人工智能软件现状与挑战

赵鹏，英特尔高级软件架构师

人工智能软件的发展

经典的软件开发流程

问题，需求

算法，模型

代码实现

非常成熟

编译

硬件

CPU

VERY HEAVY FLOW

集成到现有软件

算法验证

利用*GPU*编程语言
重写代码

新编译器，优化方法

GPU

XPU

(CPU, GPUs, FPGA, AI Chips)

人工智能的软件发展模式

oneAPI

oneAPI：跨架构的业界标准

intel

多架构下的

软件挑战

自顶向下

- 专用工作负载日益增多
- 需要各种不同的计算模式

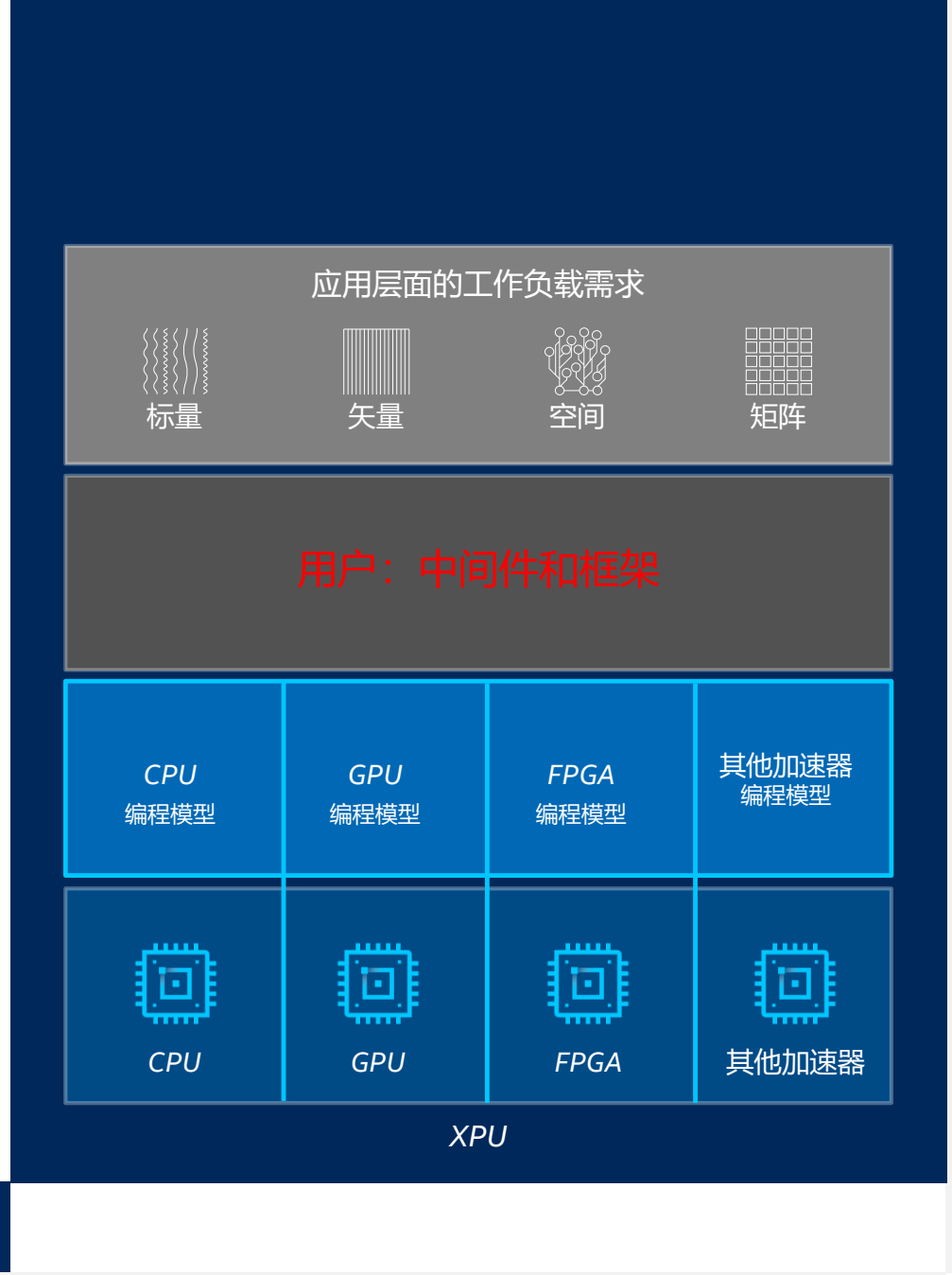
自低向上

- 每种架构都需要单独的编程模型和工具链

大量的工作集中在了中间层

导致软件维护扩展问题

限制了算法和架构选择的自由



英特尔® *oneAPI*

一种用于多种架构和厂商的编程模型

实现所有硬件价值

- 跨 *CPU*、*GPU*、*FPGA* 和其他加速器的性能

放心地开发和部署软件

- 开放的行业标准为未来提供了一条安全、清晰的道路
- 与现有的语言和编程模型兼容，包括 *C++*、*Python*、*SYCL*、*OpenMP*、*Fortran* 和 *MPI*

解耦中间层与底层硬件以及编程模型的绑定

- 开发更倾向于以用户需求为目标
- 选择软件无法决定的最佳加速技术

英特尔® *oneAPI*

oneAPI是一套包括：

完整高级编译器、库以及移植、分析和调试器工具的集合

- 利用最先进的硬件功能加速计算
- 可与现有的编程模型和代码库 (*C++*、*Fortran*、*Python*、*OpenMP* 等）互操作，开发人员可确信现有应用能够与 *oneAPI* 无缝协作
- 使用单一代码库即可轻松过渡到新系统和加速器，使开发人员有更多时间投入创新

现已上市

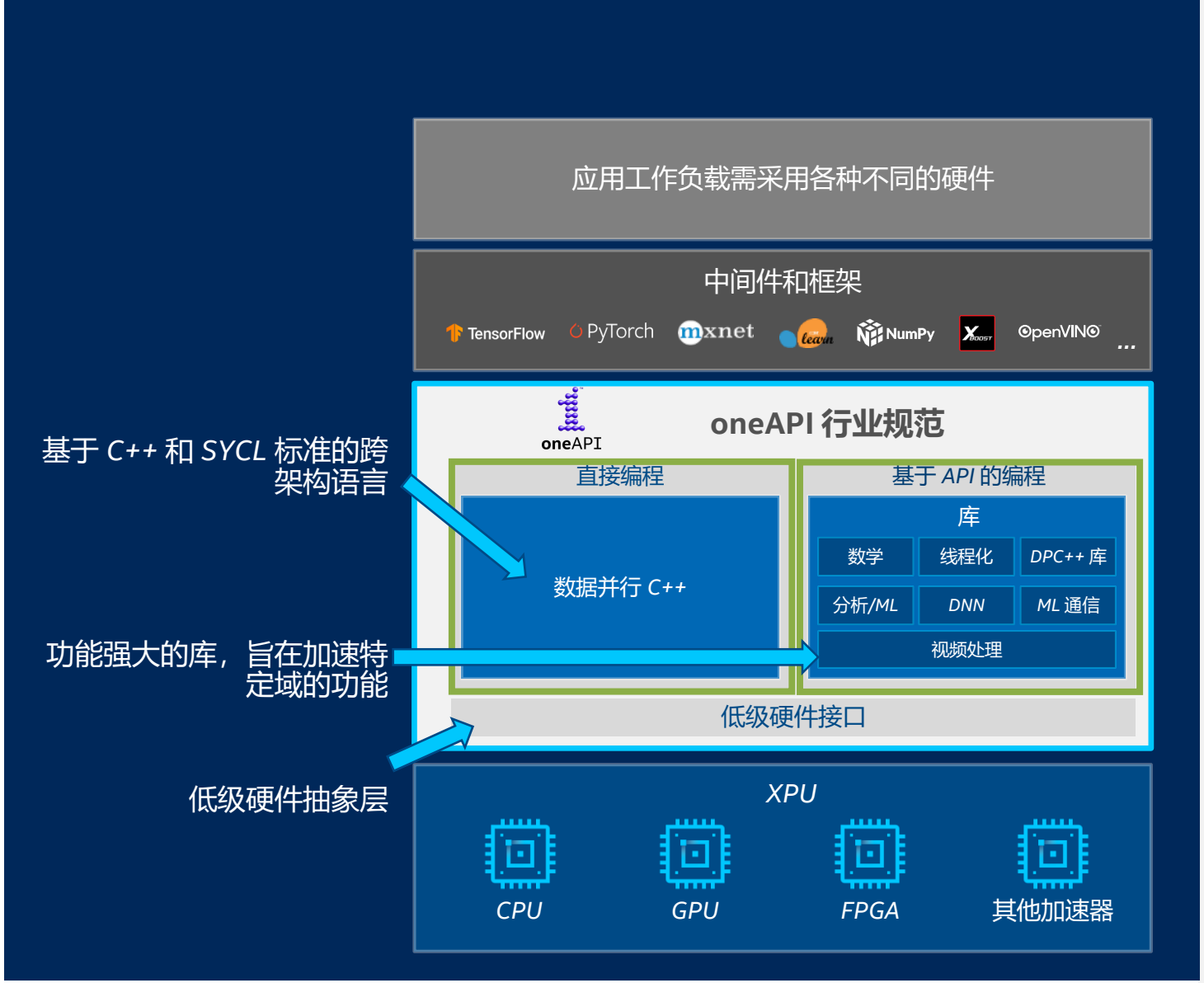
更多详情请访问 *software.intel.com/oneapi*
部分功能可能因架构不同而有所区别，需要进行自定义调试。未来将支持其他加速器。

oneAPI 行业计划

突破限制，避免厂商锁定

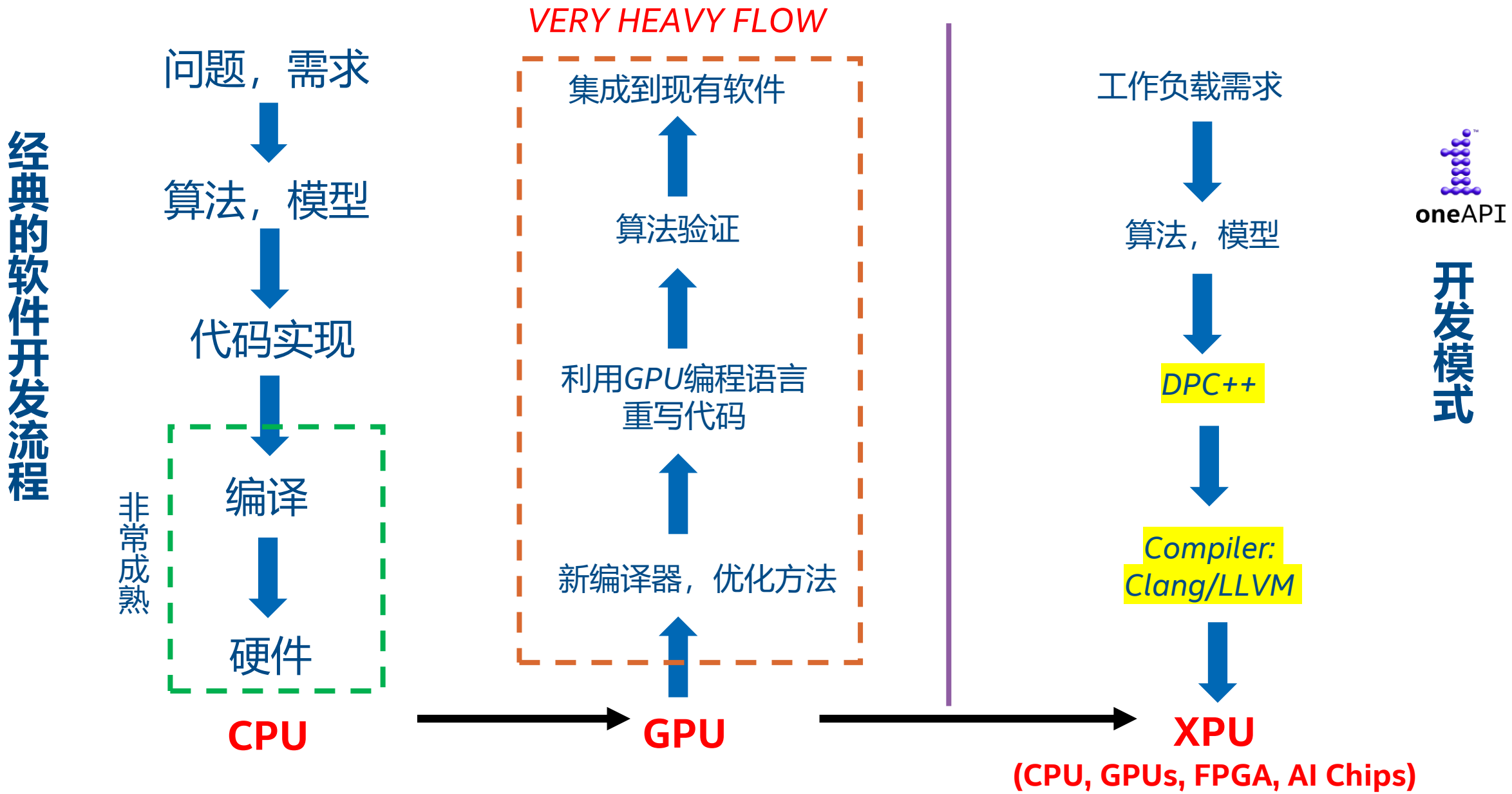
通过开放促进社区和行业协作

支持针对不同的架构和供应商重复使用代码



oneAPI

高效、智能且自由地实现加速计算，摆脱专有编程模型的经济和技术负担


经典的软件开发流程
问题，需求
算法，模型
代码实现
编译
硬件
非常成熟
CPU
VERY HEAVY FLOW
集成到现有软件
算法验证
利用GPU编程语言
重写代码
新编译器，优化方法
GPU
工作负载需求
算法，模型
DPC++
Compiler:
Clang/LLVM
XPU
(CPU, GPUs, FPGA, AI Chips)
oneAPI
开发模式

oneAPI对异构硬件的支持

Freedom of Choice in Hardware Drives Productivity

Codeplay contribution to DPC++ brings SYCL support for NVIDIA GPUs

oneAPI oneDNN on Arm for A64FX Fugaku

Extending DPC++ with Support for Huawei AI Chipset

ARGONNE, ORNL AWARD CODEPLAY CONTRACT TO STRENGTHEN SYCL SUPPORT FOR AMD GPUS

European exascale combines SiPearl's CPU RHEA with Intel's Xe GPU PVC

NERSC, ALCF, CODEPLAY PARTNER ON SYCL FOR NEXT-GENERATION SUPERCOMPUTERS

on Nvidia

*oneAPI*对人工智能生态系统支持

随着 AI、机器学习和以数据为中心的应用的增长，行业需要一种编程模型，以允许开发人员充分利用处理器架构的快速创新。TensorFlow 支持 oneAPI 行业计划及其基于标准的开放规范。

*oneAPI 补充了 TensorFlow 的模块化设计，并提供了**更多的**硬件厂商和处理器架构**选择**，以及**对下一代加速器的更快支持**。如今，TensorFlow 在至强处理器上使用 oneAPI，我们期望在未来的英特尔架构上使用 oneAPI。*

*"...oneAPI 的目标是提供更多的硬件厂商选择、处理器架构以及对下一代加速器的**更快支持**。*

*作为其计划的一部分，Microsoft 一直在英特尔硬件产品中使用 oneAPI 元素，并支持**基于开放标准的规范**。*

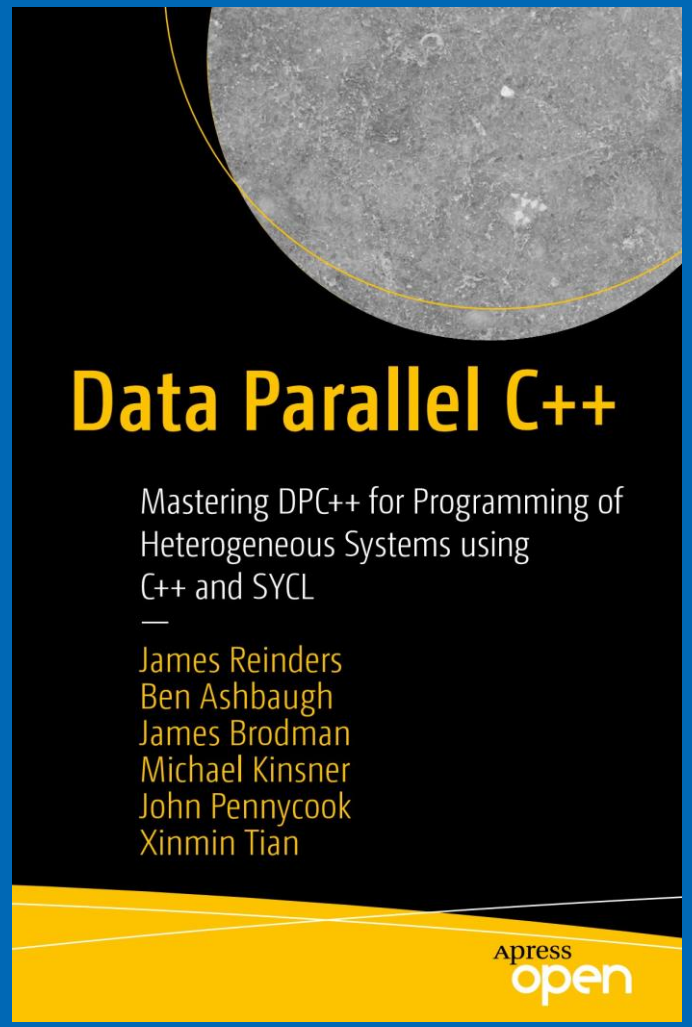
*我们很高兴能够为客户提供丰富的**选择**，并**加速 AI 和机器学习的发展**。"*

这些组织支持 *oneAPI* 计划提出的统一的编程模型"概念"，以便顺利地进行跨架构开发。它并不表示任何购买或使用英特尔产品的协议。
* 其他的名称和品牌可能是其他所有者的资产。

oneAPI Ecosystem Support

These organizations support the oneAPI initiative 'concept' for a single, unified programming model for cross-architecture development.
*It does not indicate any agreement to purchase or use of Intel's products. *Other names and brands may be claimed as the property of others.*

oneAPI异构编程模式

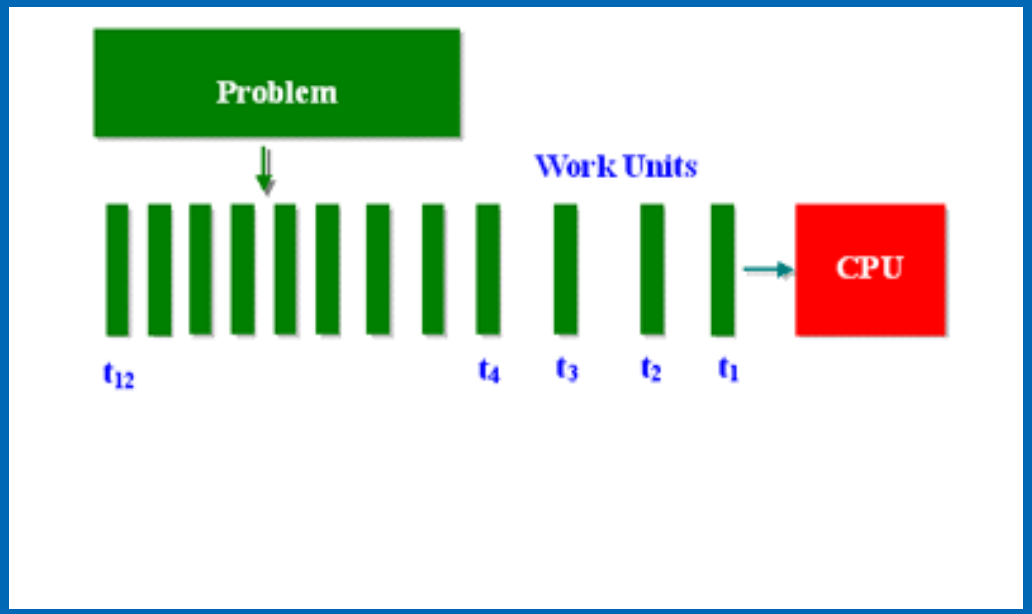


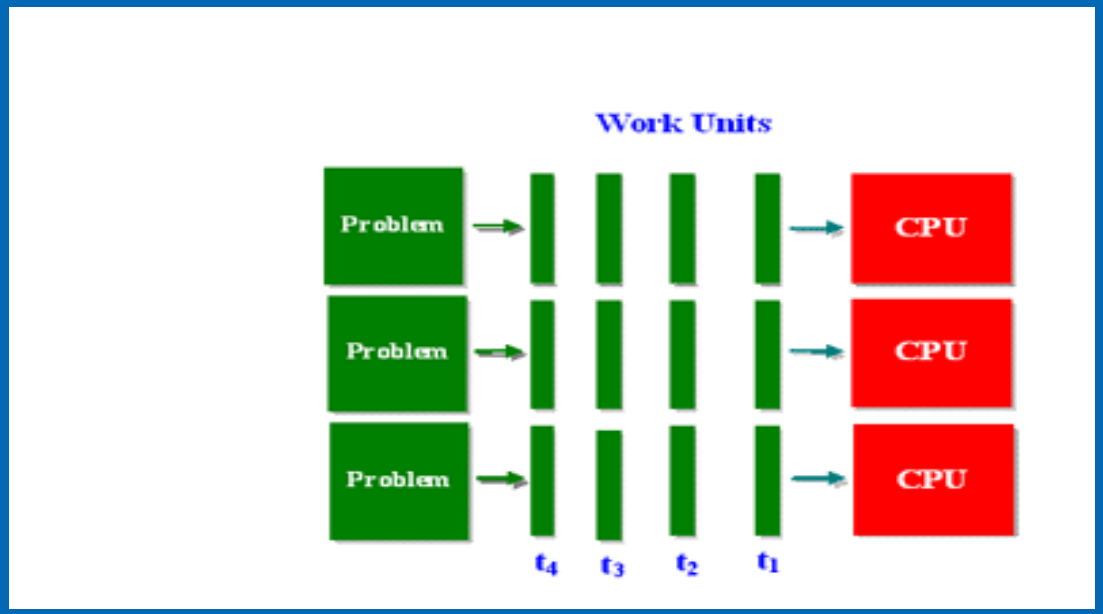
我们为什么要学一门新的编程语言？

异构编程与并行计算

异构编程：在不同的体系机构上，将不同的任务分配给不同的设备处理

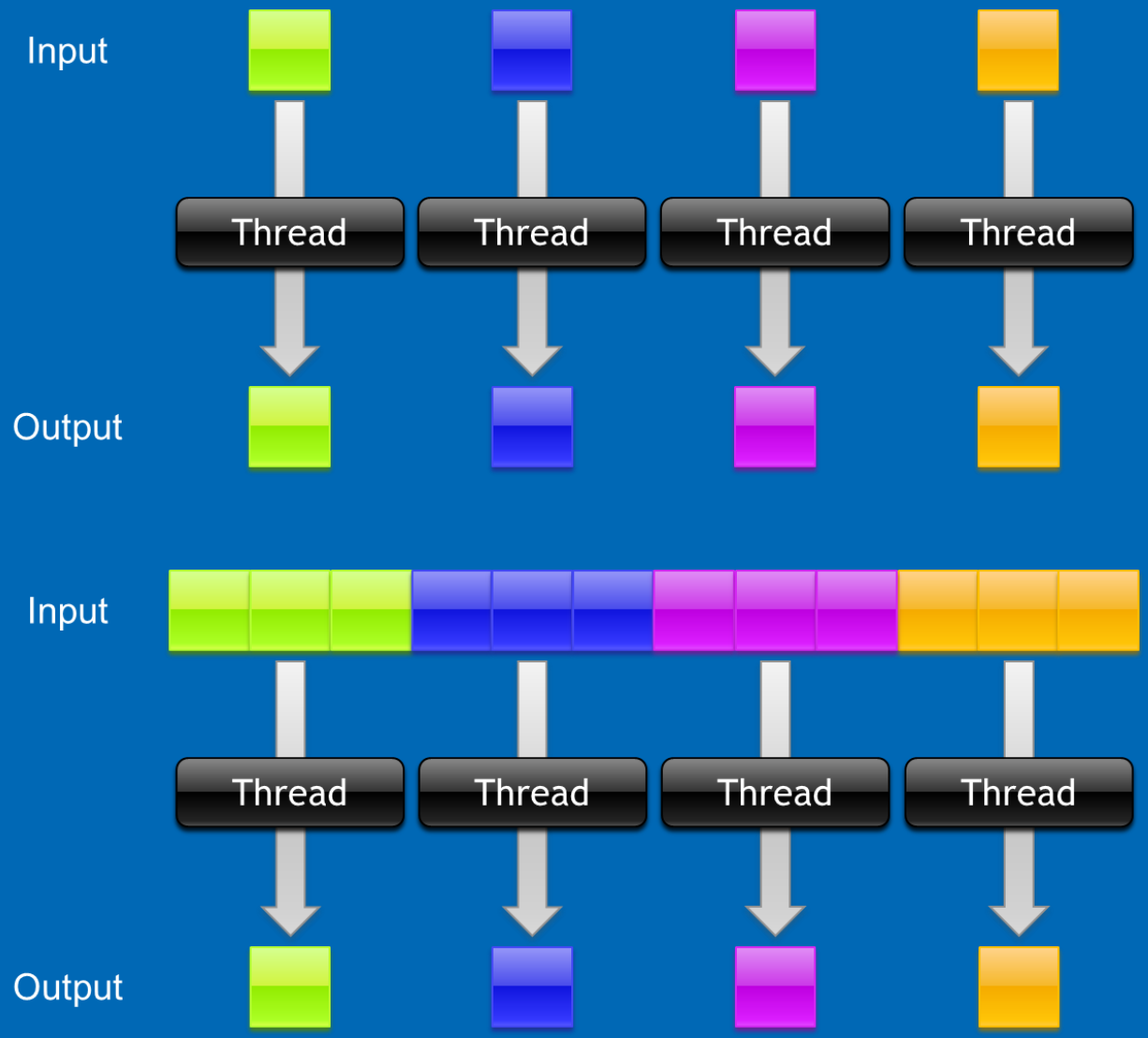
并行计算：将任务划分为相互独立的部分，然后交给更多的处理器来计算



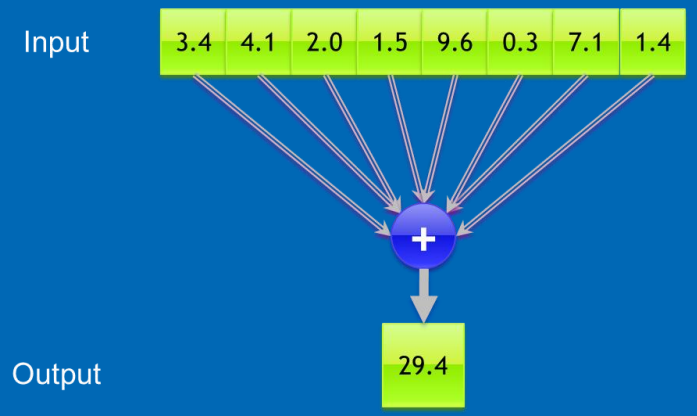


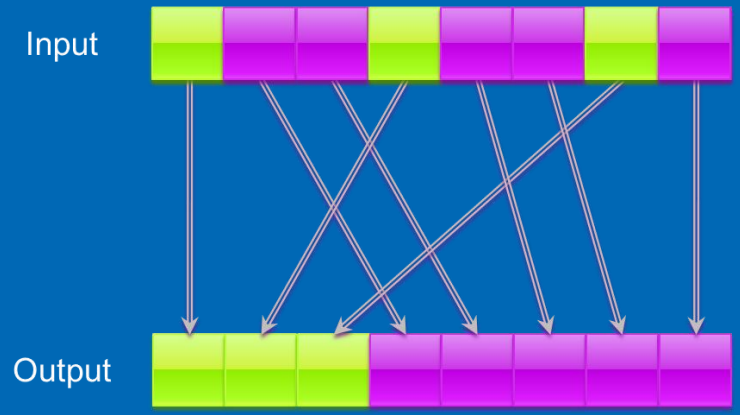
并行算法设计的两个关键点

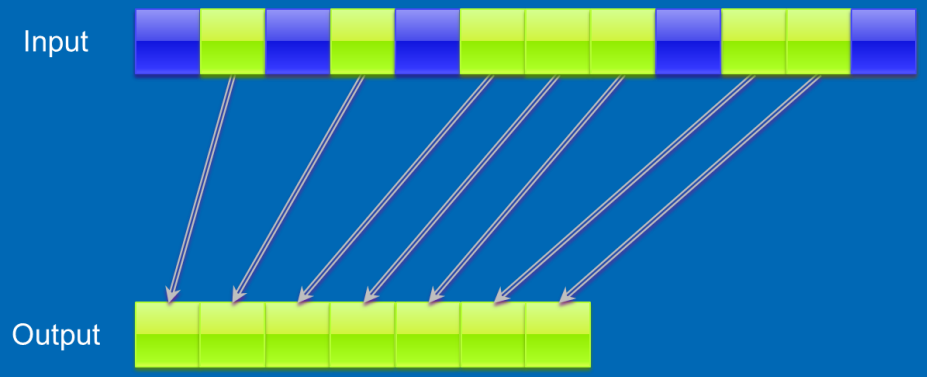
- 任务划分
- 数据访问

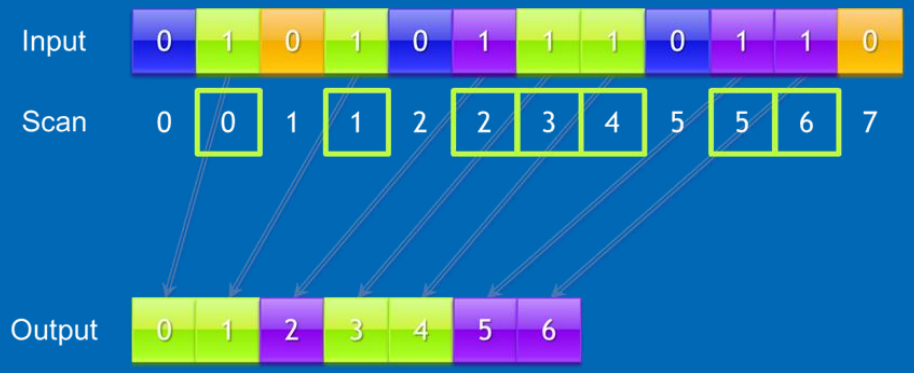


更多更复杂的模式


Input
3.4
4.1
2.0
1.5
9.6
0.3
7.1
1.4
+
Output
29.4


Input
Output


Input
Output


Input
0 1 0 1 0 1 1 1 0 1 1 0
Scan
0 0 1 1 2 2 3 4 5 5 6 7
Output
0 1 2 3 4 5 6

Data Parallel C++ 是什么?

基于C++语言的扩展，它提供了：

- **硬件设备的抽象**
- **数据访问的方法**
- **并行性的表达**

设备和队列

- **设备,** 表示 *OneAPI* 系统中的各种硬件

  设备类是预定义的设备选择和查询的方法

  包含用于查询设备信息的成员函数，

  支持创建不同硬件， *CPU/GPU/FPGA/...*

- **队列，** 是一种将工作提交到设备的机制

主程序将任务推入队列，异构设备从队列中获得执行任务

一个队列映射到一个设备，多个队列可以映射到同一设备。

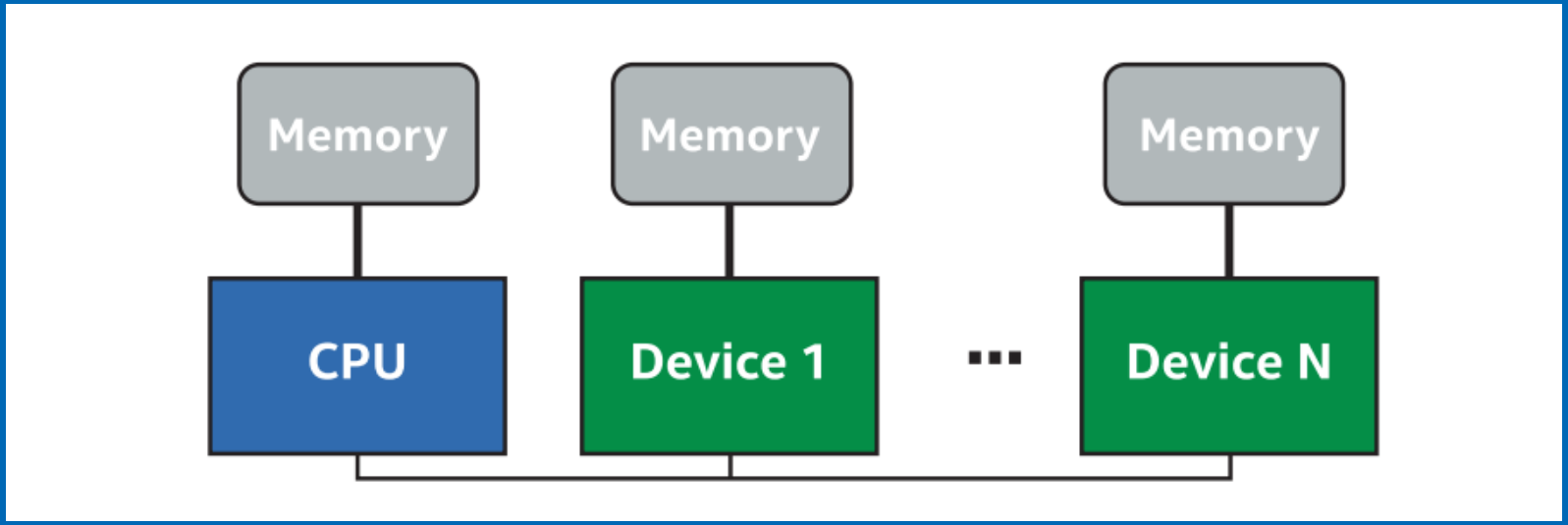
Code Example

```
#include <CL/sycl.hpp>
#include <iostream>
using namespace sycl;
int main() {
  queue my_gpu_queue( gpu_selector{} );
}
```

```
patric@gpu:~$ dpcpp gpu_selector.cpp
patric@gpu:~$ ./a.out
Selected GPU device: Intel(R) Iris(R) Xe MAX Graphics [0x4905]
```

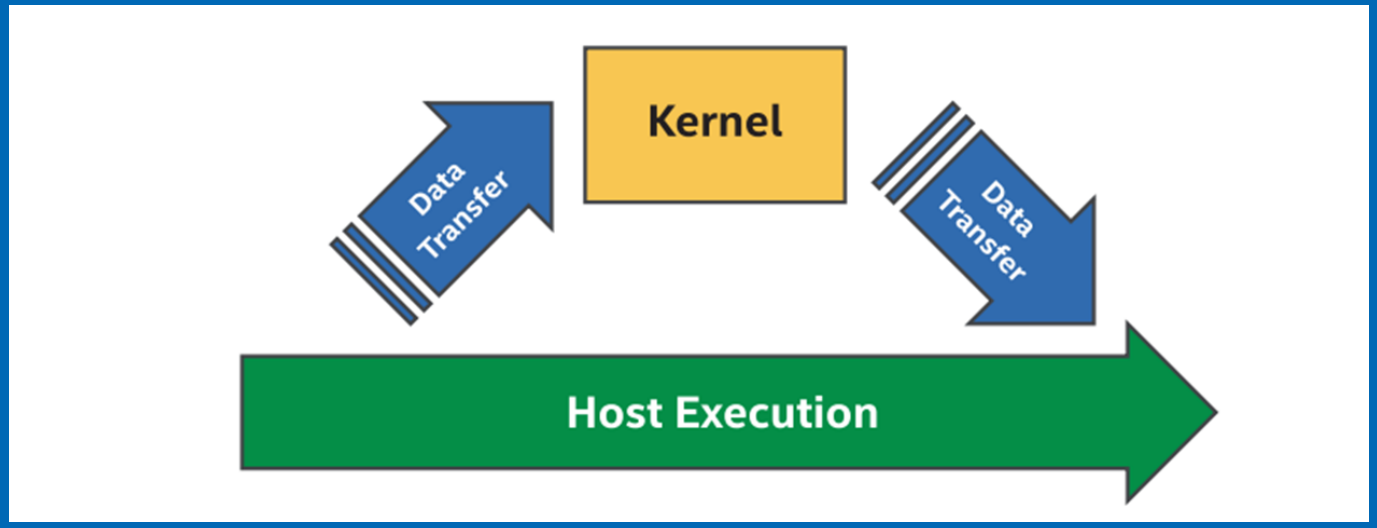
数据移动

- *CPU*和*GPU*具有独立的存储空间



程序员需要考虑如何在不同设备之间移动数据

- 显式的数据移动，*CPU->GPU*，计算， *GPU -> CPU*
- 隐式的数据移动， 数据在被访问的时候，系统自动进行数据迁移
- 不同的数据移动方式会影响性能



- # 显式内存申请

Traditional C/C++

Allocate memory block

Allocates a block of size bytes of memory, returning a pointer to the beginning of the block.

Data Parallel C++

- # 显式内存拷贝

Traditional C/C++

Copy block of memory

Copies the values of num bytes from the location pointed to by source directly to the memory block pointed to by destination.

Data Parallel C++

- 内存释放

Traditional C/C++

Deallocate memory block

A block of memory previously allocated by a call to , or is deallocated, making it available again for further allocations.

Data Parallel C++

Free memory allocated by , ,
or .

代码实例

```
constexpr int N = 10;

  int *host_mem   = malloc_host<int>(N, my_gpu_queue);
  int *device_mem = malloc_device<int>(N, my_gpu_queue);

  // Init CPU data
  for(int i = 0; i < N; i++) {    host_mem[i] = i;    }

  // Copy from host(CPU) to device(GPU)
  my_gpu_queue.memcpy(device_mem, host_mem, N * sizeof(int)).wait();

  // do some works on GPU

  // Copy back from GPU to CPU
  my_gpu_queue.memcpy(host_mem, device_mem, N * sizeof(int)).wait();

  printf("\nData Result\n");
  for(int i = 0; i < N; i++) {   printf("%d, ", host_mem[i]);  }
```

https://github.com/pengzhao-intel/oneAPI_course/blob/main/code/data_movement_ex.cpp

代码

```
constexpr int N = 10;

  int *host_mem   = malloc_host<int>(N, my_gpu_queue);
  int *device_mem = malloc_device<int>(N, my_gpu_queue);
```

编译：

```
patric@gpu:~/course$ dpcpp data_movement_ex.cpp -o data_move
```

运行输出：

```
patric@gpu:~/course$ ./data_move
Selected GPU device: Intel(R) Iris(R) Xe MAX Graphics [0x4905]

Data Result
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
Task Done!
```

https://github.com/pengzhao-intel/oneAPI_course/blob/main/code/data_movement_ex.cpp

内核

- 什么地方需要并行化?

计算量最大，最耗时的地方，通常是循环内的部分

串行的执行方式

```
for(int i=0; i < 1024; i++){
   a[i] = b[i] + c[i];
});
```

并行的执行方式

```
launch N kernel instances {
    int id =
get_instance_id();
    c[id] = a[id] + b[id];
}
```

parallel_for

- 并行化 *for*循环是并行计算的核心

  在该循环中，每个迭代应该是完全独立的，并且不分顺序。
- 并行内核使用 *parallel_for* 函数表示

串行的执行方式

```
for(int i=0; i < 1024; i++){
   a[i] = b[i] + c[i];
});
```

使用 **parallel_for** 卸载到加速器

```
h.parallel_for(range<1>(1024), [=](id<1> i){
   a[i] =  b[i] + c[i];
});
```

基础并行内核

基本并行内核的功能通过 *range*、*id* 和 *item* 类提供。

- *range* 用于描述任务空间的大小
- *item* 代表内核函数的单个实例，向执行范围的查询属性公开其他函数
- 利用*item*的信息来将每个线程对应到整体的任务空间

```
h.parallel_for(range<1>(1024), [=](item<1> item){
                auto idx = item.get_id();
                auto R = item.get_range();
                // CODE THAT RUNS ON DEVICE
});
```

代码实例

```
// Copy from host(CPU) to device(GPU)
my_gpu_queue.memcpy(device_mem, host_mem, N * sizeof(int)).wait();

// do some works on GPU
// submit the content to the queue for execution
my_gpu_queue.submit([&](handler& h) {

    // Parallel Computation
    h.parallel_for(range{N}, [=](id<1> item) {
        device_mem[item] *= 2;
    });

 }); // end submit

// wait the computation done
my_gpu_queue.wait();

// Copy back from GPU to CPU
my_gpu_queue.memcpy(host_mem, device_mem, N * sizeof(int)).wait();
```

https://github.com/pengzhao-intel/oneAPI_course/blob/main/code/basic_parafor.cpp

代码实例

```
// Copy from host(CPU) to device(GPU)
my_gpu_queue.memcpy(device_mem, host_mem, N * sizeof(int)).wait();

// do some works on GPU
// submit the content to the queue for execution
my_gpu_queue.submit([&](handler& h) {
```

运行输出：

```
patric@gpu:~/course$ ./basic_parafor
Selected GPU device: Intel(R) Iris(R) Xe MAX Graphics [0x4905]

Data Result
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
Task Done!
```

```
my_gpu_queue.memcpy(host_mem, device_mem, N * sizeof(int)).wait();
```

https://github.com/pengzhao-intel/oneAPI_course/blob/main/code/basic_parafor.cpp

oneAPI 可在英特尔®*DevCloud*演示

一个开发沙盒，支持您使用英特尔的 *oneAPI* 软件在所有英特尔 *CPU*、*CPU* 和 *FPGA* 上开发、测试和运行工作负载。

即刻正常运行！

software.intel.com/devcloud/oneapi

intel DevCloud

1 分钟编码

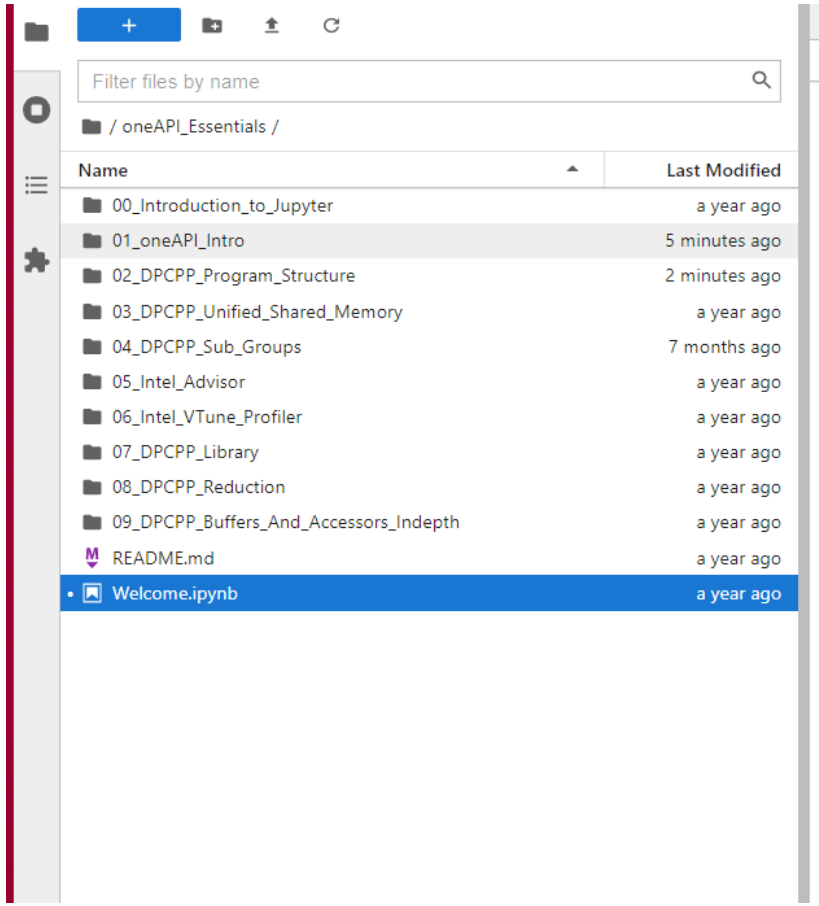
无需购买硬件

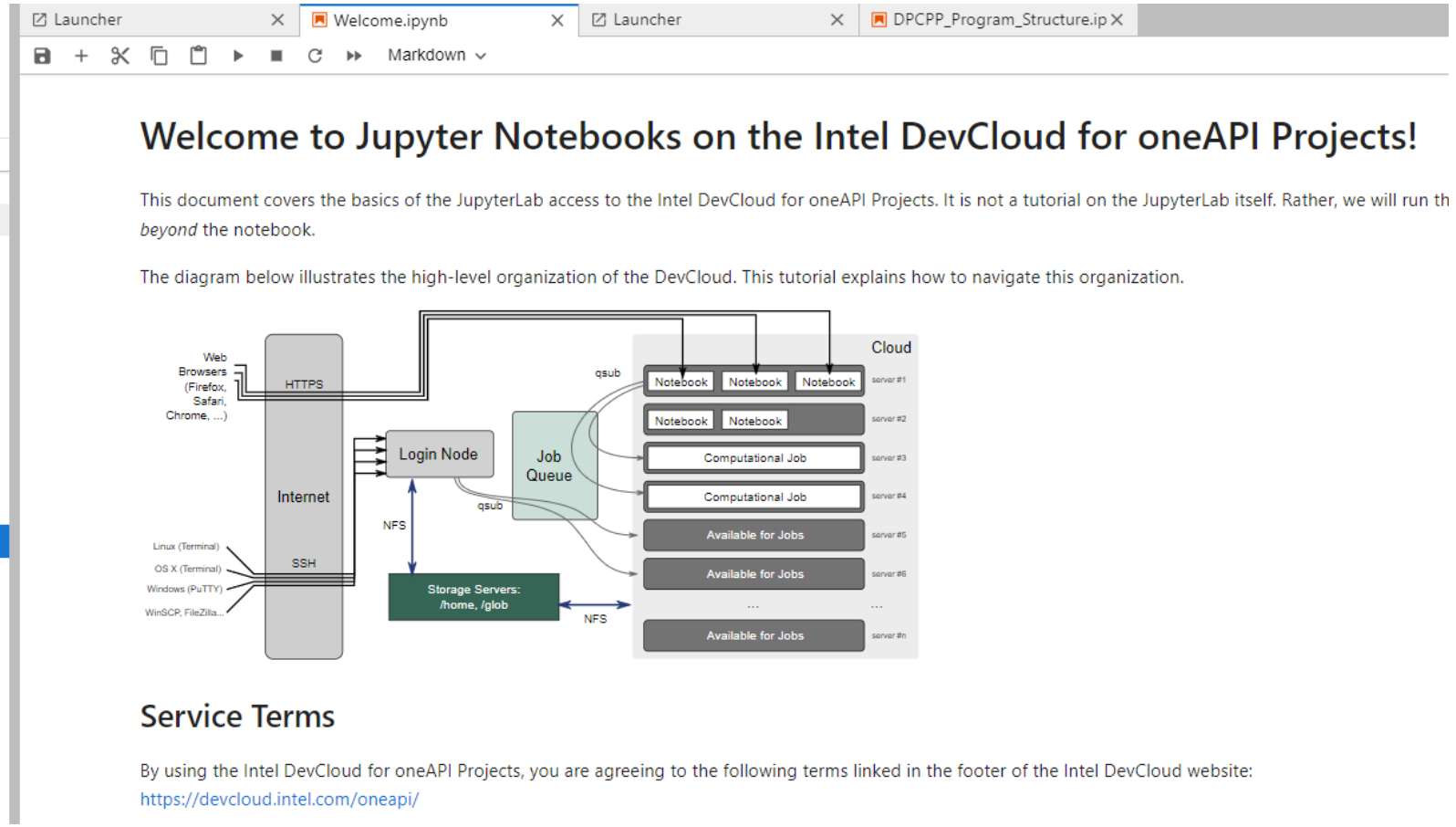
无需下载、安装或配置

轻松访问示例和教程

支持 *Jupyter Notebooks*，*Visual Studio* 代码

Full Course for You !





小结

- 理解异构计算的核心思想
- 能够实现简单的并行内核
- 具有系统和内核层性能分析的能力

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Results may vary.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details.
No product or component can be absolutely secure.

Slide 50 - **Texas Advanced Computing Center (TACC) Frontera references**
Article: HPCWire: Visualization & Filesystem Use Cases Show Value of Large Memory Fat Notes on Frontera.
www.intel.com/content/dam/support/us/en/documents/memory-and-storage/data-center-persistent-mem/Intel-Optane-DC-Persistent-Memory-Quick-Start-Guide.pdf
software.intel.com/content/www/us/en/develop/articles/introduction-to-programming-with-persistent-memory-from-intel.html
wreda.github.io/papers/assise-osdi20.pdf
KFBIO
KFBIO m. tuberculosis screening detectron2 model throughput performance on 2nd Intel® Xeon® Gold 6252 processor: NEW: Test 1 (single instance with PyTorch 1.6: Tested by Intel as of 5/22/2020. 2-socket 2nd Gen Intel® Xeon® Gold 6252 Processor, 24 cores, HT On, Turbo ON, Total Memory 192 GB (12 slots/16 GB/2666 MHz), BIOS: SSE5C620.86B.02.01.0008.031920191559 (ucode: 0x500002c), Ubuntu 18.04.4 LTS, kernel 5.3.0-51-generic, mitigated Test 2 (24 instances with PyTorch 1.6: Tested by Intel as of 5/22/2020. 2-socket 2nd Gen Intel Xeon Gold 6252 Processor, 24 cores, HT On, Turbo ON, Total Memory 192 GB (12 slots/16 GB/2666 MHz), BIOS: SSE5C620.86B.02.01.0008.031920191559 (ucode: 0x500002c), Ubuntu 18.04.4 LTS, kernel 5.3.0-51-generic, mitigated BASELINE: (single instance with PyTorch 1.4): Tested by Intel as of 5/22/2020. 2-socket 2nd Gen Intel Xeon Gold 6252 Processor, 24 cores, HT On, Turbo ON, Total Memory 192 GB (12 slots/16 GB/2666 MHz), BIOS: SSE5C620.86B.02.01.0008.031920191559 (ucode: 0x500002c), Ubuntu 18.04.4 LTS, kernel 5.3.0-51-generic, mitigated.

Tangent Studios
*Configurations for Render Times with Intel® Embree, testing conducted by Tangent Animation Labs. Render farm: 8x Intel® Core™ processors +hyperthread*2 + 128gig. In-office workstations: Intel® Xeon® processors HP blade c7000 chassis, with HP460 gen8 blades - 2x Intel Xeon E5-2650 V2, Eight Core 2.6GHz-128GB. Software: Blender 2.78 with custom build using Intel® Embree. For more information on Tangent's work with Embree, watch this video:*
www.youtube.com/watch?time_continue=251&v=_2Ia4h8q3xs&feature=emb_logo
Recreation of the performance numbers can be recreated using Agent327, Blender and Embree.

Chaos Group - Up to 90% Memory Reduction for Displacement
Testing conducted by Chaos Group with Intel® Embree 2020. Software Corona Renderer 5 with Intel Embree. Up to 90% memory reduction calculated using Corona Renderer 5 with regular displacement grids per triangle of 154 bytes versus Corona Renderer 5 with Intel Embree, which has a displacement capability grid of 12 bytes per grid triangle. (12/154 = 7.8% usage or >90% memory reduction.) Recreation of the performance numbers can be accomplished using Corona Renderer 5 and Embree. For more information, visit the Corona Renderer Blog: blog.corona-renderer.com/corona-renderer-5-for-3ds-max-released/

The Addams Family 2 - Gained a 10% to 20%—and sometimes 25%—efficiency in rendering, saving thousands of hours in rendering production time.
Testing Date: Results are based on data conducted by Cinesite 2020-21. 10% to up to 25% rendering efficiency/thousands of hours saved in rendering production time/15 hrs per frame per shot to 12-13 hrs.
Cinesite Configuration: 18-core Intel® Xeon® Scalable processors (W-2295) used in render farm, 2nd gen Intel Xeon processor-based workstations (W-2135 and -2195) used. Rendering tools: Gaffer, Arnold, along with optimizations by Intel® Open Image Denoise.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.