

SUSTCSC竞赛培训-1

超算相关的基础知识介绍

SUSTCSC 竞赛组

本教程基于赖海斌同学B站教程「从0到1在学校Top500超算集群跑HPL程序」有兴趣同学可以前往观看完整版，同时本培训大量参考了CS149网课内容，感兴趣同学也可以前往<https://gfxcourses.stanford.edu/cs149/fall24> 查看

目录

- 超级计算机 (Bulk Picture)
- 节点登陆/Shell基础指令/Vim基础指令
- 处理器/进程/线程模型/节点/集群
- 通信网络/高速通信网络
- 操作系统
- 集群构建/作业调度系统
- 环境/容器
- 有关比赛

超级计算机 (Bulk Picture)

超算，全称是超级计算机，和个人主机（PC）不同的地方在于，超级计算机通常拥有非常复杂的硬件环境和软件环境，被用于解决一些非常复杂的问题

- **复杂的硬件环境**

- 不同型号、不同厂商的主机（一般也可能是统一的）
- 不同型号的CPU、GPU
- 复杂的节点网络拓扑

- **复杂的软件环境**

- 不同的操作系统
- 不同的运行环境

- **复杂的问题**

- 科学研究类问题：气候模拟、蛋白分子预测、RNA结构预测、流体力学
- 人工智能与大数据：图像生成、大语言模型
- 量子计算机：量子算法验证、量子线路模拟

为什么要学习超级计算机？

- 没有人可以了解一个超级计算机的全貌，当处于不同视角下的时候，我们对于超算应当有着截然不同的理解：
 - 我是超算的使用者 环境使用、管理，脚本提交
 - 我是超算的应用开发者 软件优化、MPI通信、异构计算、并行编程
 - 我是超算的管理者 权限控制、空间管理、资源监控、运维
 - 我是超算的设计者 网络拓扑设计、绿色集群设计、硬件配置



从节点登陆开始...

- ssh(Secure Shell)是一种加密的网络协议

- 安全远程登录 (ssh)
- 安全文件传输 (scp)
- 端口转发与隧道 (tunnel)

```
ssh-keygen -t id_rsa #生成密钥
```

```
ssh-copy-id -p port user@host # 传输密钥
```

```
ssh -p port user@host #登陆
```

- `ssh -p 18188 username@172.18.6.40` 输入密码即可

- 身份验证 (公钥登陆)

- 公钥 (复制本地 `~/.ssh/id_rsa.pub` 到服务器
`~/.ssh/authorized_keys`)
- 密钥 (通常是本地的 `~/.ssh/id_rsa`)

- 更复杂的使用可以用ssh来建立tunnel (tensorboard可能需要)

登陆后：欢迎来到命令行的世界

- 服务器通常通过CLI（Command Line Interface进行交互），而不是GUI（Graphic User Interface）
- 命令行通过Shell/Terminal执行和系统进行交互
- 常用的Linux语言命令可以看这里
- <https://www.practicelinux.com/home>
- 常用的指令：
 - 文件路由：cd, ls, cp, rm
 - 文件操作：cat, grep, echo, touch, mkdir
 - 重定向：>, >>
 - 系统查看：lscpu, nvidia-smi, df, du
 - 进程管理：top, kill, tmux, screen

Vim in one page

- Vim是在命令行环境下对文件进行编辑的工具
- 三种模式
 - 普通模式：执行指令（默认，esc返回）
 - 插入模式：编辑文本（按i进入）
 - 可视模式：选择文本（按v进入）
- 普通模式下的指令
 - 移动：h j k l 上下左右 w b 下一个/上一个单词 0 \$ 行首行尾 gg G 开头结尾
 - 编辑：i a 插入在光标前/后 o O 在行下上方插入 dd 删除当前行 yy 复制当前航 p/P 粘贴在光标前/后 u 撤销 Ctrl+r 重做
 - 保存与退出：:w 保存 :q 退出 :wq/:x 保存并推出 :q! 强制退出（不保存）
 - 搜索关键词：/ + word 输入vimtutor 即可学习

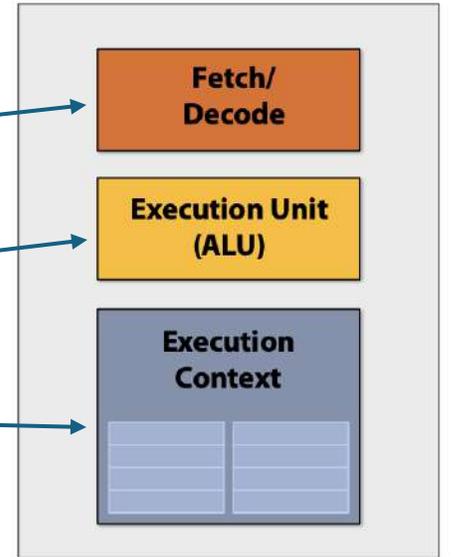
程序 (Program)

- 一串指令，用于和计算机来交互
- 高级语言（接近人类自然语言和数学表达）
 - Python、Java、C/C++
- 低级语言（更接近计算机硬件）
 - 汇编语言（汇编指令集如RISC-V）、机器语言（二进制）
- 通常来说连接这些语言需要「翻译」
 - 编译器 (Compiler)：一起翻译
 - 解释器 (Interpreter)：一行一行翻译
 - Java：编译器 (javac) + 解释器 (JVM混合执行)

中央处理器 (CPU)

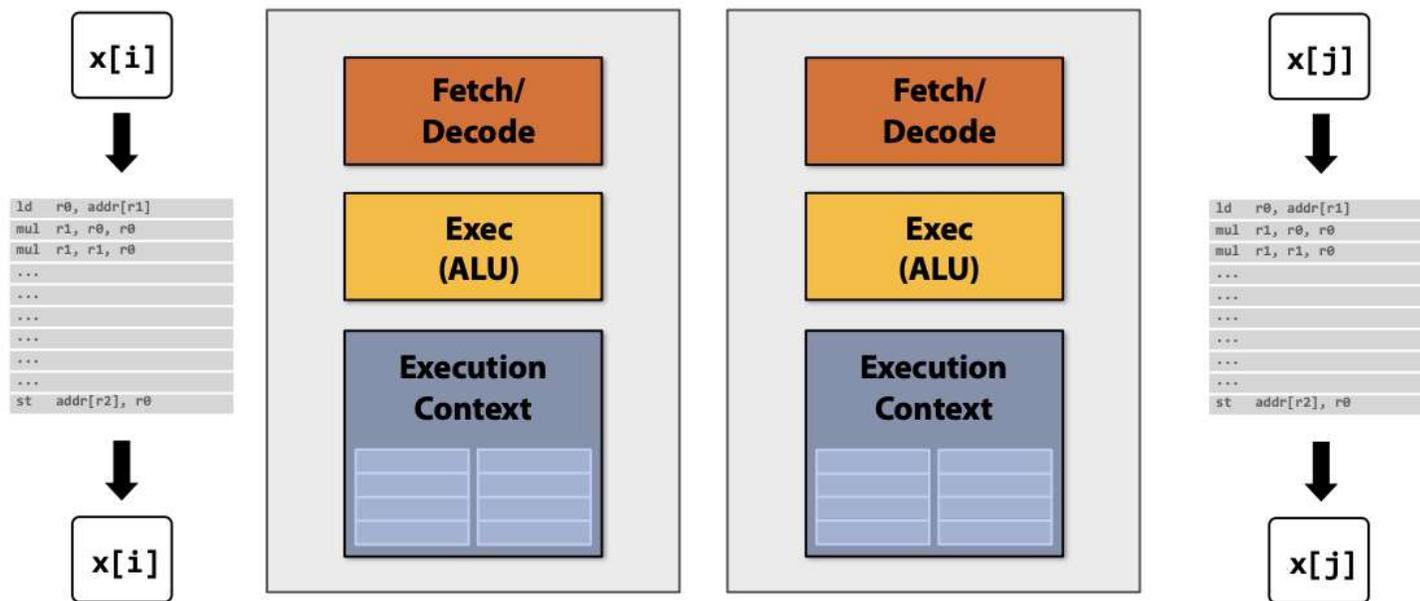


- 用于执行编译/解释后产生的机器码，处理「指令」(Instruction)
- 常见的PC用CPU有Intel i5/i7/i9，还有苹果的M1-4
- 组成部分包括
 - 控制单元 (Control Unit)
 - 运算单元 (Arithmetic and Logic Unit)
 - 寄存器 (Register)
 - 缓存 (Cache)
 - 内存 (Random Access Memory)
- 通常来说拥有一个完整的右边的结构，为有一个完整的核心



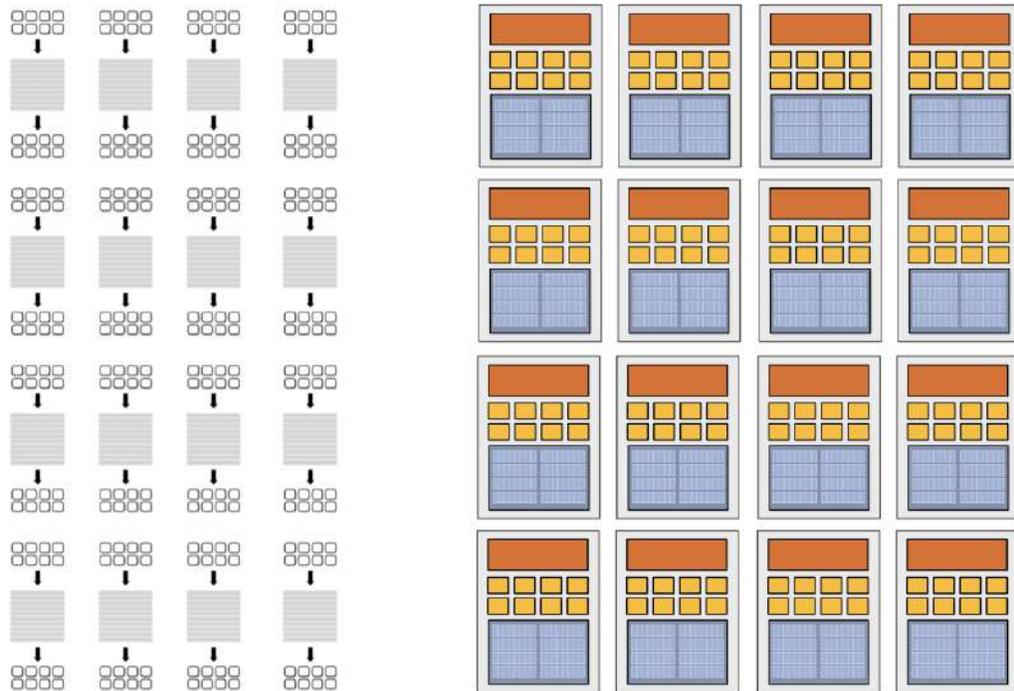
多核处理器 (Multi-core Processor)

- 在一个简单的Processor模型中，拥有独立的
 - 控制单元 (Control Unit)
 - 运算单元 (Arithmetic and Logic Unit)
 - 寄存器 (Register)
- 就可以视为一个核
- 右图便是2个核在同时执行两个指令
- 通常来说多核处理器共享Cache/内存
 - 但是每个核一般会有自己独立的L1 Cache

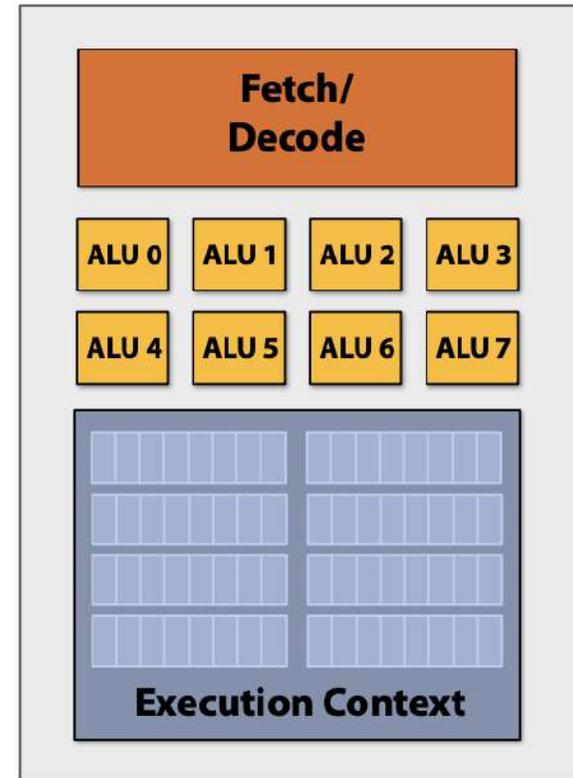


SIMD处理器

- SIMD代表Single Instruction Multiple Data, 单指令多数据流
- 在处理一些向量运算的时候特别有用



16 cores, 128 ALUs, 16 simultaneous instruction streams

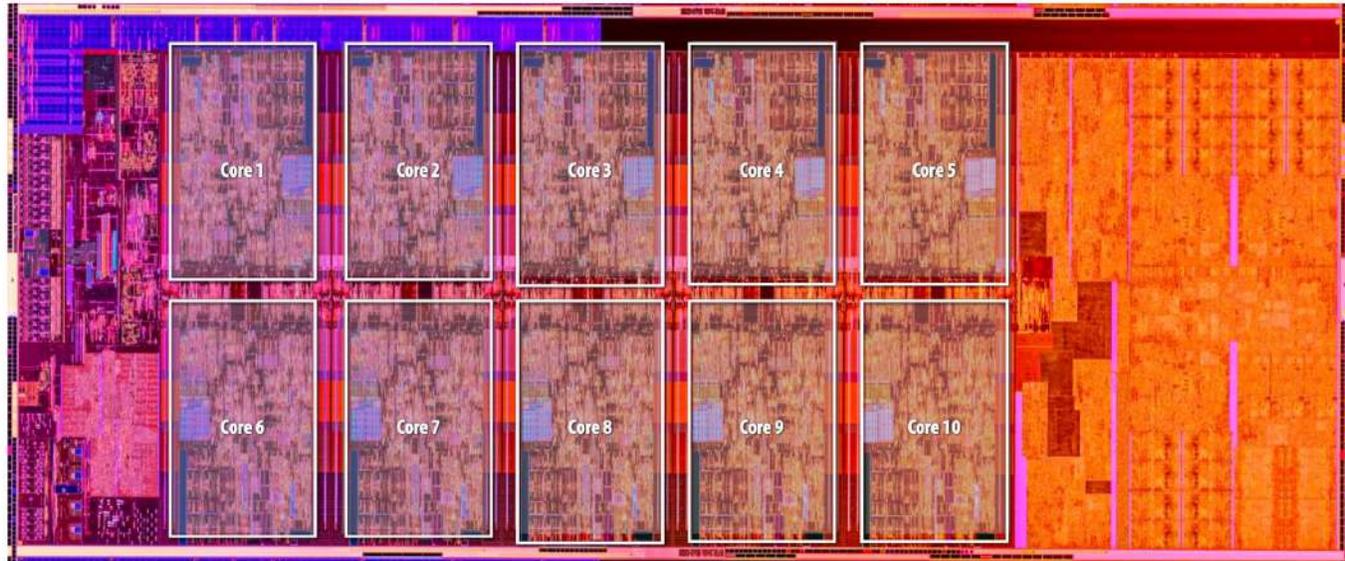


需要使用特定的编程范式比如AVX指令集

Check it out!

- Linux环境：lscpu
- Windows环境：systeminfo

Intel "Comet Lake" 10th Generation Core i9 10-core CPU (2020)



Two "big cores" + four "small" cores

2 "big" CPU cores

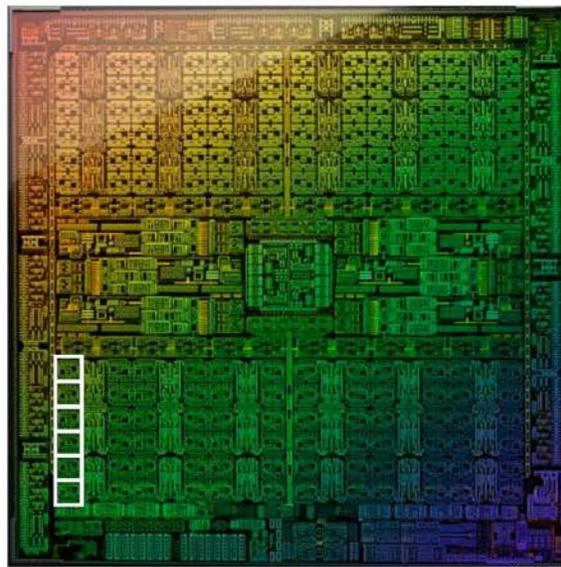
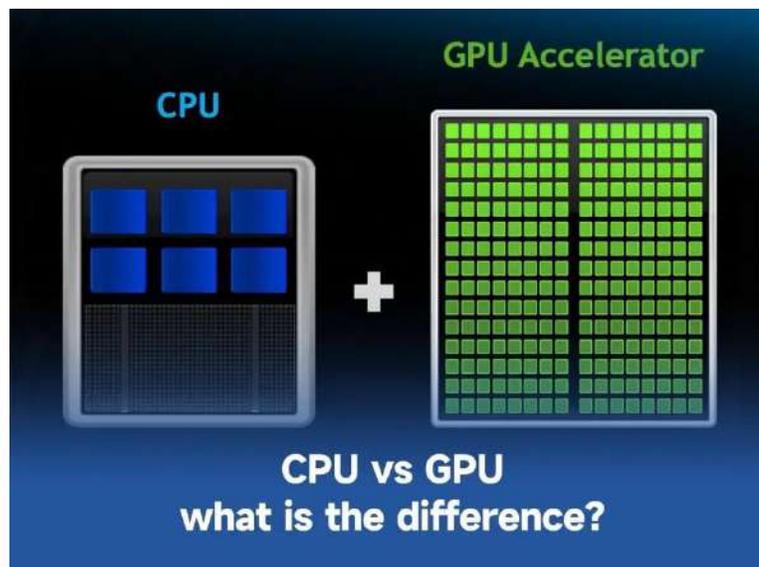
4 "small" CPU cores



图形处理器 (GPU)



- 和CPU不同的地方在于，GPU通常来说会有很多个计算核心，但是每个计算核心（SM）的能力相比CPU要弱很多
- 在GPU上，更利于执行并行的任务



CPU核心在服务器的主频通常来说在2.5GHz左右，GPU核心的主频通常就在1GHz上下

80 "SM" cores
128 SIMD ALUs per "SM" (@1.6 GHz) = 16 TFLOPs (~250 Watts)

NVIDIA的GPU可以通过nvidia-smi来查看信息

虚拟核/NUMA节点

- 每个核通常来说只能运行一个线程，而通过多线程技术（SMT），一个核可以运行多个线程

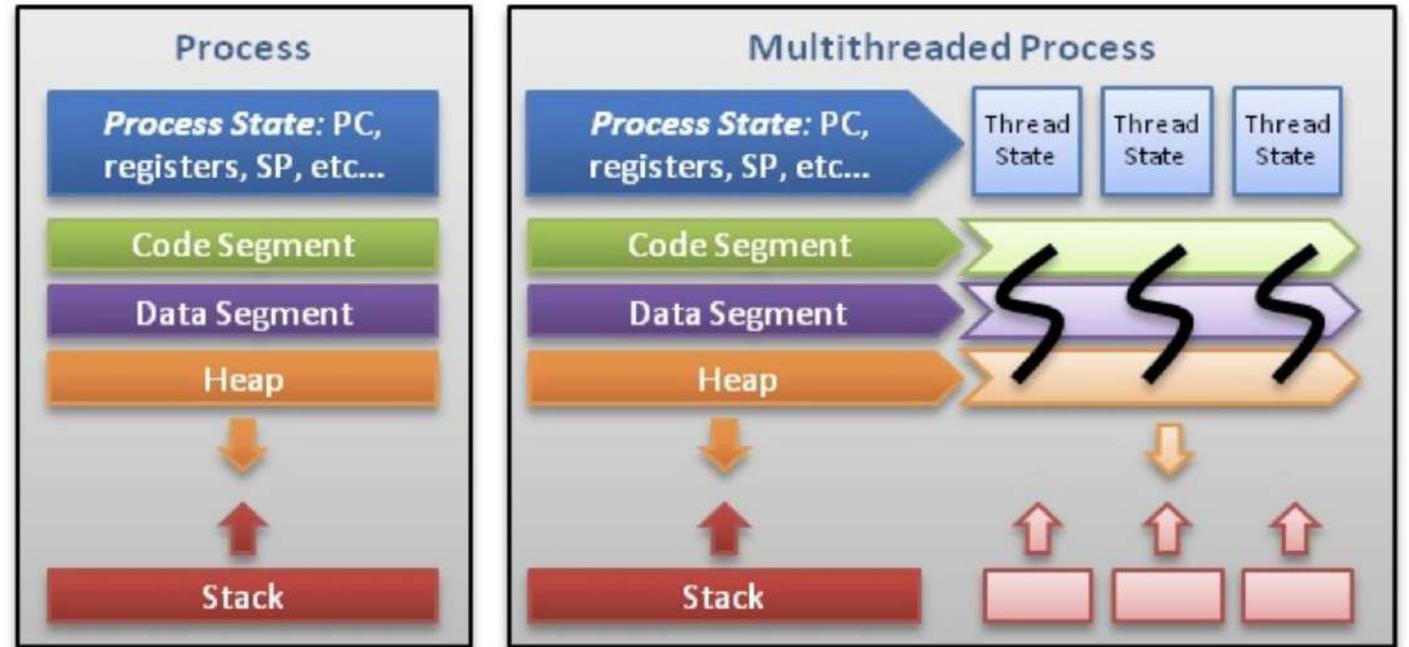
```
Thread(s) per core: 1  
Core(s) per socket: 16  
Socket(s): 2
```

- 虚拟核则是基于此的一个逻辑概念
- 为了解决核心增长而导致的内存竞争问题，NUMA（Non-Uniform Memory Access）的概念被提出

```
Socket(s): 2  
NUMA node(s): 2
```

线程/进程模型 (Thread/Process)

- **线程**只包含：
 - Stack 栈
 - Registers Snapshots
 - Program Counter
 - Thread-specific Data
- 和进程**其他线程**共享代码、数据、地址空间
- **进程切换**通常更耗时 (Context Switch)
- 操作系统管理线程/进程



Threads contain only necessary information, such as a stack (for local variables, function arguments, return values), a copy of the registers, program counter and any thread-specific data to allow them to be scheduled individually. Other data is shared within the process between all threads.

Linux通过htop/top查看进程情况， Windows/Mac可以直接查看进程管理器

节点/服务器 (Node/Server)

- 一个节点通常就是有以下组成部分：
 - 处理器 (Processor)
 - 内存 (Memory)
 - 总线 (Bus)
 - 存储 (Storage)
- 节点内通常需要操作系统来管理软件和硬件之间的交互
- 服务器也可以根据其配置不同分为
 - 刀片服务器：一个机箱多个节点
 - 机架服务器：一个机箱一个节点



机架服务器



刀片服务器

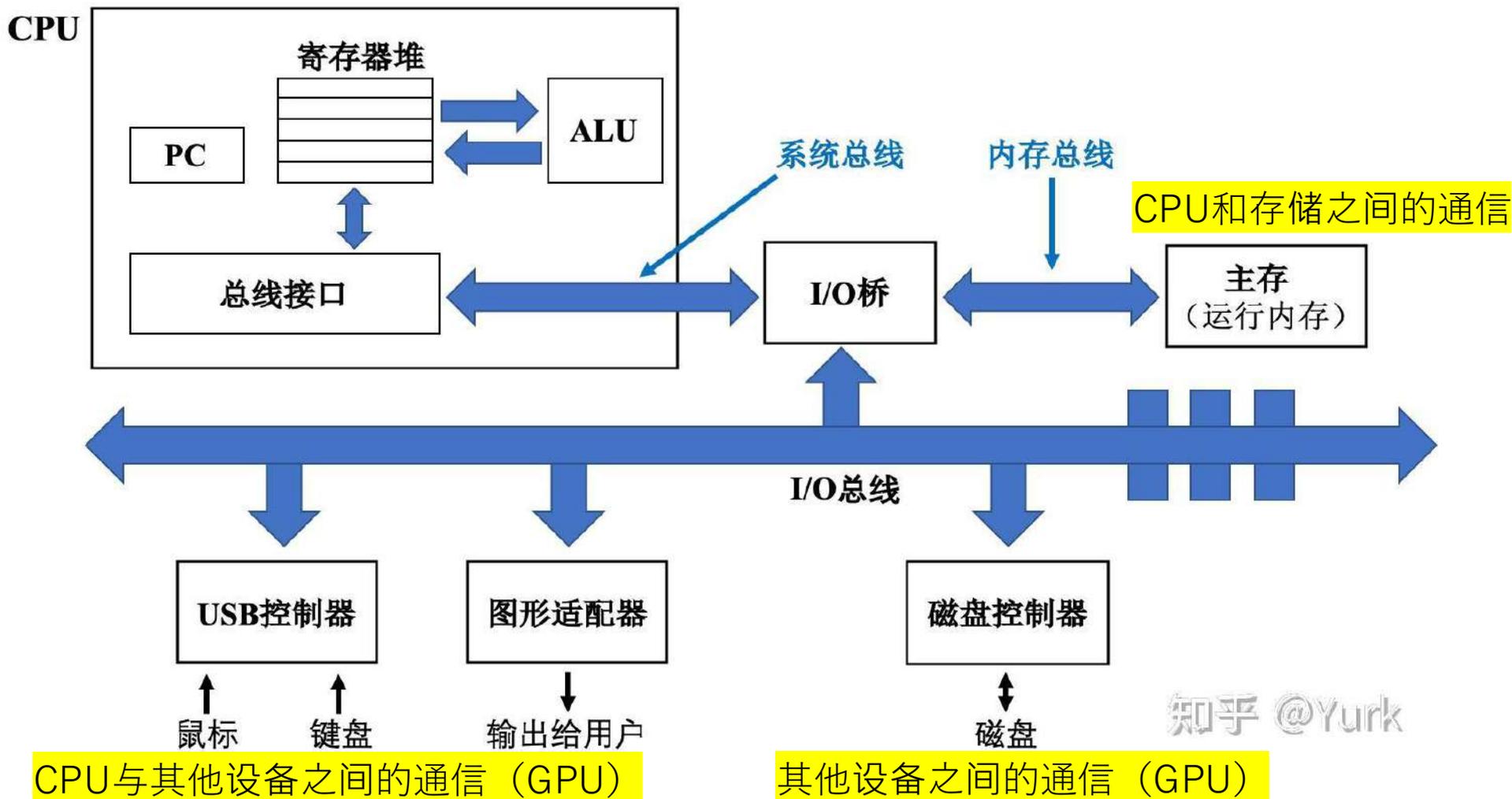
集群 (Cluster)

- 集群：几台主机连在一起就可以被称为一个集群，但通常一个集群会由一整个机柜来管理
- 服务器：计算机的一种
 - 运行快、负载高、提供计算/应用服务
 - CPU核心更多更快（PC核数通常是？）
 - 需要长时间可靠运行（PC一般的运行时间是？）
 - 强大的数据存储（PC的存储一般是什么量级？）
- 并行计算（不同程度）
 - 多核多线程：单机单进程（共享内存）
 - 多核多进程：单机多进程（通过管道通信）
 - 分布式：多机多进程（需要网络）



通信

CPU内核之间的通信

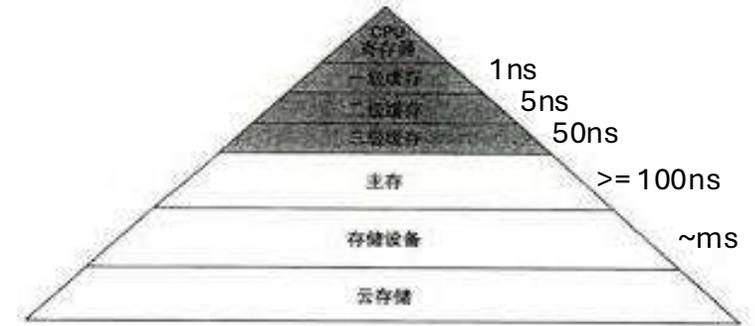


CPU和存储之间的通信

CPU与其他设备之间的通信 (GPU)

其他设备之间的通信 (GPU)

通信

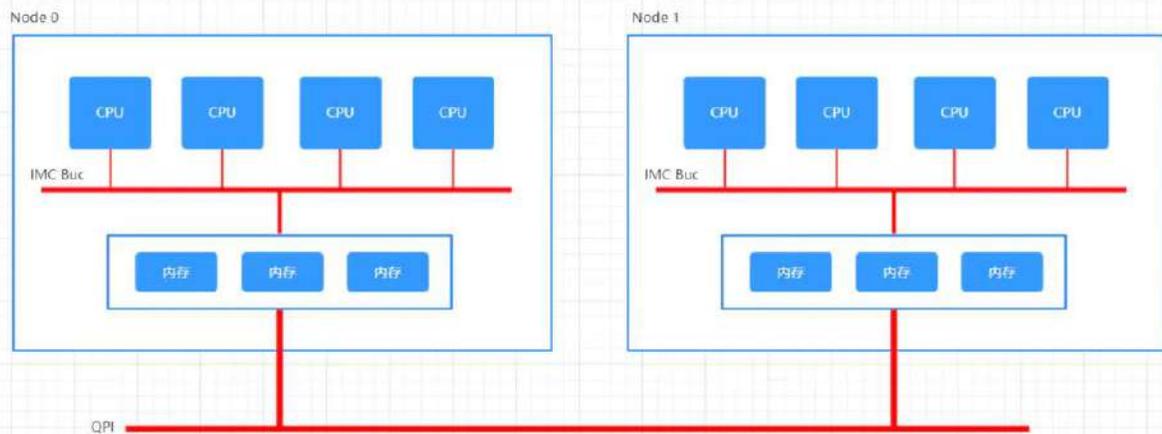
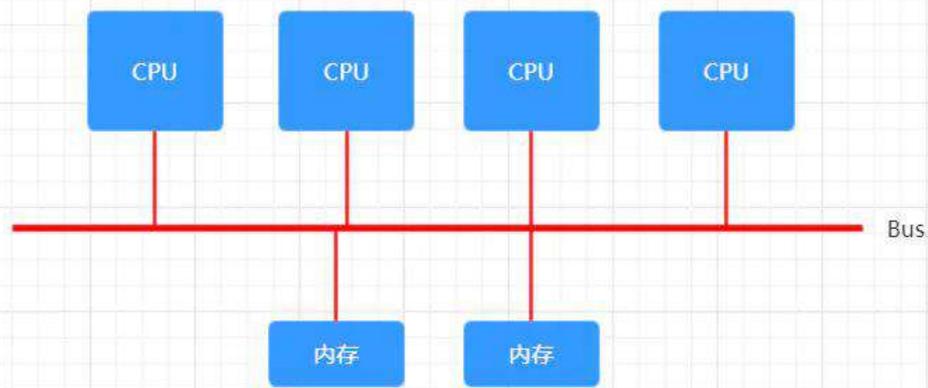


- CPU和存储之间的通信通常有多级访问机制
 - Register 寄存器
 - L1/L2/L3 Cache, 在物理位置上越来越远, 访存速度也逐级递减
 - Memory/Storage, 通常来说比Cache要耗时很多, Storage的访存通常会用异步的方式处理【PCIe】
- 多核CPU中, 内核之间的通信 (多线程/多进程) 通常是通过共享内存或者特定的通信通道 (缓存一致性协议) 实现的
 - 共享内存 (Shared Memory)
 - 缓存一致性协议 (MESI)
 - 原子操作和锁 (Atomic Operation)
 - MPI通信协议 (Message Passing Interface)
- 在内核通信模式中, 分为UMA和NUMA两种通信模式

UMA 和 NUMA

- UMA (Uniform Memory Access)
- 一致性内存访问
- 多个CPU通过同一根总线访问内存
- 串行访问，较慢，会成为瓶颈

- NUMA (Non-Uniform Memory Access)
- 非一致性内存访问
- 每个NUMA节点都分配了一块内存，这样的话，不同NUMA节点可以并行访问各自的内存
- 并行访问，跨NUMA Node访存慢



CSDN @张孟浩_jay

CSDN @张孟浩_jay

还有一种访存模式是DMA/RDMA (Direct Memory Access/Remote DMA) 大家感兴趣可以搜一搜

通信

- CPU和GPU的以及GPU与GPU之间的通信通常需要经过PCIe插槽或者NVLink来实现
 - PCIe通常带宽20~40GB/s的量级
 - NVLink的带宽200GB~400GB/s的量级
- 内存交互机制通常需要Cuda通信库（针对Nvidia显卡系列）
 - 数据拷贝，通过cudaMemcpy来显示传输内存
 - 固定内存，通过Pinned Memory加速（DMA）
- 驱动：通常需要装在Nvidia Driver
- CPU和其他设备（网卡等）的交互也类似，需要有特定的Driver

通信

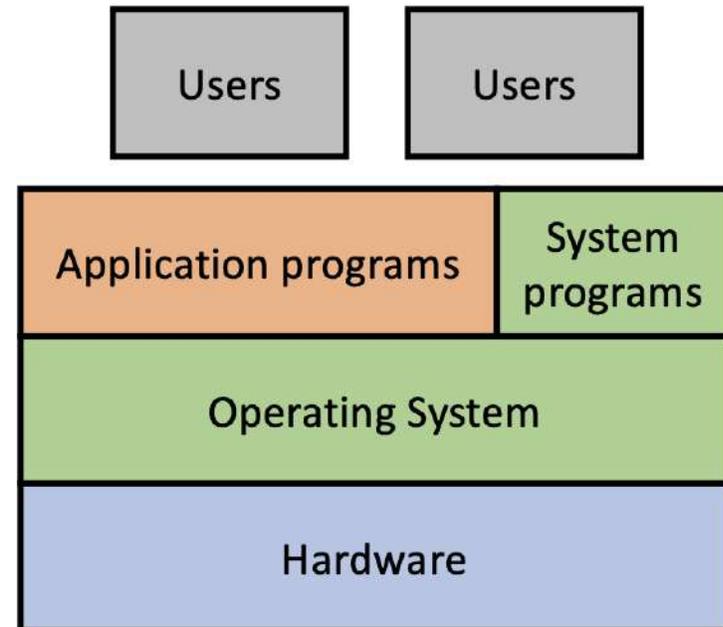
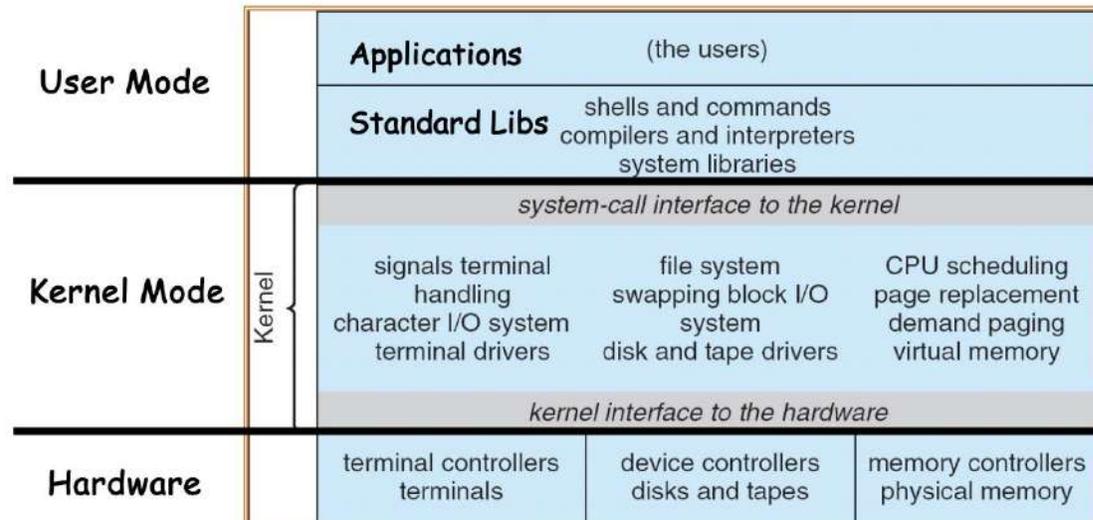
- 节点间通信通常分为Ethernet 以太网通信和Infiniband高速网络通信
 - 通常来说以太网的通信速度在10GB/s的量级
 - Infiniband高速网络的通信速度在100GB/s的量级
- 操作系统的选择会影响Infiniband的通信效率
- Infiniband的传输协议就是基于RDMA的通信协议
- 以太网的通信协议通常需要分为自顶向下的五层（TCP/IP）：
 - 应用层 》 传输层 》 网络层 》 链路层 》 物理层
- 不需要层层打包，因而Infiniband的通信速度更快，常用于高性能集群中

通信模型

- OpenMP 共享内存并行
 - 是基于 **多线程** 的共享内存并行模型，一般用于**单台多核CPU服务器**
 - 指令级并行（通过`#pragma omp parallel` 让编译器来负责并行）
- MPI 分布式并行
 - 是基于 **进程间通信** 的分布式内存模型，一般用于**多机集群**
 - 代码级并行（需要手动接受/发送消息协调多节点计算）
- Cuda
 - 是基于 **CPU-GPU通信** 的异构模型，一般用于**GPU节点上**
 - 通过内核函数（Kernel）触发事件/流（Event/Stream）在GPU上启动并行计算
 - 细粒度代码级并行（需要手动撰写Cuda Kernel）

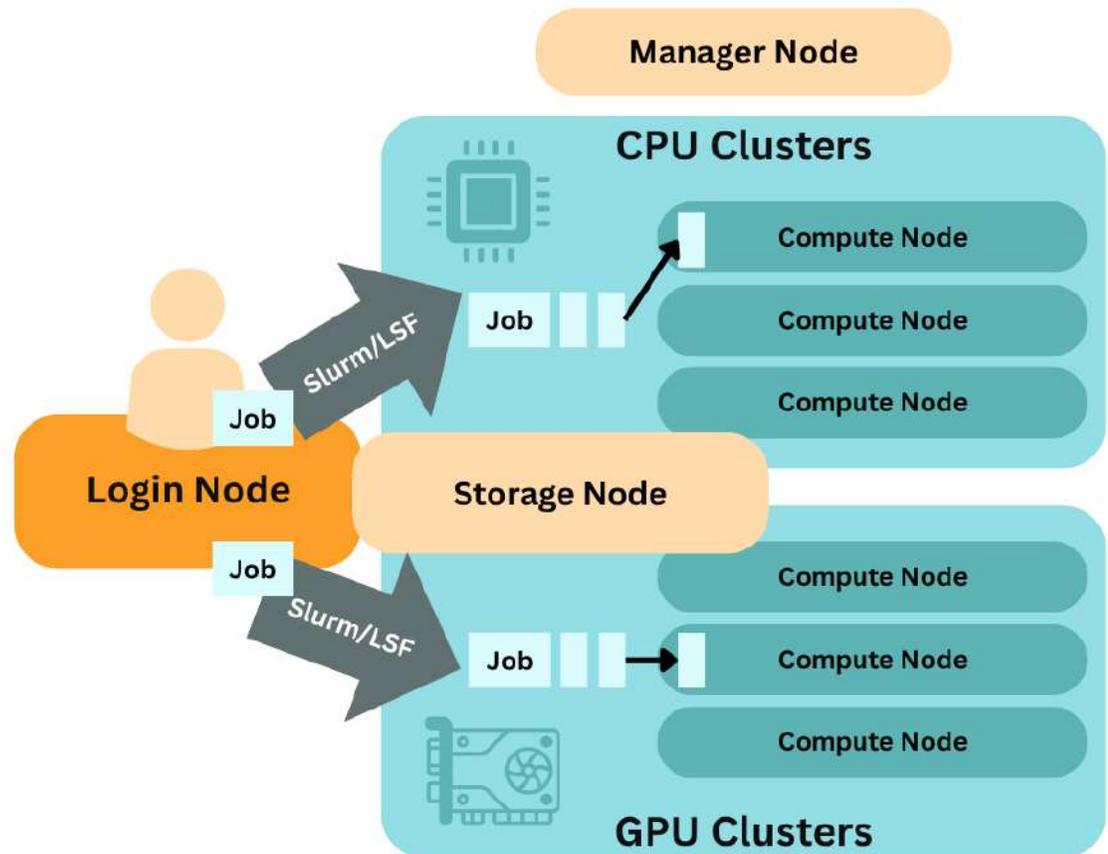
操作系统

- 在超算系统中，一个好用的操作系统很重要，如何定义好用？
 - 进程管理：高效处理任务的调度、分配和回收
 - 存储管理：分配、共享、保护、扩张
 - 设备管理：设备分配、设备传输控制、设备独立性
 - 文件管理：文件存储空间的管理、目录管理、文件操作管理、文件保护
- 有时候操作系统的选择会影响
 - 程序性能、传输效率、任务运行效率



集群构建

- 已经介绍了这么多概念的，我们可以看看一个超算集群是怎么构建起来的
- 通常一个集群会有
 - 登录节点（提交作业）
 - 计算节点（运算任务）
 - 存储节点（存储大型文件）
 - 管理节点
- 通常来说，集群内部会有**共享目录**，保证在登录节点和计算节点程序访问文件的路径是相通的



作业调度系统

- 核心价值在于资源的高效利用与公平分配
 - 保证每个任务按规则排队执行（不用手动执行了）
 - 支持任务并行、分布式执行（多核多节点）
 - 合理预估资源占用，防止系统过载
 - 记录任务状态与日志，方便跟踪和管理
- SLURM常用指令
 - sinfo 查看队列信息
 - sbatch 提交作业
 - squeue 查看作业执行情况
 - scancel 取消作业

作业脚本案例

```
#!/bin/bash
```

```
#SBATCH --job-name=cpu_job_test
```

```
#SBATCH --partition=8175m
```

```
#SBATCH --nodes=1
```

```
#SBATCH --ntasks=1
```

```
#SBATCH --cpus-per-task=4
```

```
#SBATCH --time=02:00:00
```

```
#SBATCH --output=cpu_job_%j.out
```

```
#SBATCH --error=cpu_job_%j.err
```

```
# 作业名称
```

```
# 提交到队列
```

```
# 所需节点数
```

```
# 总任务数
```

```
# 使用的 CPU 核心数
```

```
# 最长运行时间
```

```
# 标准输出日志
```

```
# 标准错误日志
```

```
lscpu
```

环境管理

- 环境工程通常来说在计算机里面可以单列一门课程
- 简单来说，环境工程的意义在于，你要高速计算机要去「哪里」找一个程序所需要的库文件
- 通常来说环境都是使用环境变量来管理的
- 最基础的环境管理就是通过手动设置环境变量就可以
 - 使用过Windows的朋友应该都还记得每次下载一个软件都要去环境变量里面添加一条PATH，这就是手动设置环境变量
 - 在Linux中可以通过下面的指令来设置环境变量，常用的环境变量有
 - `export PATH=$PATH:/bin # 二进制文件`
 - `export LIBRARY_PATH=$LIBRARY_PATH:/lib # 静态链接库`
 - `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/lib # 动态链接库`
 - `export C_INCLUDE_PATH=$C_INCLUDE_PATH:/include # 头文件`

环境管理

- 超算环境中的环境管理只能说是地狱
- 因为
 - 在登录节点配置环境，但是在计算节点运行，如何保证两个操作系统中的环境是**一致**的？
 - 每个节点上的驱动版本/MPI版本/软件版本可能都不一致，如何保证一个并行程序/分布式程序可以在不同节点上运行起来？
- 因此我们需要一些软件来帮我们辅助
 - 前面提到的挂载的文件系统也减轻了我们在配置环境的时候的崩溃

环境管理

- ModuleFiles

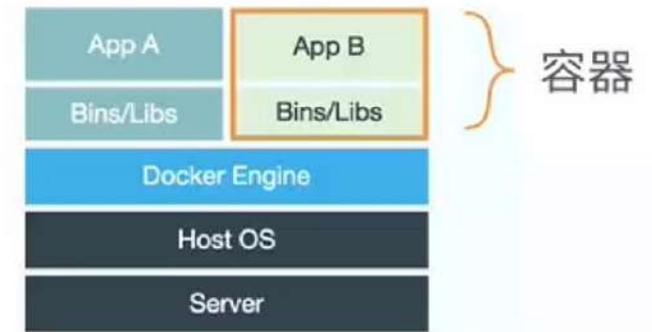
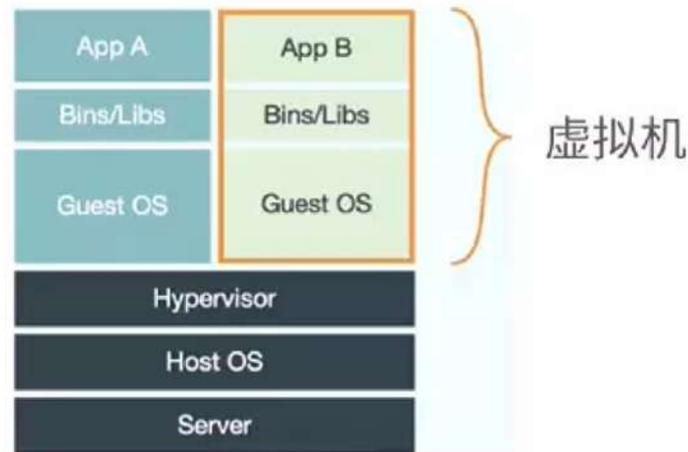
- module avail
- module load
- module purge

- Conda

- conda create -n env python=3.10
- conda activate env
- conda deactivate

- 容器 (Singularity)

- singularity build --sandbox
- singularity run/exec --nv
- singularity shell



有关比赛

-  比赛时间：即刻开始 ~ 7月25日
-  比赛地点：线上形式，校外人员需要使用VPN连接校内集群
-  比赛文档：<https://handicraft-computing-team.github.io/sustcsc-doc/>
-  基础赛道：请点击首页基础赛道查看详情
-  进阶赛道：请点击首页进阶赛道查看详情
-  比赛内容：代码运行 + 报告撰写（根据具体赛题要求）

温馨提醒

- 比赛分为多阶段测评，有赛中测评和赛后测评，赛中测评后在排行榜上排行前三的在总分/单项奖中有加分
- 请仔细查看对应  提交指南

代码提交「赛中评测」

请根据每道赛题指定的评测方案打包代码，在规定评测提交时间内提交到集群对应的文件夹 (`/work/share/sustcsc-submit/{team_id}/{taskname}`) 中，评测提交时间为比赛期间**每周星期二、星期五** (7月8日、11日、15日、18日、22日、25日) 中午12:00至下午18:00，评测结果将在当天晚上或第二天白天统一更新至排行榜：

- 每周五赛中评测结束后，第一名的队伍将在**最终总得分**获得 +2% 的分数加成，第二名 +1.5%，第三名 +1% (可叠加)
- 每周五赛中评测结束后，Rust LWE挑战赛单项排名第一的队伍将在**评选Rustacean**单项得分中获得 +2% 的分数加成，第二名 +1.5%，第三名 +1% (可叠加)

代码提交「赛后评测」

7月25日最终提交时间为23:59，组委会会根据最后一次提交结果进行性能分评测

代码提交「赛中评测」

请根据每道赛题指定的评测方案打包代码，在规定评测提交时间内提交到集群对应的文件夹 (`/work/share/sustcsc-submit/{team_id}/{taskname}`) 中，评测提交时间为比赛期间**每周星期二、星期五** (7月8日、11日、15日、18日、22日、25日) 中午12:00至下午18:00，评测结果将在当天晚上或第二天白天统一更新至排行榜：

- 每周五赛中评测结束后，第一名的队伍将在**最终总得分**获得 +2% 的分数加成，第二名 +1.5%，第三名 +1% (可叠加)
- 每周五赛中评测结束后，DiT图像生成挑战赛单项排名第一的队伍将在**评选AI Specialized**单项得分中获得 +2% 的分数加成，第二名 +1.5%，第三名 +1% (可叠加)

代码提交「赛后评测」

7月25日最终提交时间为23:59，组委会会根据最后一次提交结果进行性能分评测

资源使用

队列名称	内存	CPU类型	GPU类型	节点数
8175m	196GB	Intel Xeon Platinum 8175M @ 2.50GHz	-	27
8v100-32	32GB * 8	Intel Xeon Platinum 8255C @ 3.80GHz	8 * Tesla V100 32GB	1

- 集群配置

- 每道赛题限制使用资源如下

	量子计算	Rust	Cloverleaf	HGEMM	DiT	WRF
CPU	1节点	1节点	<=2节点	1节点	1节点	<=2节点
GPU	/	/	/	1张卡	<=4张卡	1张卡

- Slurm作业系统提交限制如下

- 最多提交10个作业脚本
- 最多同时运行5个作业脚本

- 请各位合理利用资源

培训日程提醒

2025年6月29日星期日		
时间	具体安排	地点
20:00 - 22:00	WRF 数值天气预报挑战 (通用 HPC 优化思路) 讲解一	腾讯会议
2025年6月30日星期一		
时间	具体安排	地点
10:00 - 12:00	WRF 数值天气预报挑战 (通用 HPC 优化思路) 讲解二	南方科技大学 三教 113 教室
14:00 - 16:00	WRF 数值天气预报挑战 (通用 HPC 优化思路) 讲解三	南方科技大学 三教 113 教室
2025年7月1日星期二		
时间	具体安排	地点
10:00 - 12:00	CUDA 高性能计算编程一	南方科技大学 三教 113 教室
14:00 - 16:00	CUDA 高性能计算编程二	南方科技大学

ix基本命
规则介绍
戈

00		三教 113 教室
2025年7月2日星期三		
时间	具体安排	地点
10:00 - 12:00	Python GPU 计算编程一	南方科技大学 三教 113 教室
14:00 - 16:00	Python GPU 计算编程二	南方科技大学 三教 113 教室
16:30 - 17:30	参观南科大科学与工程计算中心	科学与工程计 算中心机房
2025年7月3日星期四		
时间	具体安排	地点
10:00 - 12:00	OpenAI Triton 编程一	南方科技大学 三教 113 教室
14:00 - 16:00	OpenAI Triton 编程二	南方科技大学 三教 113 教室
2025年7月4日星期五		
时间	具体安排	地点
10:00 - 12:00	GPU 优化案例实战	南方科技大学 三教 113 教室

Q & A

- 请填写此份问卷星链接保证各位对本次比赛基本情况有了解
- 最后会有确认报名赛道环节，请组员内部商量之后填写一份即可

