一个求解旅行商问题的遗传算法

**重点。**

小组完成Part C，在DevMesh提交

### 个人部分（讲清楚自己的工作。建议不少于10页）

### 团队项目（建议不少于40页）

#### 项目组成员

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 项目分工 | 贡献占比 |
|  |  | XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX | 40% |
|  |  |  | 30% |
|  |  |  | 30% |

#### 项目简介

|  |  |
| --- | --- |
| 项目名称 | XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX |
| 项目摘要或简介 |  |

#### 项目背景

**TSP问题**

TSP就是Travelling Salesman Problem的缩写，即旅行商问题。旅行商问题是一个在数学和计算机领域中比较著名的问题，是一个组合优化问题。这个问题要解决的是，如果有一个旅行商人要拜访n个城市，那么这个商人他需要选择所走的路径，这里有一个规定是每一个城市商人都只能拜访一次，且要求最终需要回到原先出发时的那一个城市。旅行商路径的选择，目的就是要求取出一条路径，路径是所有路径之中的最小值。TSP问题是一个NP难的问题。现在可以说是还没有发现一个有效的算法来对旅行商问题的大型实例进行精确的求解，因而只能够使用一些近似的算法来对旅行商问题进行求近似解。

**遗传算法**

达尔文的进化论和孟德尔的遗传学说是人类科学史上的重要学说。在人工智能领域也有学者根据这两个学说抽象出 了基于“进化”观点的学习理论，即进化计算（Evolutionary Computation)。进化计算是一类模拟生物进化、自然选择过 程与机制求解问题的自组织、自适应人工智能技术。遗传算法（Genetic Algorithm）就是进化计算的典型代表。

进化计算的核心思想认为,生物进化过程(从简单到复杂,从低级向高级)本身是一个自然的、并行的、稳健的优化过程。这一优化过程的目标是对环境的自适应性。生物种群通过“优胜劣汰”及遗传变异来达到进化(优化)的目的。根据生物进化和遗传理论,进化过程通过繁殖、变异、竞争和选择这4种基本形式实现。如果把待解决的问题理解为对某个目标函数的全局优化,则进化计算就是建立在模拟生物进化过程基础上的随机搜索优化技术。

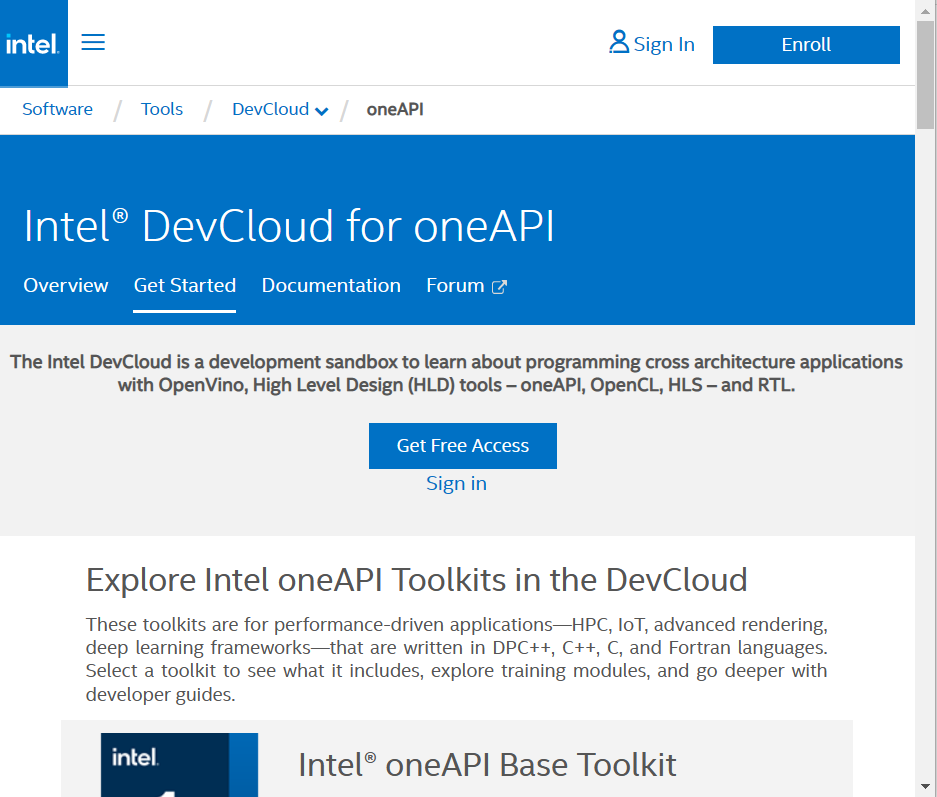
#### 项目内容和方案

本项目的内容是使用遗传算法来求解TSP问题，在实现这样一个遗传算法的过程当中，其部分的内容使用了oneAPI来进行并行的处理，从而使解决TSP问题的遗传算法变得更加的高效率。

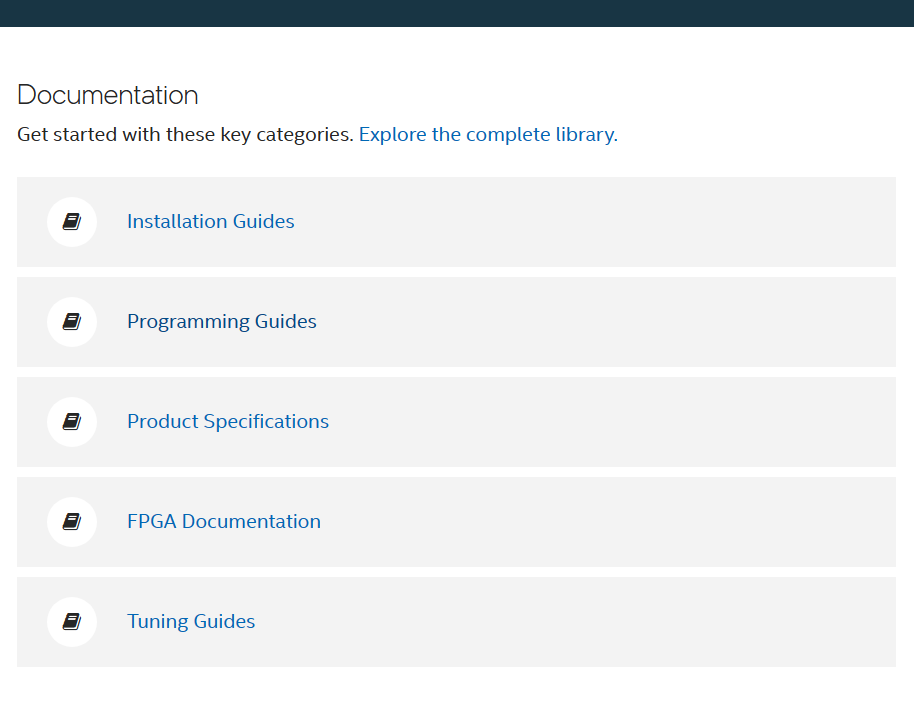
**一、开发环境的准备**

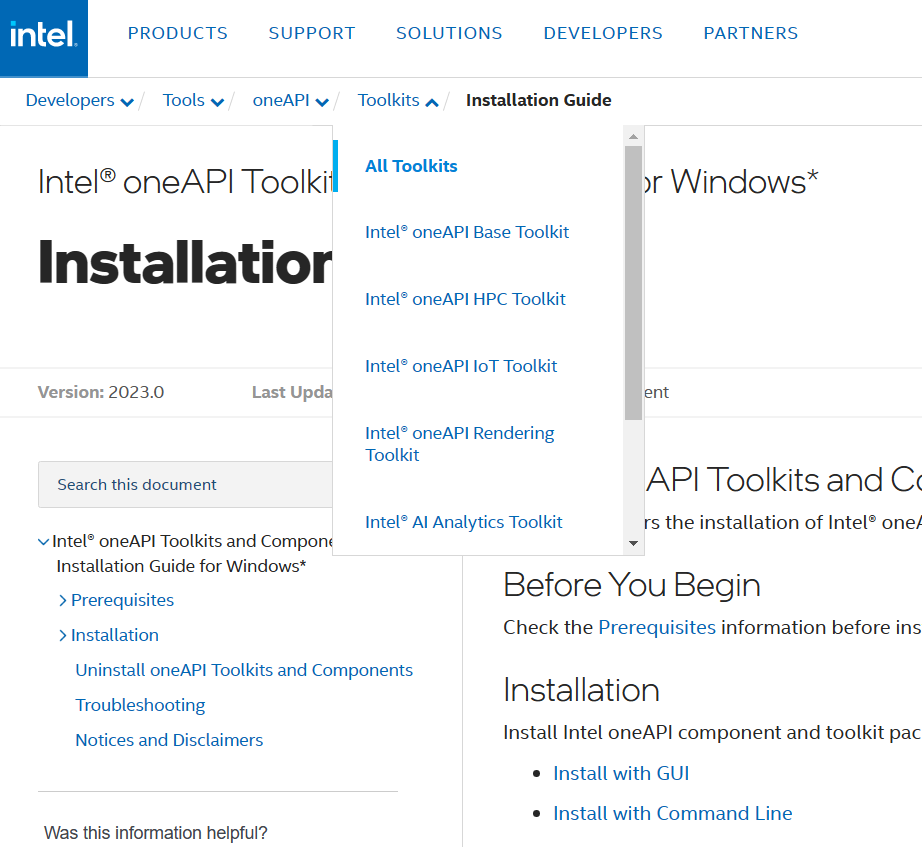
在开始之前，我们首先要安装好oneAPI的开发环境，从而能够在windows 10系统上进行oneAPI相关的开发。

首先我们打开oneAPI的官方网站，devcloud.intel.com/oneapi/get\_started

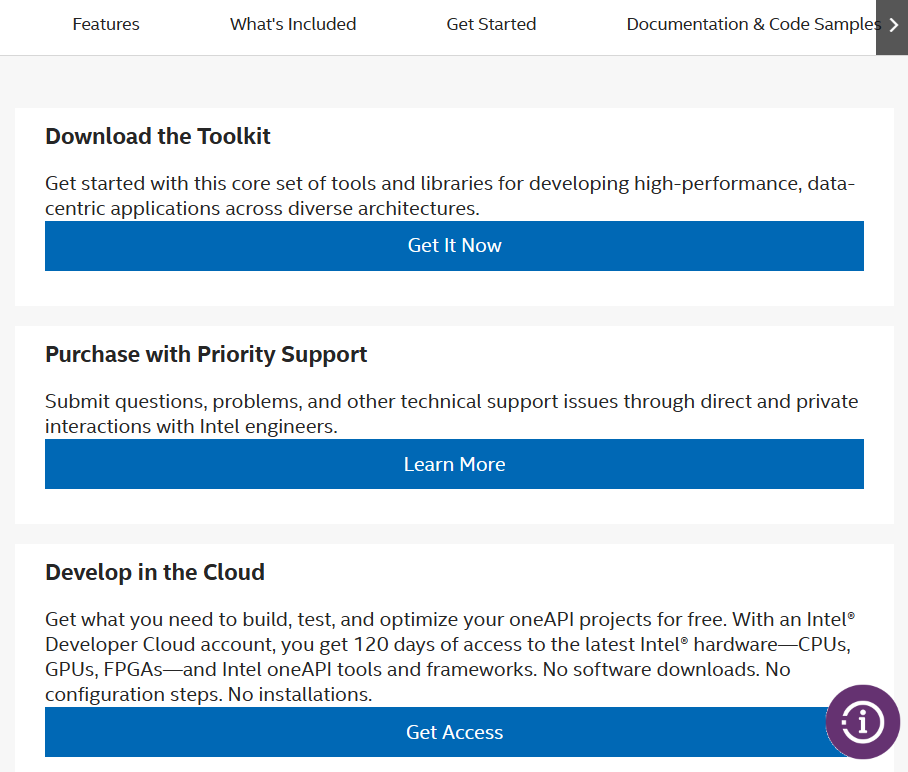


然后我们点击左上角的Software进入到Devzone，点击页面中的Installation Guides。

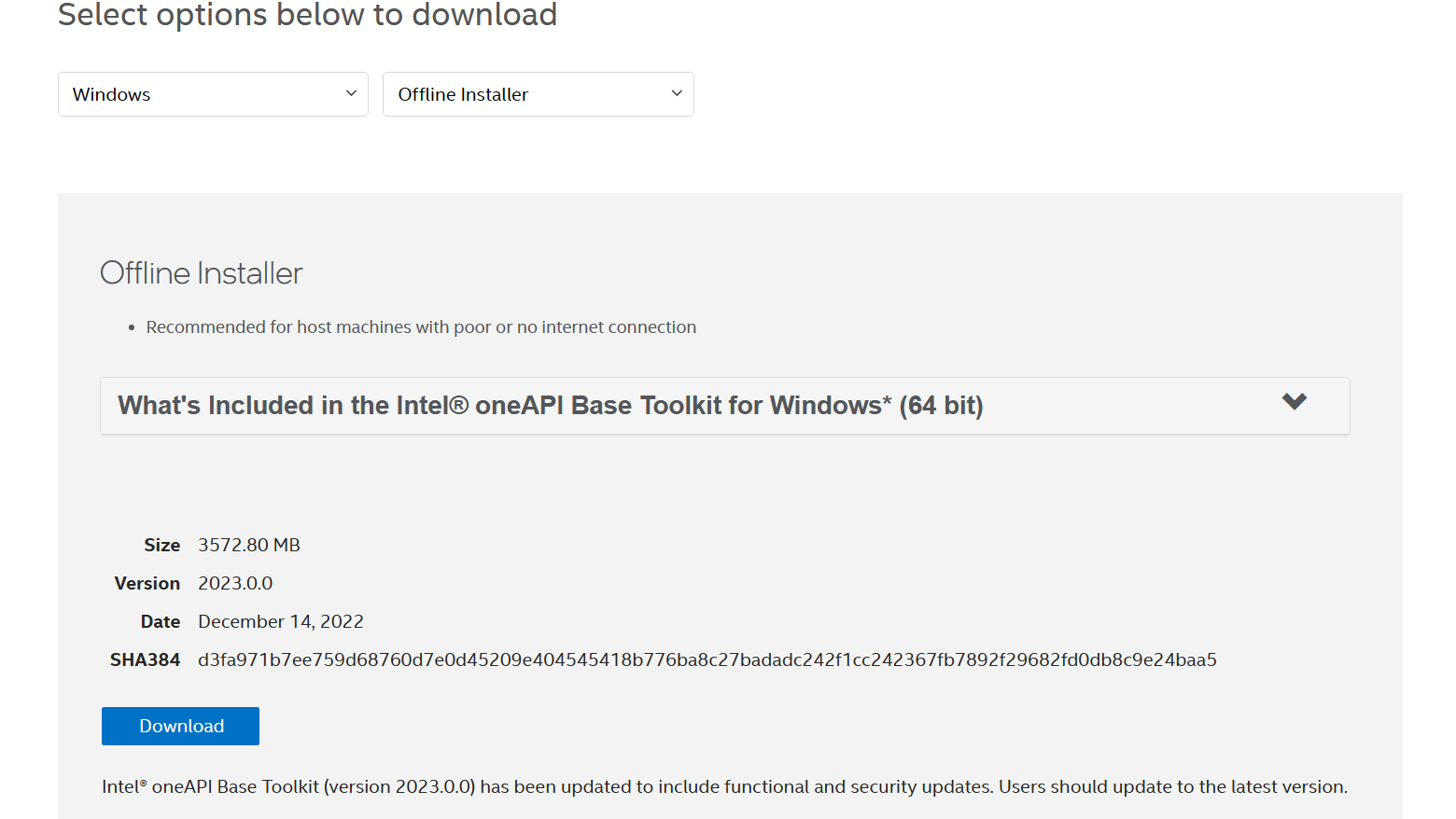


进入到Installation Guides页面之后，选择Intel® oneAPI Toolkits Installation Guide for Windows\*，查看在windows 10 系统下oneAPI Toolkit的安装引导。然后点击上方导航栏Toolkits下拉菜单，弹出一些Toolkit的选项，这里我们选择Intel oneAPI Base Toolkit。

进入到Intel oneAPI Base Toolkit的页面，在这里我们看到Download the Toolkit， 在此处点击Get It Now。



进入到下载页面，此处操作系统我们选择Windows，然后安装形式选择Offline Installer，因为选择了离线安装包便不用担心在安装过程中可能出现的网络问题。这之后点击Download按钮，下载Offline Installer。

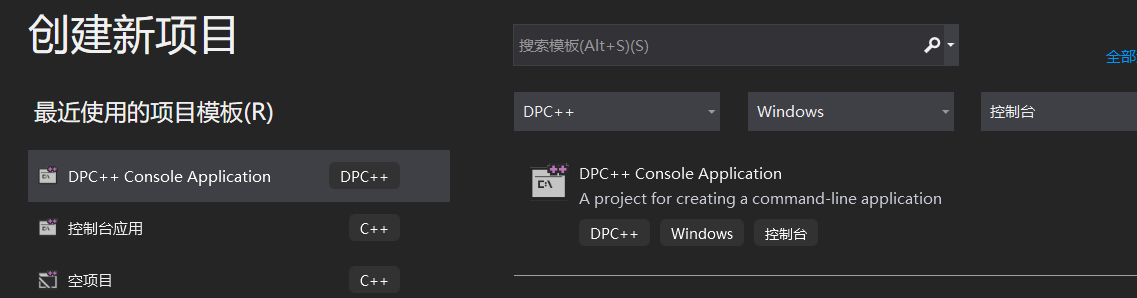


下载完成之后便会得到w\_BaseKit\_p\_2023.0.0.25940\_offline .exe文件。双击打开进行安装。



安装时只需要根据步骤一步步来就行，安装过程中记得要勾选关联visual studio的选项，这样我们才能够在visual studio当中使用 oneAPI Base Toolkit的内容，创建DPC++项目。

安装完成之后，打开visual studio， 可以看到，已经可以创建DPC++ 项目，看到这个选项说明 oneAPI Base Toolkit安装成功，并且已经成功与visual studio进行关联。

到这里为止，我们项目的环境准备就完成了。

**二、遗传算法的设计**

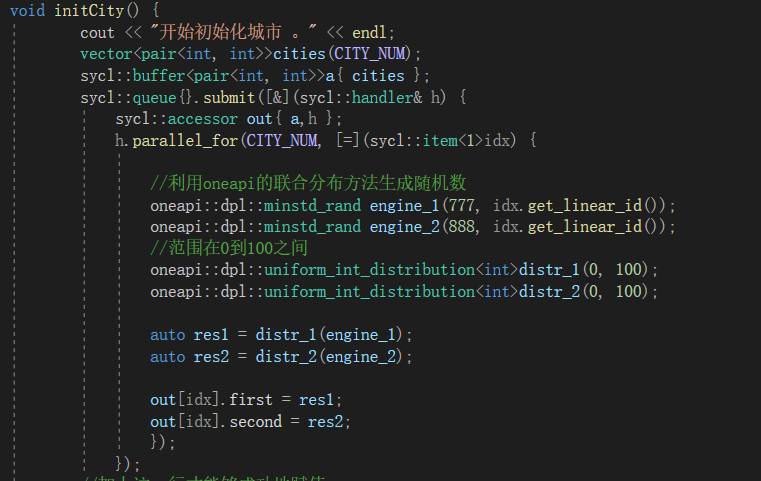
遗传算法主要需要设计几个部分，首先是随机生成城市的坐标，用以初始化旅行商问题。然后需要设计编码的策略，再者需要初始化种群，然后编写计算适应值的函数，设计选择算子。还需要设计交叉算子以及变异算子。



**1、城市坐标的生成**

城市的坐标使用C++的std::pair来表示，每个城市的坐标是一个std::pair<int,int>实例。每个城市的坐标都给它随机地来生成。把所有的城市坐标给放到一个vector中。

创建sycl::buffer, 将存放城市坐标的vector传入到buffer当中，接着使用sycl::queue，此队列连接到一个加速器(accessor)当中，当队列创建时，调用成员函数submit把工作提交到加速器(accessor)当中。使用lambda表达式，在其中提交并行处理的工作。调用sycl::handler的成员函数paraller\_for, 第一个参数表示sycl::buffer当中元素的个数，第二个参数表示buffer的下标。然后在parallel\_for中的lambda表达式中进行并行的操作。

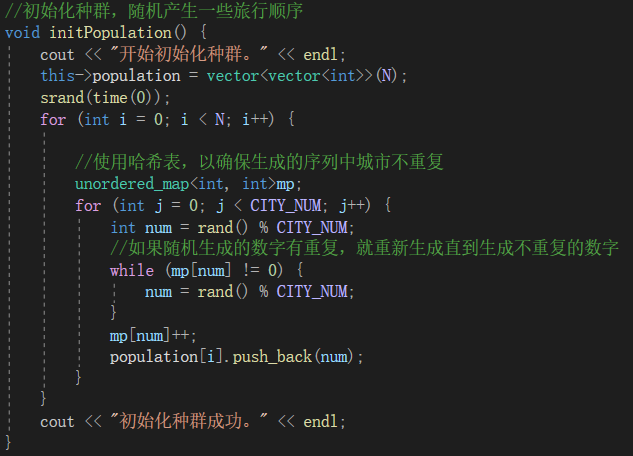
创建两个oneapi::dpl::minstd\_rand engine, 构造函数中第一个参数传入的是生成随机数的种子，第二个参数是偏移量。然后创建两个oneapi::dpl::uniform\_int\_distribution, 构造函数中的两个参数代表生成随机数的上界和下界。然后把生成的两个随机数给赋值到加速器(accessor)中的相应坐标的first和second当中。

**2、编码策略**

编码的策略是，此项目我们直接使用城市的序号来进行编码。编码的时候，每一个个体基因的个数等于城市的数量。对于每个基因都生成城市编号之内的随机数，由于可能生成重复的城市编号，因而此处需要进行去重的操作。去重的时候就是一直生成城市编号之内的随机数，直到这个随机数是一个未使用的城市编号，重复这个过程，直到一个个体每个基因的号码都是独一无二的，针对这一个个体的去重就完成了。

**3、种群初始化**

把种群存到一个大小为N(N代表种群数量)的vector里面。然后进入一个for循环，在内层的for循环里面初始化每一个个体的CITY\_NUM(城市数量)个基因。此外还有一个while循环来进行去重。这里面由于存在while循环，循环的次数是不确定的，**因为OpenMP并行化循环只能够处理事先知道迭代次数的for循环**。所以此处没有办法进行并行的优化。



**4、计算适应值（评估）**

编写一个成员函数用来计算评估值。由于我们的种群是一个二维的vector, 而sycl的buffer不支持使用2维的vector**（因为二维的std::vector并不是把元素给连续地存储在内存当中的）**，因而，如果要使用sycl进行并行计算，首先要先把存放种群的2维vector给转换成1维的vector，这个1维vector的大小为N\*CITY\_NUM。然后可以利用OpenMP并行地把2维vector中的值给赋值到新的1维vector里面去。之后，创建buffer，用展平的种群vector创建一个buffer，pop\_buf, 用存储评估值的vector创建一个buffer，eval\_buf, 用存放城市坐标的vector创建一个buffer，city\_buf。

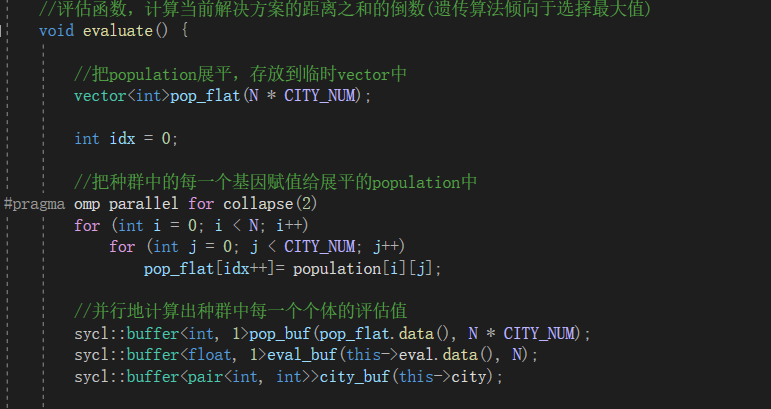
创建一个sycl队列用以连接到加速器，当队列创建时，调用成员函数submit把工作提交到处理器(handler)。再调用handler的成员函数parallel\_for,利用lambda表达式把工作给并行化。

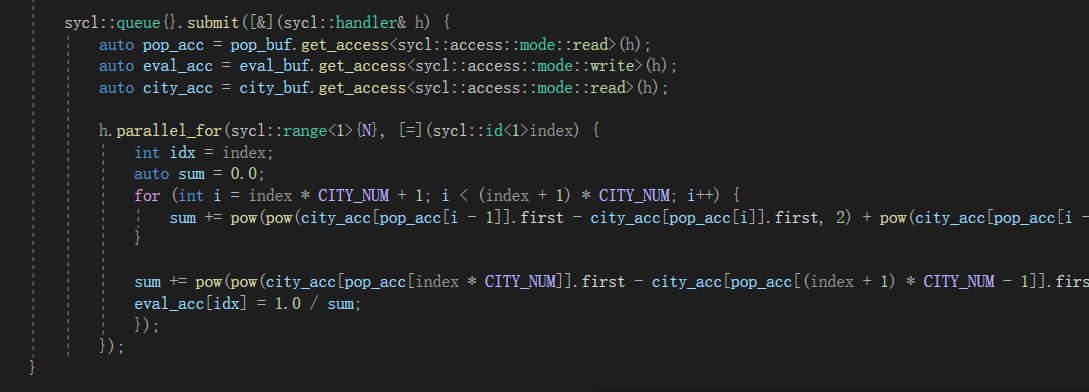
由于基因所对应的是城市的编号，记此编号为i，记城市数组为c,所对应的城市坐标就是 c(i)。要计算一条路径中的总花费，就是要计算个体DNA中的每一对相邻的基因所对应的城市的距离 计算出来，并将他们相加（最后一个基因所对应的城市要计算和第一个基因所对应城市的距离）。 计算评估值的公式如下，i表示第j个个体中的第i个基因。

**因为遗传算法倾向于选择值更大的解，然而我们要求的是最小的路线代价，所以，评估值应该为路线代价的倒数，这个倒数越大越好。**









**5、选择算子**

采用轮盘赌的选择方式。把每一个个体的适应值eval(j)加起来成为总的适应值。然后将每一个个体的适应值去除以总的适应值，得到被选择的概率p(j)。



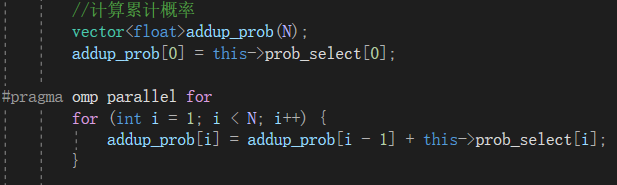
然后再对第j个个体，将每一个p(j)前面的所有概率累加起来得到P(j)。



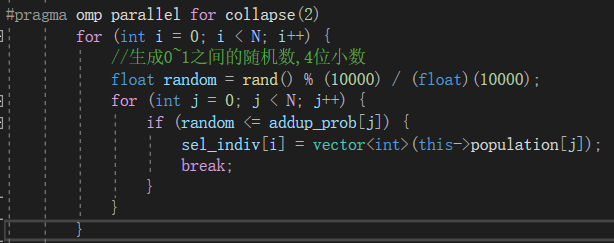
接下来生成随机数rand,如果rand大于P(j-1)而小于P(j)，那么就选择第j个个体。



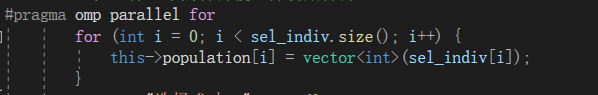
最终选择完成之后，将选择出来的个体替换掉初始的种群。此处对p(j)的计算使用了oneAPI进行并行改良。



首先生成0~1之间的随机数，比较此随机数的大小和P(j)的大小，如果恰好小于P(j)，则选择第j个个体。



所有个体选择完成之后，把选择出来的个体覆盖掉原始的种群。

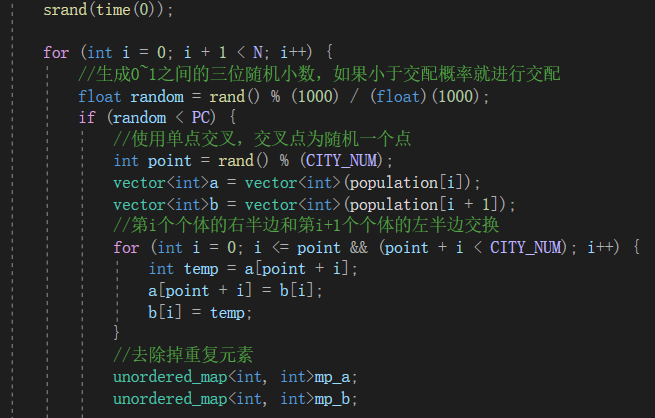


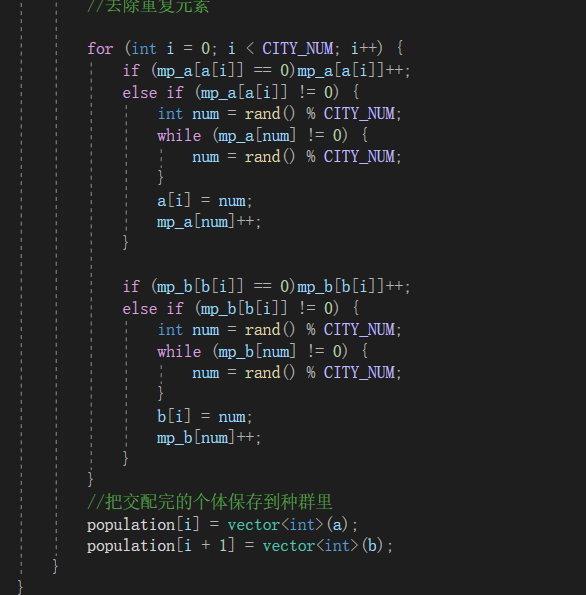
**6、交叉算子**

交叉的时候，生成0~1之间的三位随机小数，如果小于交配概率就可以进行交叉。

本项目的交叉方案使用单点交叉，交叉点是随机的一个点，相邻的两个个体交叉，记其中的一个个体是个体a，另一个个体是个体b，以交叉点为界，把个体a的右半边与个体b的左半边进行交换。之后分别对个体a和个体b进行去重。去重时随机地生成城市序号之内的随机数，如果生成的随机数还是已经被使用过了，那么就继续生成，直到生成没有被使用过的一个编号为止，如此反复直到交叉后的个体a、b的每一个基因都是没有重复的，那么去重就完成了。

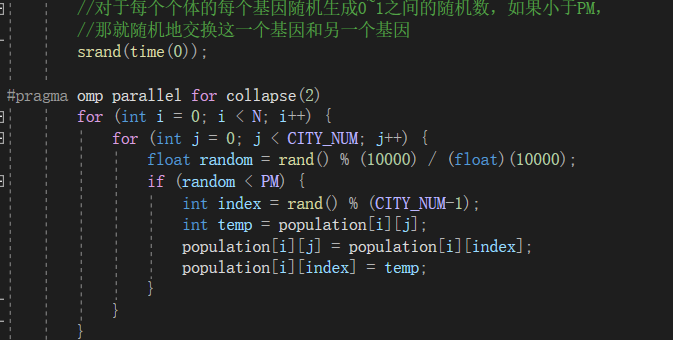
由于在去重的时候，循环的次数是不确定的，因此交叉的过程是没有办法使用并行化处理的。





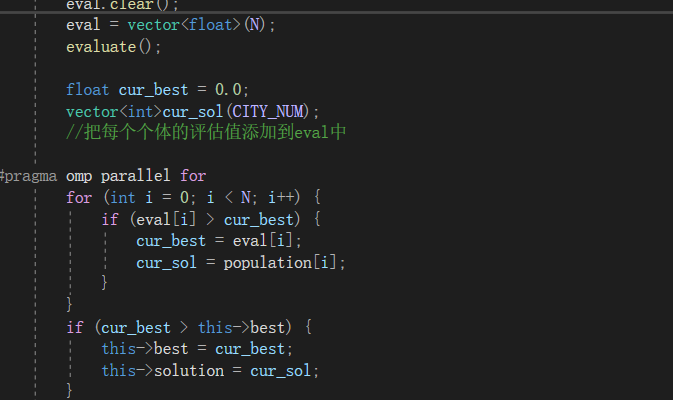
**7、变异算子**

遍历每个个体的每一个基因，每一次迭代都生成一个随机数，如果随机数小于交叉概率，那么就可以进行变异。变异的时候随机地生成一个城市数量之内的随机数记为index，将当前迭代的这个下标对应的基因与下标为index的那一个基因进行交换，完成变异。



**8、更新种群评估值和最优解**

重新初始化存放评估值的向量，然后再次调用评估成员函数，之后比较每个个体的评估值和全局最优解，如果有比全局最优解更加大的评估值，更新路线和全局最优解。

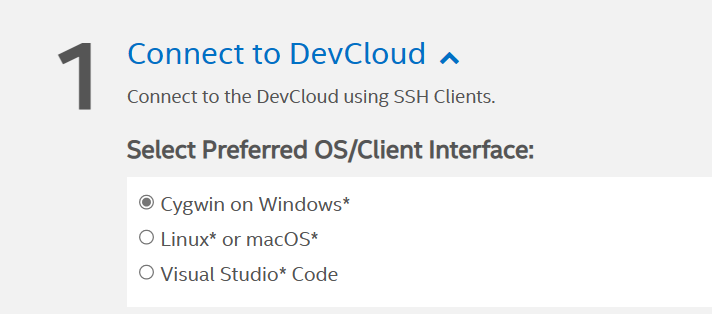


**三、运行环境的配置**

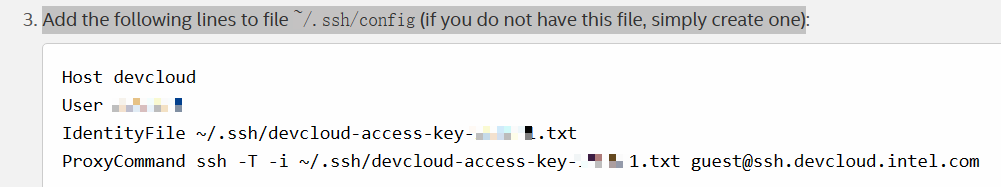
因为我们使用的电脑没有FPGA和intel的GPU，这样不能够完全发挥出oneAPI的优势，所以我们需要再DevCloud上面来运行我们编写的代码。

首先登录oneAPI的网站，进入到Get Started页面，看到Intel® oneAPI Base Toolkit，在此处我们点击“Get Started with your first Sample”。

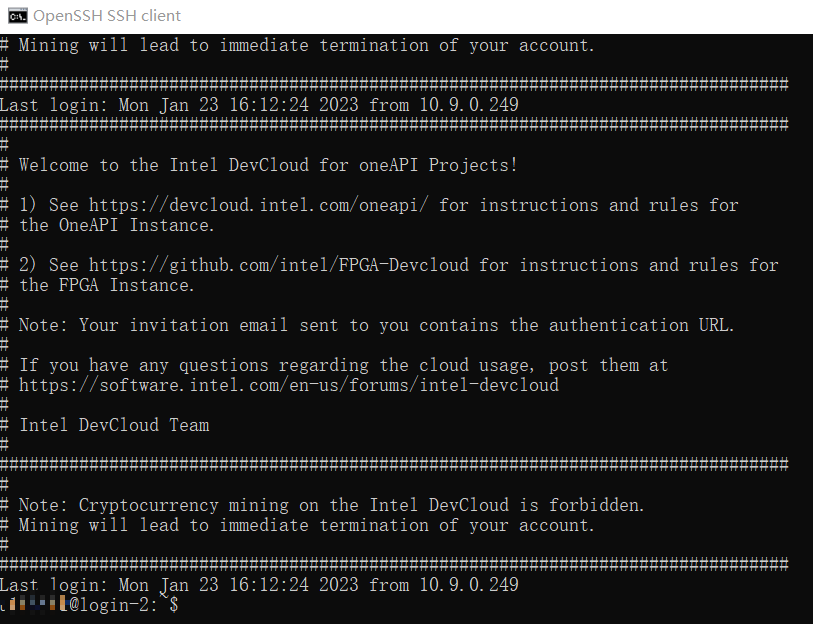
然后可以看到有Connect to DevCloud的选项，在这里选择Cygwin on Windows，不过并不使用Cygwin，而是开启win10命令行的ssh功能之后，直接用cmd来连接到devcloud。

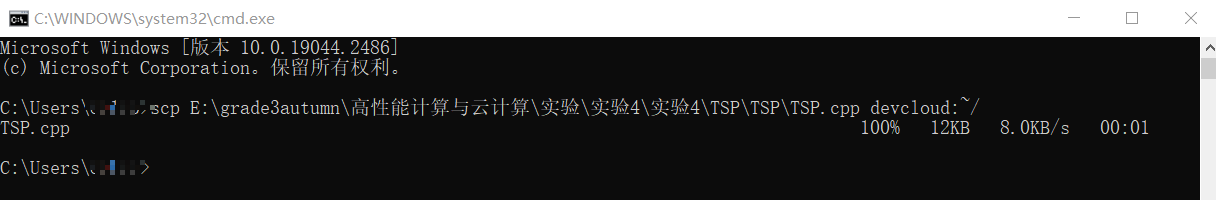


下拉找到Configure SSH Connection下面的Automated Configuration (Recommended)，然后点击Download setup-devcloud-access-XXXXXX.txt文件。下载好之后，把这个文件放到C:\Users\XXXX\.ssh文件夹下面。在这个文件夹下面进行操作，用记事本打开config文件，输入刚才网页上Manual Configuration章节中的“Add the following lines to file ~/.ssh/config (if you do not have this file, simply create one)”的内容。**需要注意的是，下面IdentityFile和ProxyCommand中sshkey的路径要修改为本机上的绝对路径，也就是C:\Users\XXXX\.ssh\devcloud-access-key-xxxxx.txt 。**



之后打开cmd，输入ssh devcloud，便可以远程登录devcloud主机。



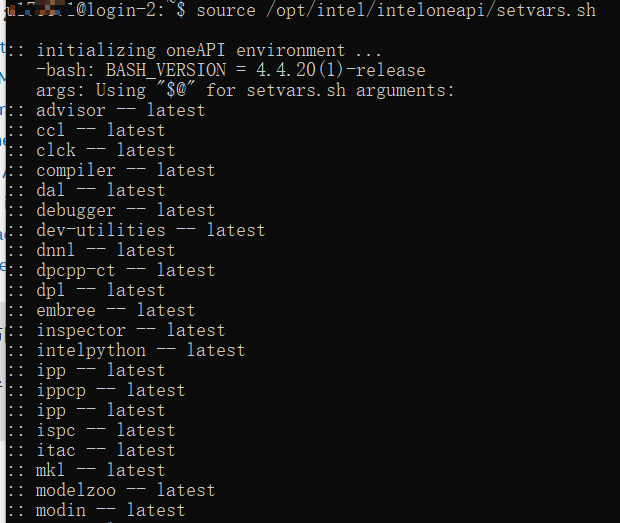
之后我们再打开一个cmd，用scp命令把我们本机上的代码给复制到devcloud。

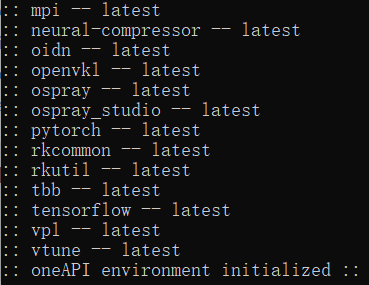


现在devcloud是还不能够执行dpcpp命令的，我们要先执行/opt/intel/inteloneapi/目录下面的setvars.sh脚本，才能够启用dpcpp命令。



输入source /opt/intel/inteloneapi/setvars.sh 启用oneAPI环境。



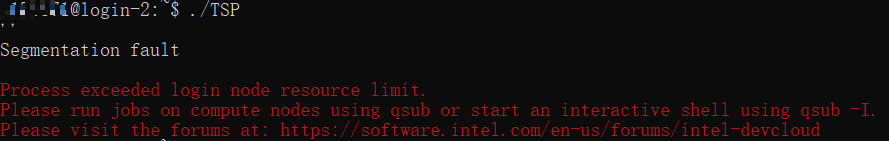


之后用dpcpp命令编译我们编写的代码，执行命令dpcpp TSP.cpp -o TSP

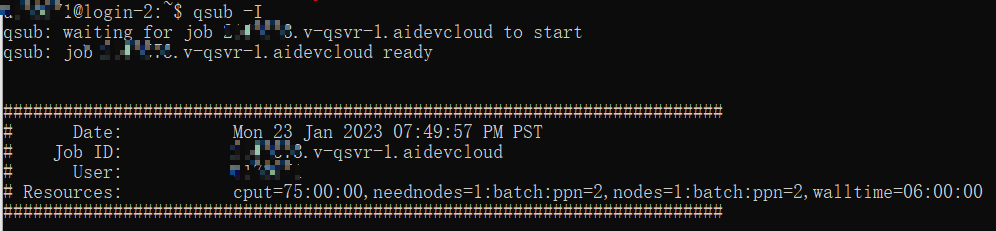




之后输入./TSP运行代码，会提示资源不足。

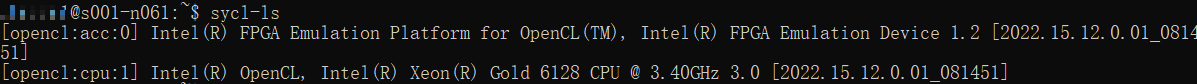


按照提示，输入qsub -I 进入计算节点



同样，计算节点的oneAPI环境也还没有启用，再次输入source /opt/intel/inteloneapi/setvars.sh 启用oneAPI环境。

输入sycl-ls查看FPGA设备。



可以看到有FPGA设备[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2022.15.12.0.01\_081451]。

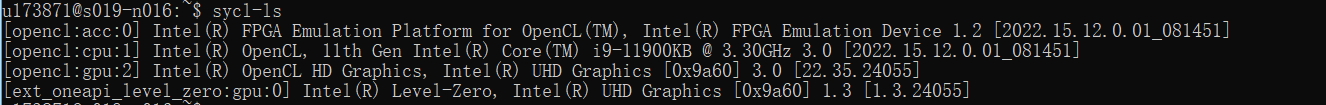
之后再执行./TSP便可以运行我们编写的程序了，运行完之后再将输出的文件再用scp命令复制到本机上。

当然除了使用FPGA计算也可以使用GPU进行计算。

先输入命令

qsub -I -l nodes=1:gpu:ppn=2 -d .

申请GPU计算节点



然后开启对double数据类型的支持

export OverrideDefaultFP64Settings=1

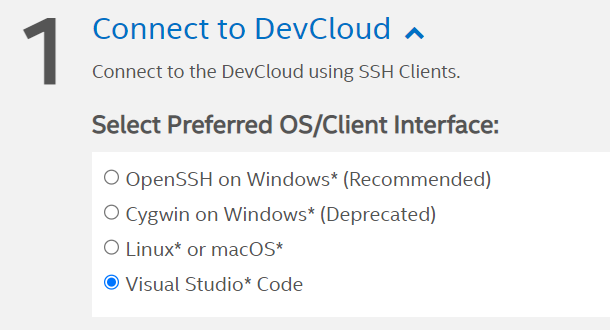
export IGC\_EnableDPEmulation=1

启用oneapi环境并编译

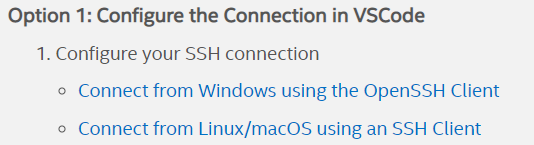
source /opt/intel/oneapi/setvars.sh

其余操作和前面的一样

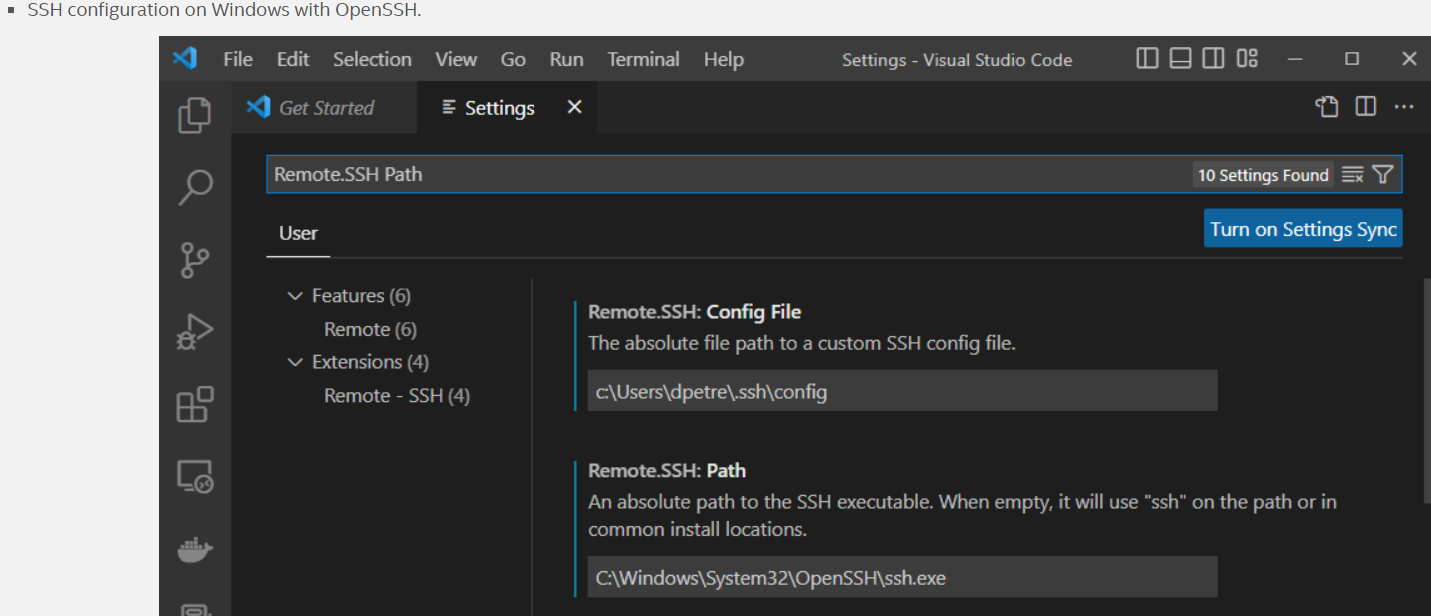
另外也可以用Visual Studio Code来连接devcloud。在Intel DevCloud官网“Get Started with your first Sample”页面中，Connect to DevCloud的选项中选择Visual Studio\* Code。



然后按照官方的指引Option 1:Configure the Connection in VSCode来操作。如果已经按照前文所述配置好了SSH，已经用cmd成功连接devcloud，那么这里1.Configure your SSH connection这一步不需要再操作。

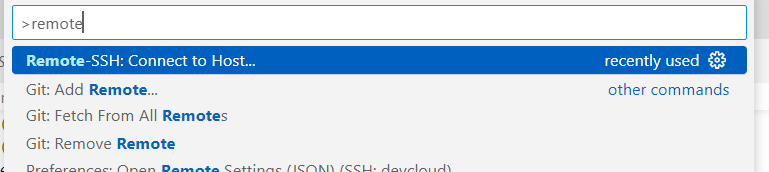


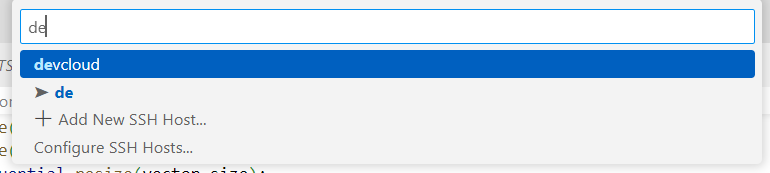
在官方指引的第5步5.Configure SSH Connection for VS Code for your OS中，按照OpenSSH的（而不是Cygwin的）示例来在VS Code设置Remote.SSH: Config File和Remote.SSH: Path这2项。



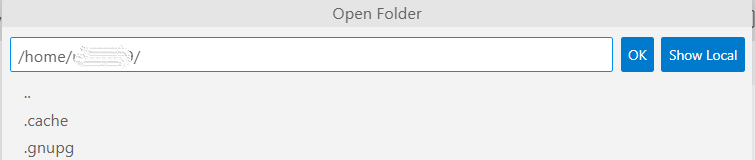
设置好这2项之后，已经可以连接devcloud了。

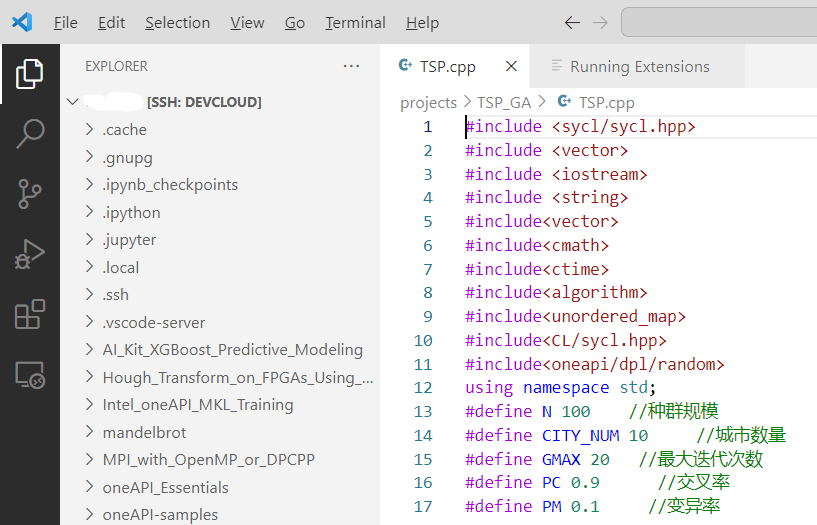
在VS Code中按Ctrl+Shift+P，输入remote-SSH: connect to host，回车，输入devcloud，回车，VS Code将自动弹出新窗口，该新窗口即远程连接到devcloud。





在连接到devcloud的窗口中，点击File->Open Folder，打开devcloud中某个目录，即可鼠标操作该目录下的文件，界面美观，目录结构清晰。这是VS Code操作优于命令行操作的地方。





点Terminal->New Terminal，即可在VS Code中使用命令行操作devcloud，就像前文所述在Windows cmd中操作一样。



之后的操作，与在Windows cmd中的操作相同。

#### 项目创新点

使用了Intel oneAPI对遗传算法的问题初始化、种群初始化、评估、选择、编译方面进行了并行优化，并且还运用了FPGA进行加速处理，提升了遗传算法解决旅行商问题的效率。

#### 项目成果

DevMesh地址：https://devmesh.intel.com/projects/tsp-ga

DevMesh截图：



项目的主要结果或截图展示：

使用oneAPI进行优化的算法与普通的串行算法进行比较：

参数：运行100代，种群数量为145000，城市数量为700，交叉概率为0.9，变异概率为0.1

图 1 串行遗传算法每一代的运行时间(ms)

图 2 使用oneAPI进行并行和加速改良的遗传算法运行时间(ms)

图 3 使用SYCL改良之后算法节约时间的比例

图 4 并行GA与串行GA运行时间相对比(ms)