是为了提高各种加速设备上的编程效率而开发的一种高级别的编程模型。简单来说它是一种跨平台的抽象层，用户不需要关心底层的加速器具体是什么，按照标准编写统一的代码就可以在各种平台上运行。SYCL大大提高了编写异构计算代码的可移植性和编程效率，已经成为了异构计算的行业标准。



2

异构计算在径迹重建中的应用

Application of heterogeneous computing in track reconstruction



聚类：TRACCC并行代码

```
// Run ccl kernel
details::get_queue(m_queue)
    .submit([&](::sycl::handler& h) {
        ::sycl::local_accessor<unsigned int> shared_uint(3, h);
        ::sycl::local_accessor<index_t> shared_idx(
            2 * max_cells_per_partition, h);

        h.parallel_for<kernels::ccl_kernel>(
            cclKernelRange, [=](::sycl::nd_item<1> item) {
                index_t* f = &shared_idx[0];
                index_t* f_next = &shared_idx[max_cells_per_partition];
                unsigned int& partition_start = shared_uint[0];
                unsigned int& partition_end = shared_uint[1];
                unsigned int& outi = shared_uint[2];
                traccc::sycl::barrier barry_r(item);

                device::ccl_kernel(
                    item.get_local_linear_id(), item.get_local_range(0),
                    item.get_group_linear_id(), cells, modules,
                    max_cells_per_partition, target_cells_per_partition,
                    partition_start, partition_end, outi, f, f_next,
                    barry_r, measurements_view,
                    *aux_num_measurements_device, cell_links_view);
            });
    })
    .wait_and_throw();
```