HPL

Haibin Lai

Southern University of Science and Technology

Content

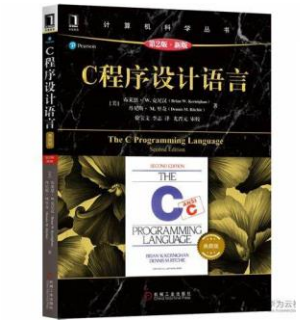
本期讲解内容

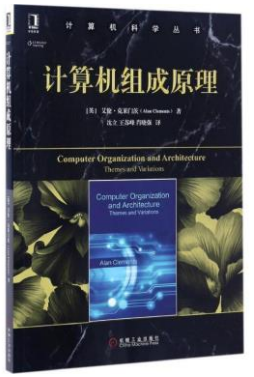
- 1.基础系统知识
- 2.HPL基础知识
- 3.HPL所需库
- 4.HPL编译
- 5.HPL运行与调优
- 6.进阶：更多HPL学习

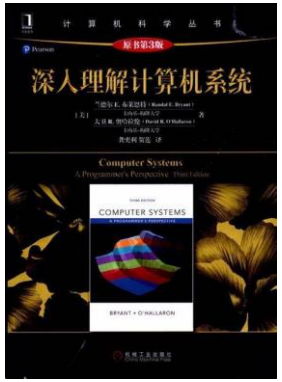
前置知识

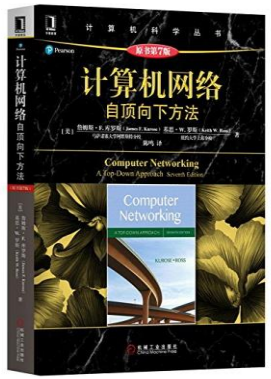
- Linux
- Vim
- C/C++
- shell

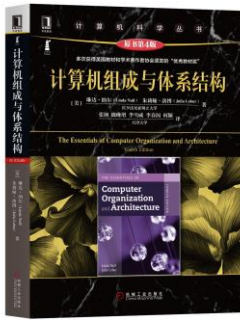
进阶知识

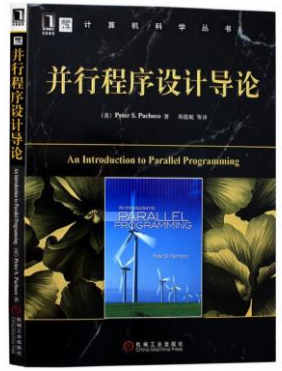












1.基础系统知识

- 节点 CPU 核 进程 线程
- 通信知识
- 指令集 汇编 编译 操作系统
- 作业调度系统
- 容器 环境 集群建设

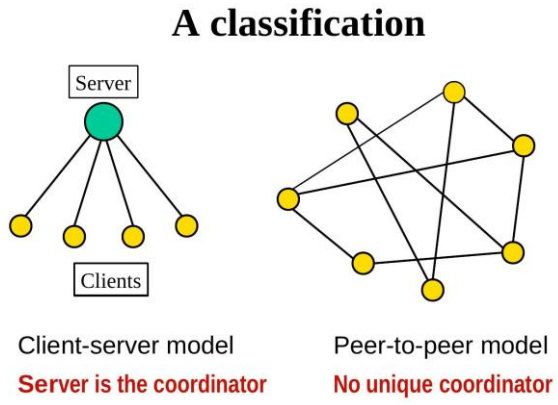
让我们从最大的开始……

- 集群Cluster：几台机器连在一起，就叫集群。

- 服务器是计算机的一种，运行快、负载高，提供计算或者应用服务。服务器具有高速的CPU运算能力、长时间的可靠运行、强大的I/O外部数据吞吐能力以及更好的扩展性。

刀片式服务器

- 分布式Distributive：一个程序或系统，只要运行在不同机器上，就叫分布式。



节点

- 在服务器中，通常都包含一个或多个节点。节点指系统中的一个独立处理单元。
- 一个节点包含了以下主要组成部分：

- 1.处理器（Processor）： 一个节点通常包含一个或多个处理器，每个处理器可能包含一个或多个核心。这些处理器用于执行计算任务。
- 2.内存（Memory）： 节点拥有自己的内存单元，用于存储程序和数据。在并行计算中，节点之间的通信通常需要通过共享内存或消息传递等机制实现。
- 3.互联网络（Interconnection Network）： 多个节点通过一个互联网络相互连接，使它们能够进行通信和协同工作。互联网络的设计对并行计算系统的性能起着关键作用。
- 4.操作系统支持（Operating System Support）： 每个节点都运行着一个操作系统，负责管理节点上的资源、调度任务以及协调节点之间的通信。

中央处理器 CPU：一个物理概念，负责读取、编译和执行指令，它主要的部分有：

控制器（Control Unit，CU）、

运算器（Arithmetic and Logic Unit, ALU）、

寄存器（Register）、

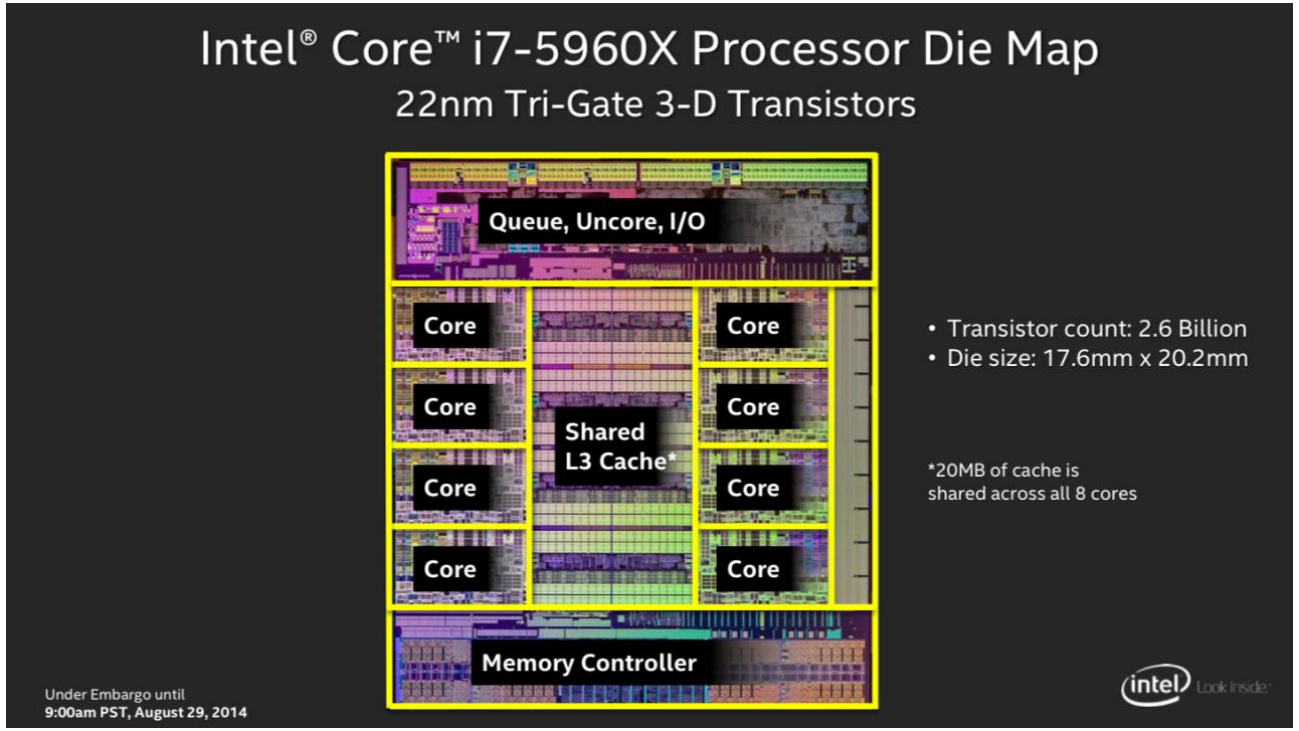
高速缓存器（Cache）以及总线。

（通常CPU数量指的是电脑中有多少个CPU卡槽或者CPU硬件数；一般家用电脑中只有一个CPU，服务器中会有多个）。

Windows: systeminfo
Linux : lscpu

内核 Core：一个物理概念，是读取和执行程序指令的独立处理单元，是CPU最重要的组成部分。CPU中心隆起的芯片就内核。（一个CPU可以有多个核，比如我PC的CPU就有两个核，我们服务器的一个CPU有64个核）

虚拟核（逻辑核、线程）Thread：一个逻辑概念，等于核执行的线程数。如果一个核可同时执行2个线程，那么该核的虚拟核数为2。（通常一个核某一个时刻只能执行一个线程，也就是一个核的线程数是1，但由于多线程技术（Simultaneous Multithreading，SMT）的出现，使得一个核具备多线程同时执行的能力。）



进程与线程

进程：程序运行的一个实体的运行过程，操作系统进行资源（包括内存、磁盘IO等）分配的最小单位，系统赋予其独立的内存地址空间；进程之间在资源上是相互独立的，但因为CPU资源有限，所以进程之间会相互制约。

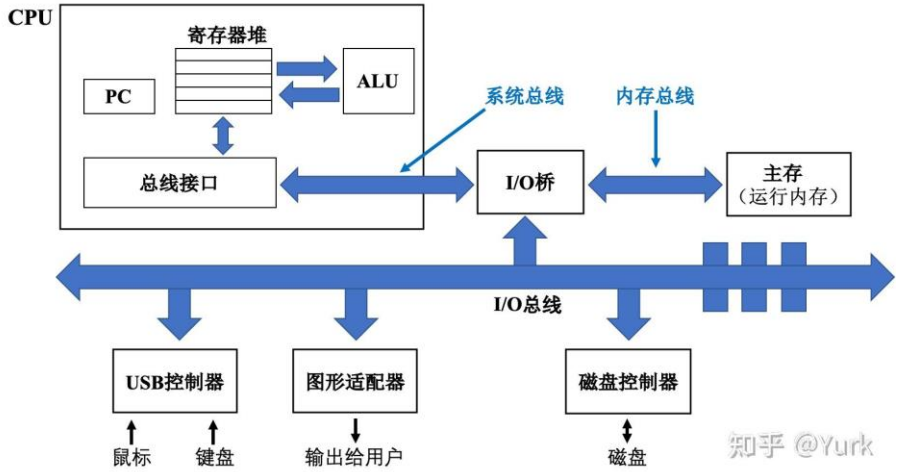
厨房

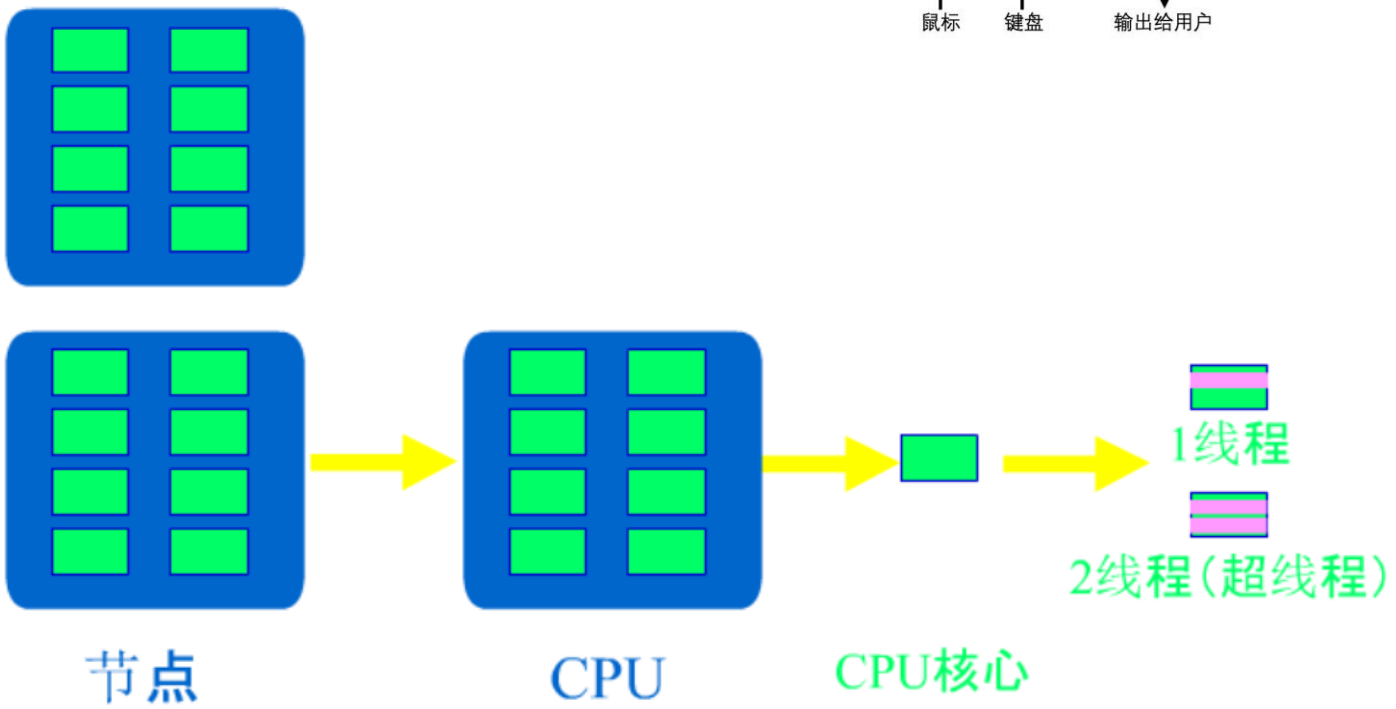
线程：进程运行和执行的最小调度单位；同一进程内的线程具有相同的地址空间，共享进程内的资源，可以相互通信相互影响。线程是一个基本的CPU执行单元，也是程序执行过程中的最小单元，由线程ID、程序计数器、寄存器集合和堆栈共同组成。线程的引入减小了程序并发执行时的开销，提高了操作系统的并发性能。

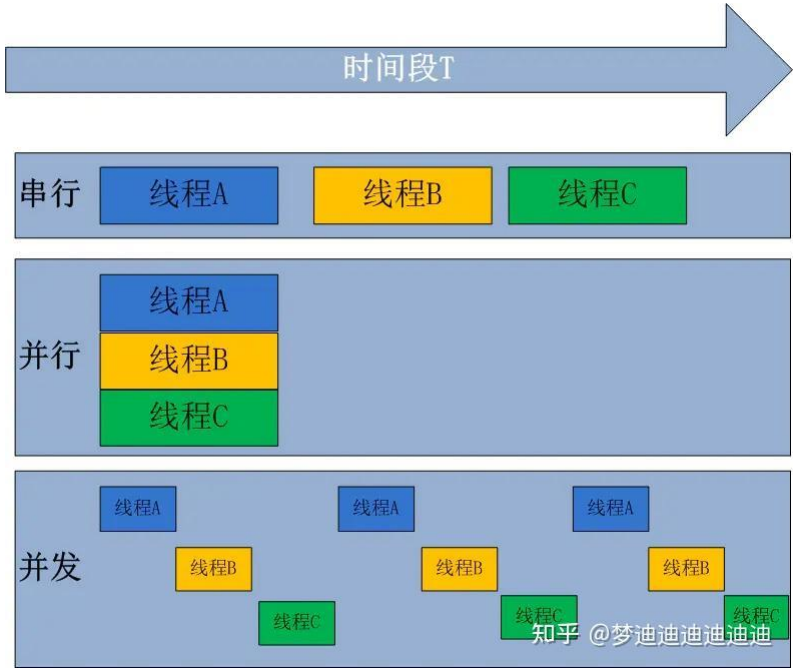
厨师

linux：htop

运行关系

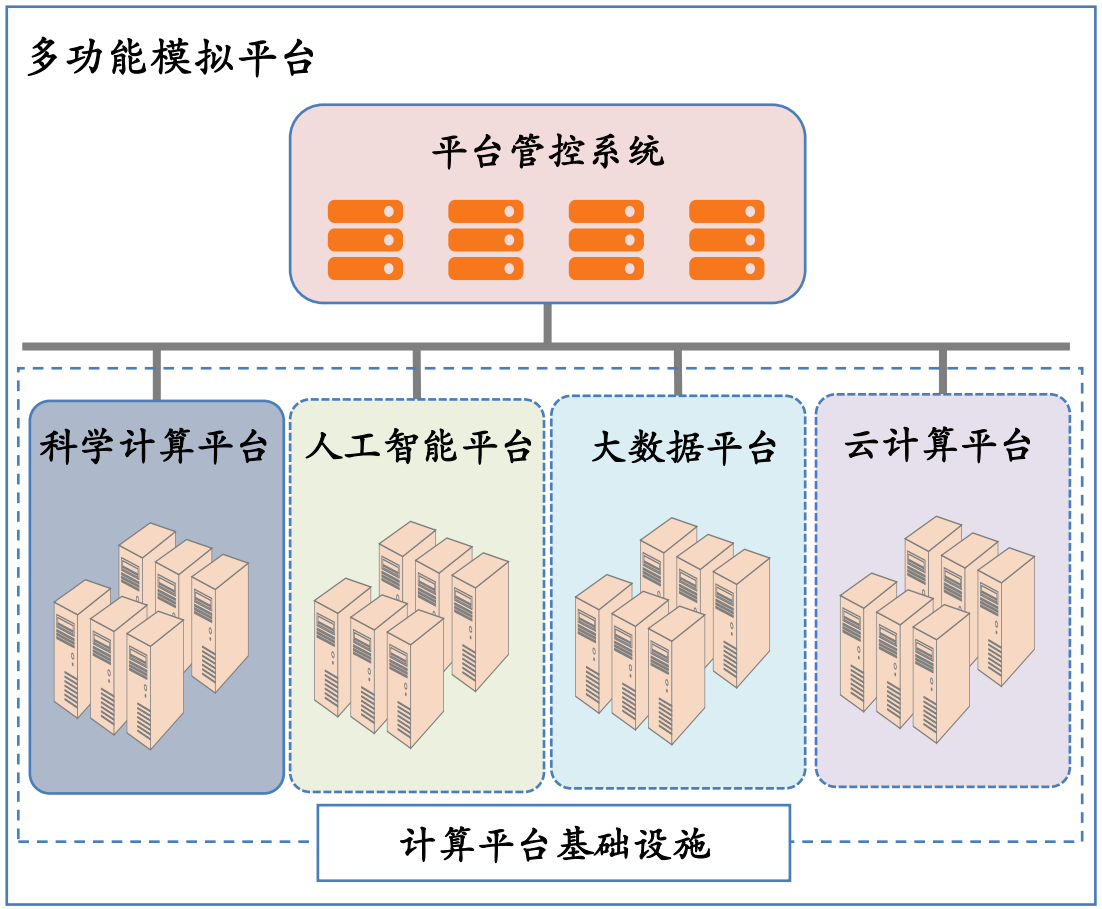

CPU
寄存器堆
PC
ALU
总线接口
系统总线
内存总线
I/O桥
主存
（运行内存）
I/O总线
USB控制器
图形适配器
磁盘控制器
鼠标
键盘
输出给用户
磁盘
知乎 @Yurk


节点
CPU
CPU核心
1线程
2线程（超线程）


时间段T
串行
线程A
线程B
线程C
并行
线程A
线程B
线程C
并发
线程A
线程B
线程C
知乎 @梦迪迪迪迪迪迪

计算中心概况--平台基本结构和资源

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY



总计算资源

约4.3 万核，总计算能力超过 3.5 PFlops，每年可提供 3.7 亿核时和33.2万卡时。

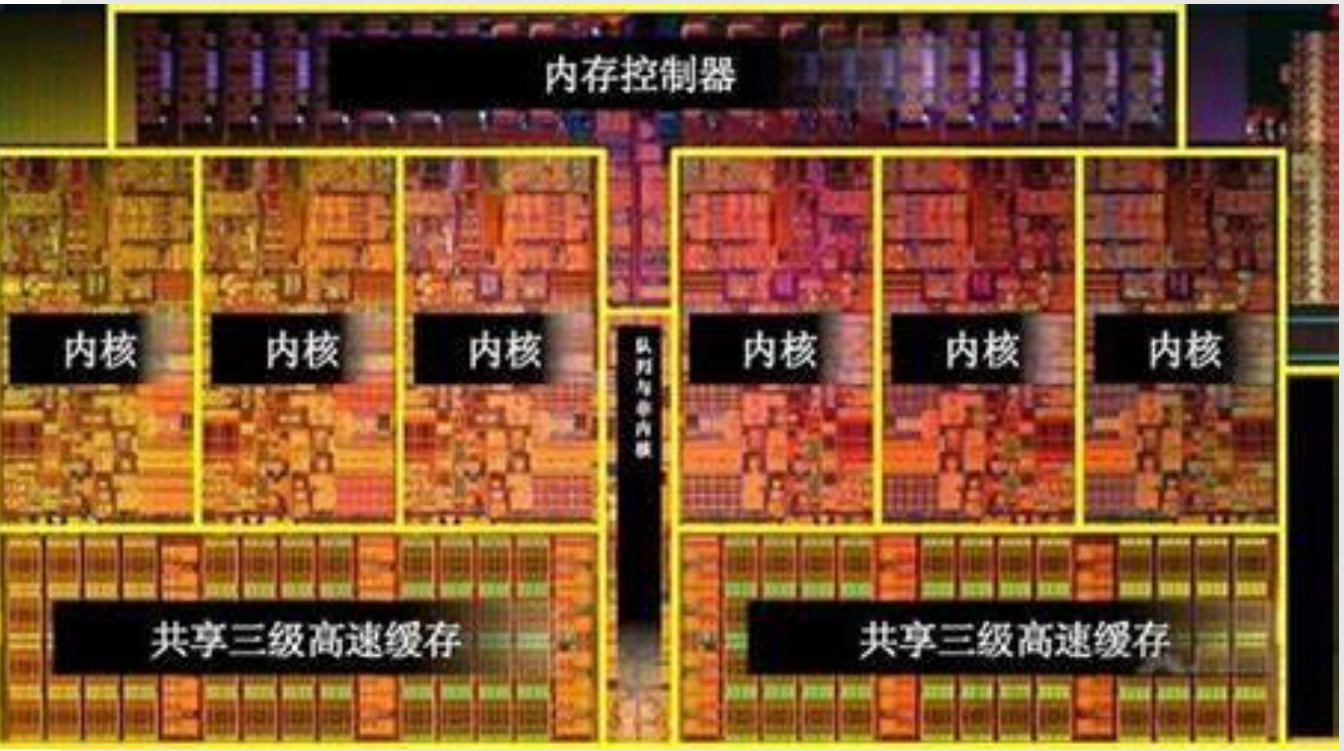
启明

节点类型	数量	单节点配置	核（卡）数	核(卡)时/年
CPU	269	E5-2690v3 / Intel 6338 / AMD 7763 / AMD 7773x	9432 核	8262.43万核时
Large Memory	11	(E7-8880v3, 6 TB) / (Intel 8350c, 2TB)		
GPU	7	(A100 40/80 GB, PCIe, SXM, NV-Bridge) / (V100 32GB, PCIe)	30 卡	26.28万卡时
Storage	1.1 PB	IO ≥ 45 GB/s	-	-

太乙

节点类型	数量	单节点配置	核心数	核时/年
CPU	815 台	Intel 6148, 192 GB	33144 核	29034.14 万核时
S8S	2台	Intel 8160, 6 TB		
GPU	4台	6148, 2*V100 16GB		
Storage	5.5 PB	IO ≥ 40 GB/s	-	-

2.通信--内核间



- 在多核CPU中，内核之间的通信通常是通过共享内存或者特定的通信通道（例如缓存一致性协议）来实现的。以下是一些常见的方式：

共享内存： 多核CPU通常共享一片物理内存。每个内核都可以访问这个内存区域，从而实现内核之间的通信。通信的方式包括直接读写共享变量，或者通过共享数据结构来传递信息。然而，需要注意的是，共享内存的并发访问可能导致数据竞争和一致性问题，因此需要使用同步机制来确保数据的一致性。

缓存一致性协议： 当多个内核拥有各自的缓存时，缓存一致性协议被用来保持各个核的缓存中的数据一致。当一个内核修改了共享数据时，它会通知其他内核相关的缓存行无效，使得其他核在下次访问这个数据时会从主内存重新获取。常见的缓存一致性协议包括MESI（Modified, Exclusive, Shared, Invalid）。

原子操作和锁： 内核之间的通信还可以通过原子操作和锁来实现。原子操作是不可分割的操作，通常用于对共享变量的操作。锁则用于确保在某一时刻只有一个内核能够访问共享资源，以防止数据竞争。

消息传递： 在一些多核系统中，内核之间的通信也可以通过消息传递来实现。这意味着一个内核可以通过发送消息给其他内核来传递信息。这样的通信模型通常被用于分布式内存系统，其中各个内核可能拥有自己的物理内存。

Cache-Conscious Algorithms

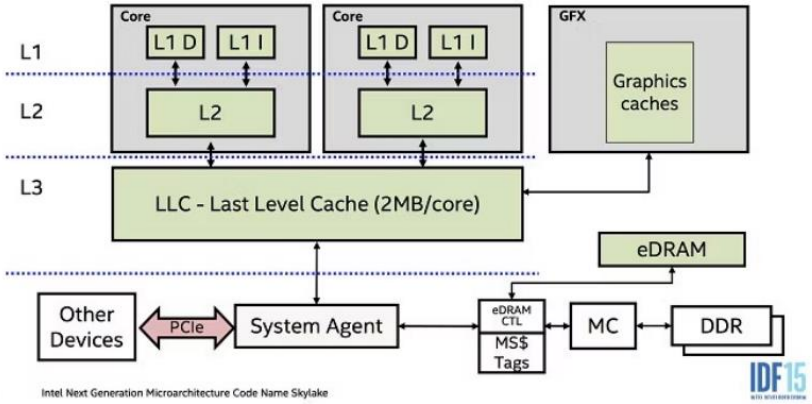
by ysq

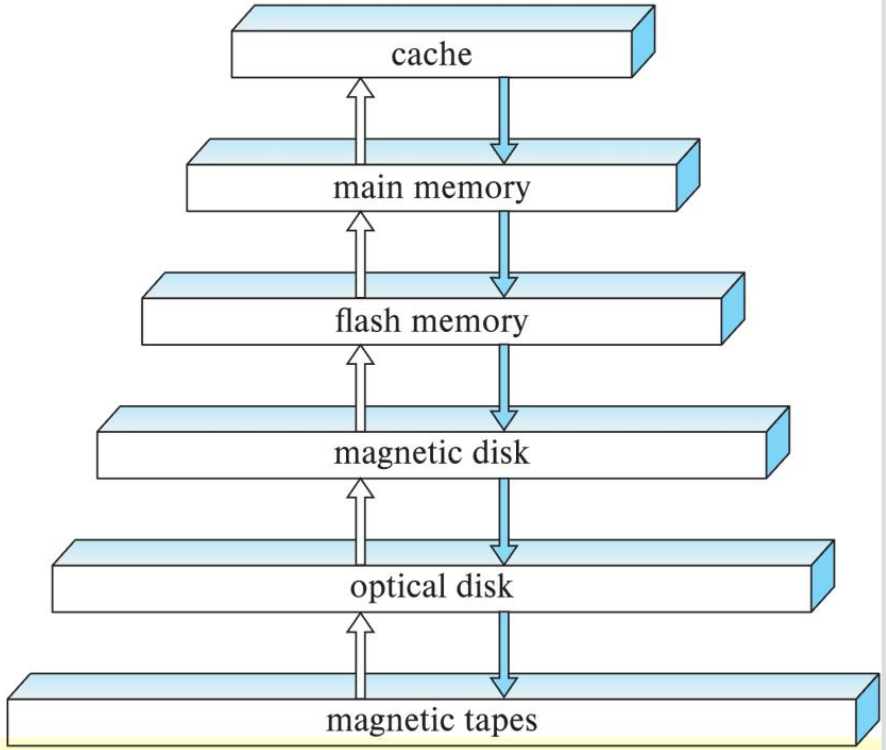
Cache, 100x faster than the main memory

13th Generation Intel® Core™ i9 Processor i9-13900K

- L1: 2.1MB, ~1ns
- L2: 32MB, ~5ns
- L3: 36MB, ~50ns

Main memory: 1-256GB, ~100ns or more

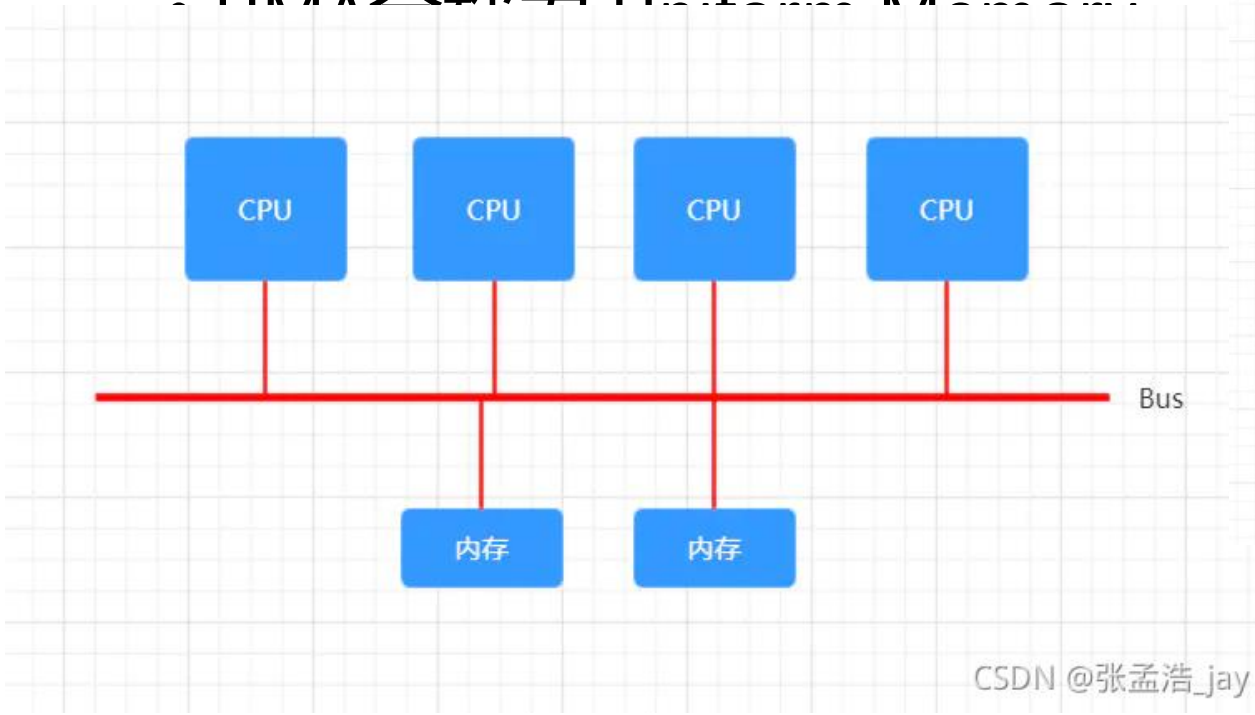


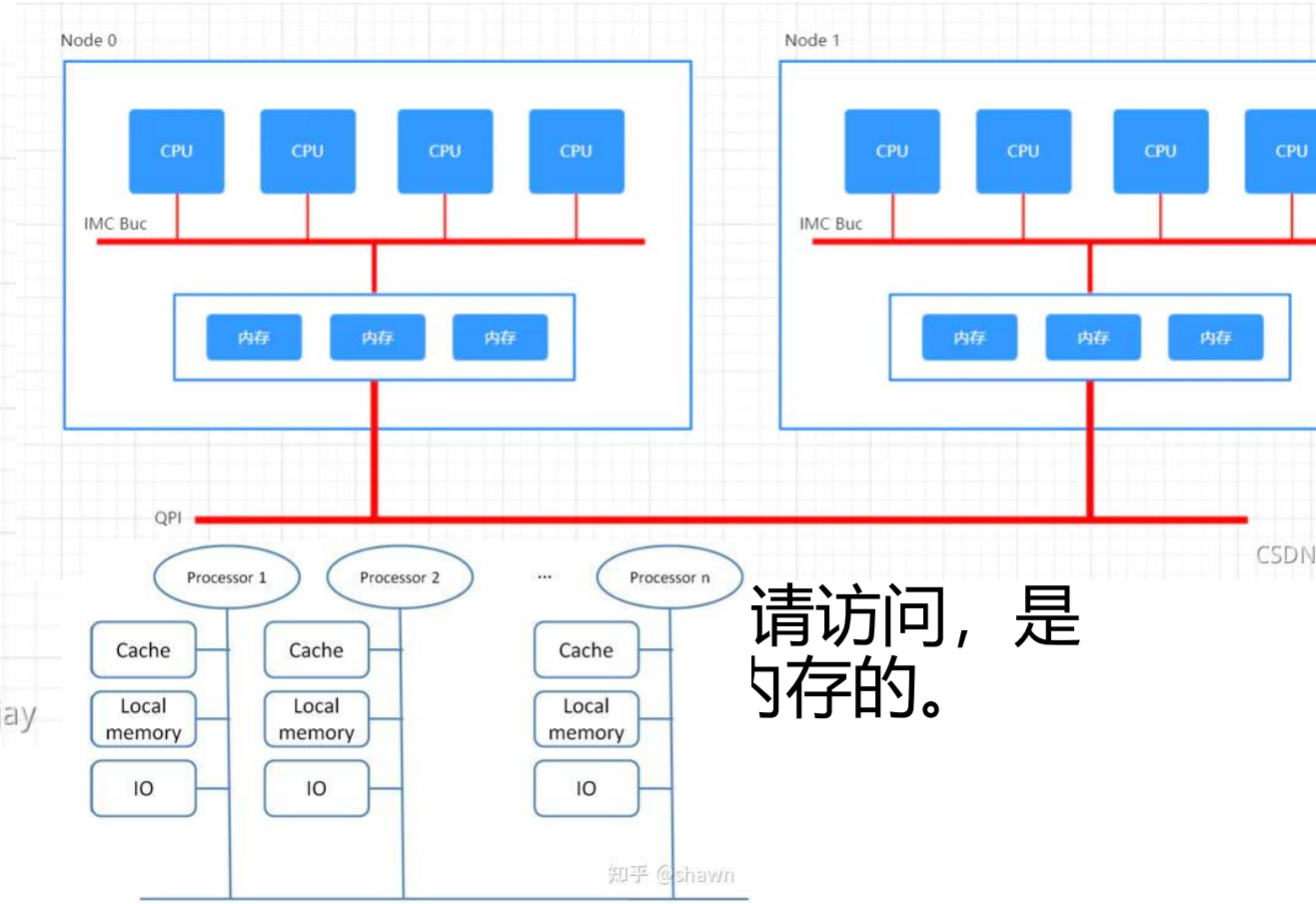


通信--CPU间

UMA结构

NUMA



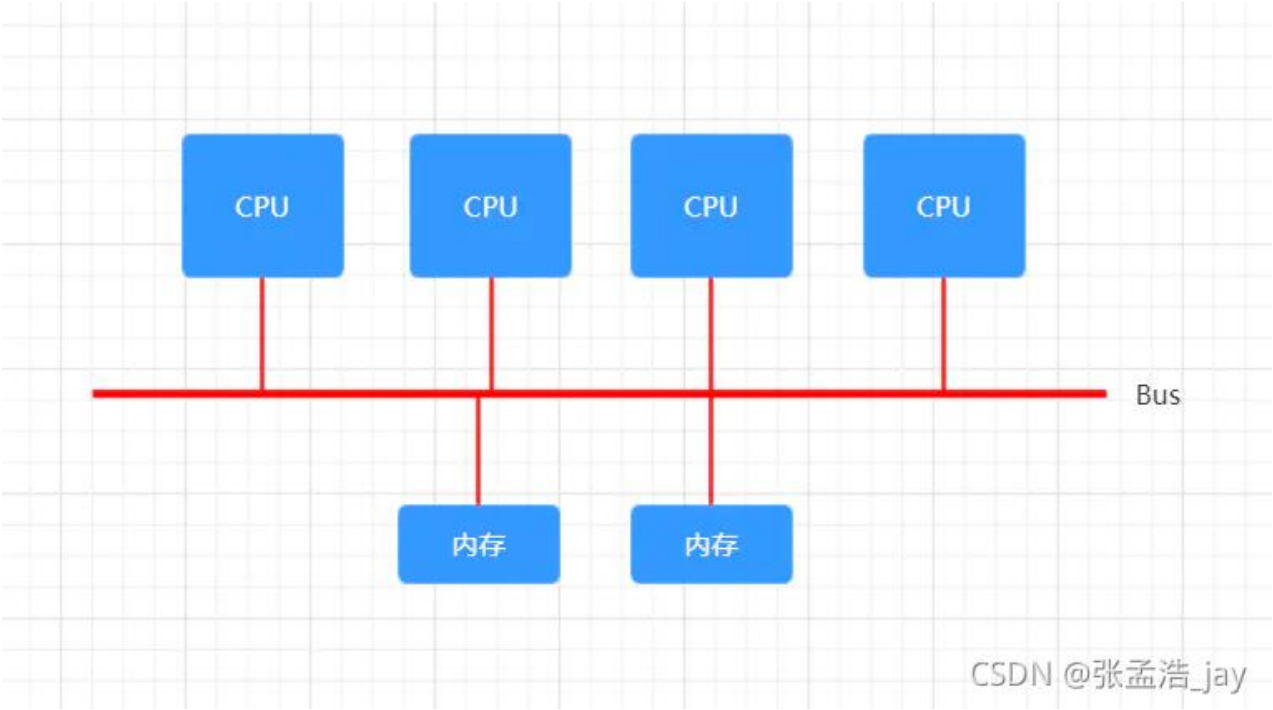


请访问，是
内存的。

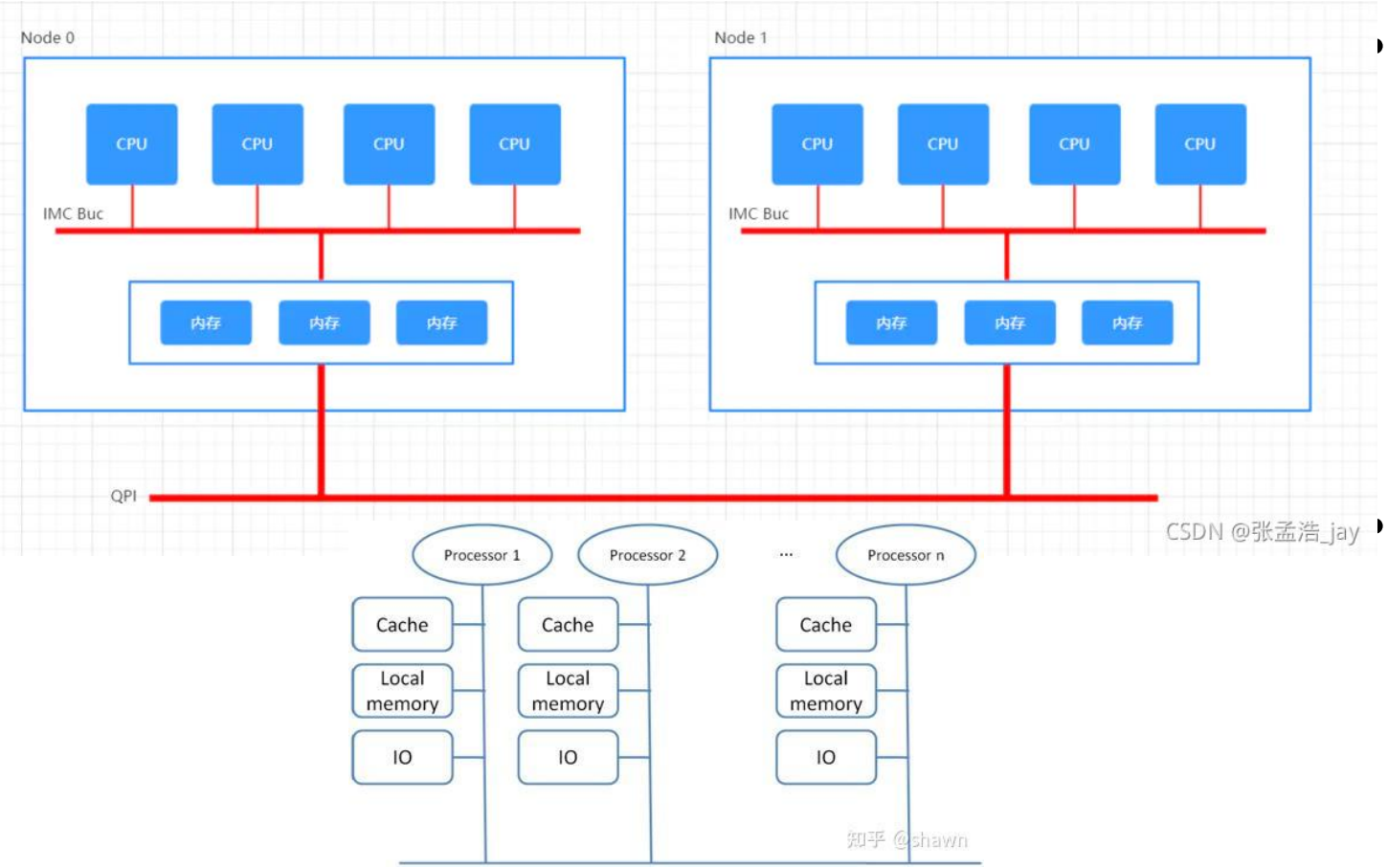
通信--CPU间

UMA结构

- UMA全称为 Uniform Memory Access，叫做一致性内存访问
- 多个CPU通过同一根总线来访问内存。无论多个CPU是访问内存的不同内存单元还是相同的内存单元，同一时刻，只有一个CPU能够访问内存。
- CPU之间通过总线串行的访问内存，所以会出现访问瓶颈
- 多见于个人电脑



通信--CPU间

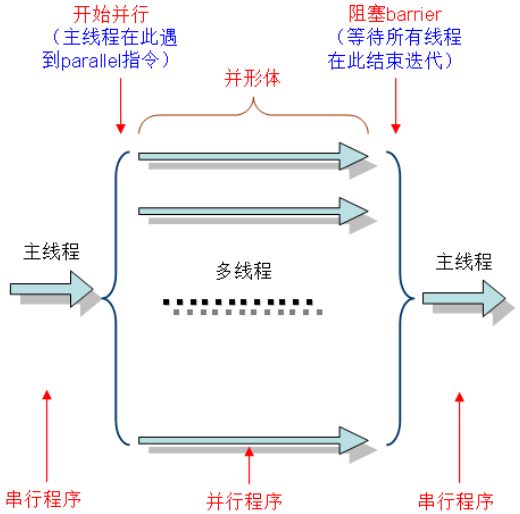


NUMA

- Non-Uniform Memory Access ,非一致性内存访问。每个CPU都分配了一块内存，这样的话，多个CPU可以同时并行访问各自的内存，这样的话，读写内存的效率就上来了。
- 但是当CPU读取其他CPU的内存的时候，需要通过QPI申请访问，是要慢于直接访问本地内存的。

通信实现：OpenMP

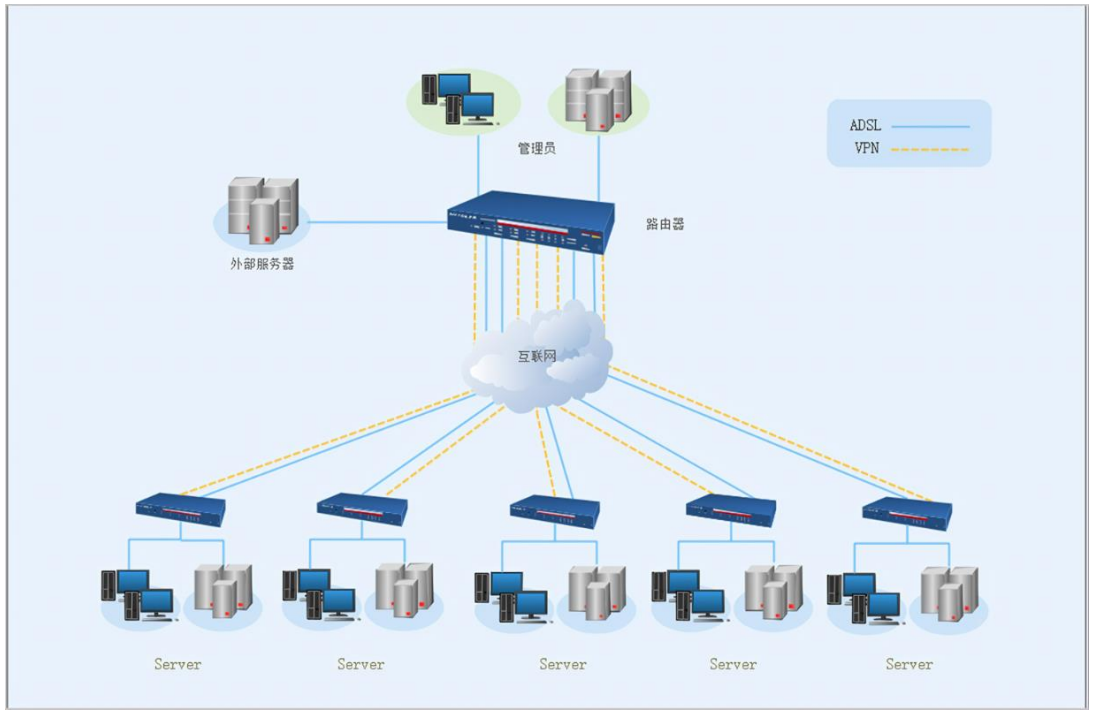
- OpenMP是一种用于共享内存并行系统的多线程程序设计方案，支持的编程语言包括C、C++和Fortran。OpenMP提供了对并行算法的高层抽象描述，特别适合在多核CPU机器上的并行程序设计。



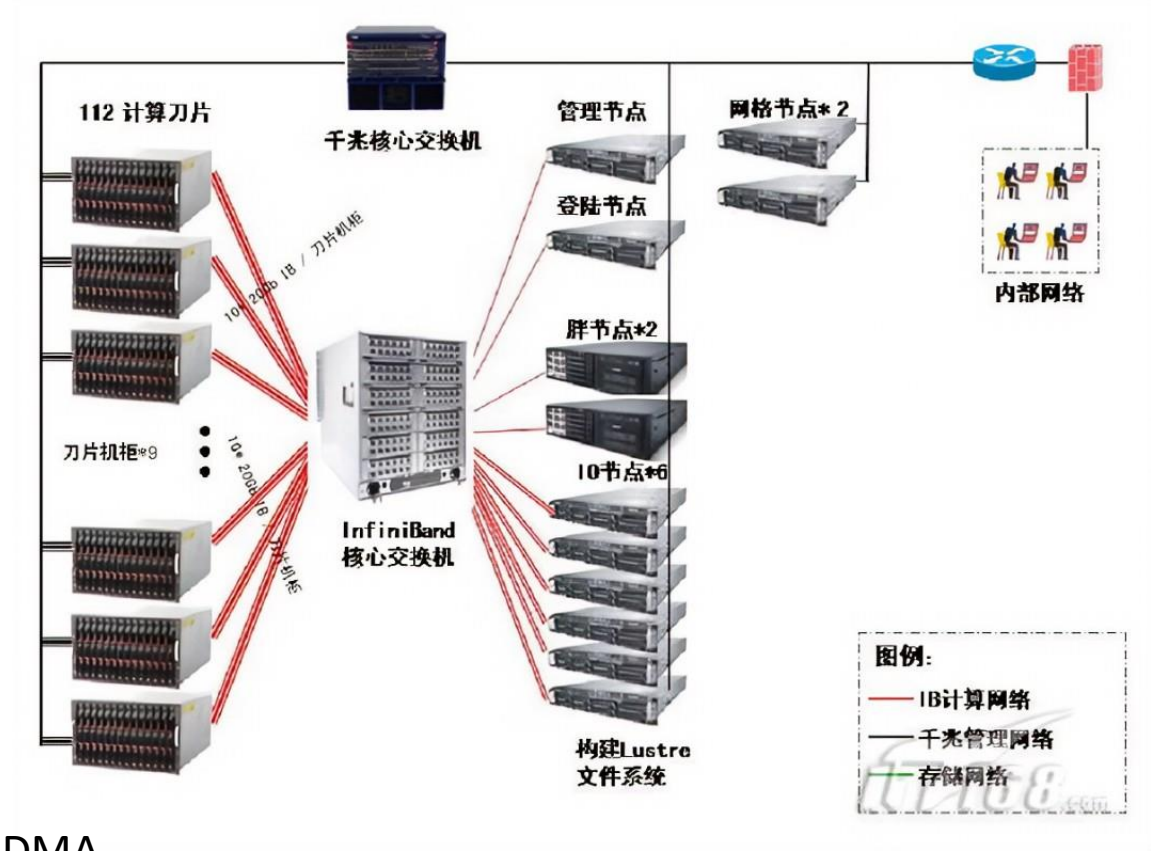
-openmp

通信：节点间

远程集群：局域网络



本地集群：交换机/连接线



RDMA

节点通信实现：MPI

```
#
# ----------------------------------------------------------------------
# - Message Passing library (MPI) ------------------------------------
# ----------------------------------------------------------------------
# MPinc tells the  C  compiler where to find the Message Passing library
# header files,  MPlib  is defined  to be the name of  the library to be
# used. The variable MPdir is only used for defining MPinc and MPlib.
#
MPdir        = /work/share/intel/oneapi/mpi/latest
MPinc        = -I$(MPdir)/include
MPlib        = $(MPdir)/lib/libmpicxx.a
#
```

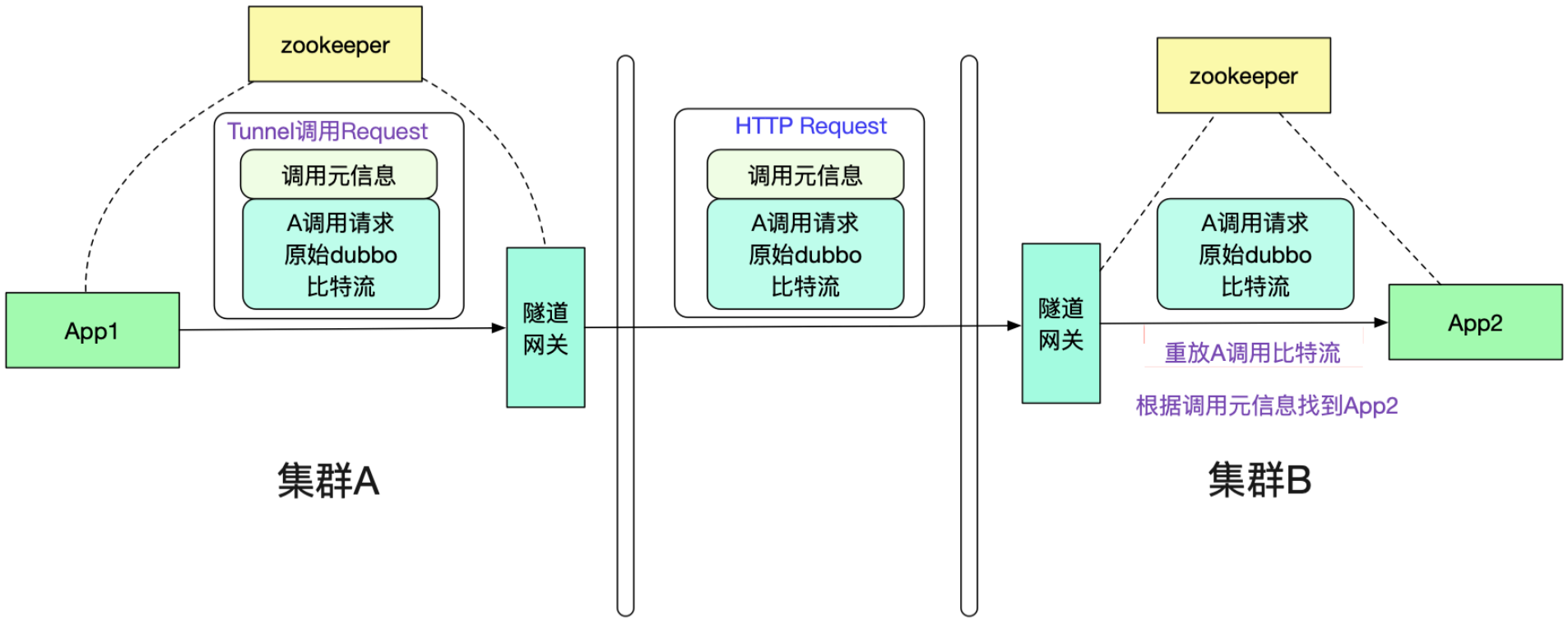
- MPI（Message Passing Interface）是一种用于编写并行程序的标准通信协议和库。它定义了一组规范，使得在多个处理器之间进行消息传递变得更加容易。MPI主要用于并行计算环境，如超级计算机集群，其中多个独立的计算节点需要协同工作以解决大规模的计算问题。
- MPI的作用包括：

- 1.消息传递： MPI 提供了一套标准的消息传递操作，允许不同的处理器在执行过程中相互交换信息。这种消息传递模型对于分布式内存环境非常重要，因为在这种环境中，各个处理器拥有自己的本地内存，而需要通过消息传递来共享数据。
- 2.进程间同步： MPI 提供了各种同步操作，使得不同处理器上运行的进程能够协同工作，同步地执行任务。这对于确保程序正确性和避免竞争条件非常重要。
- 3.集合通信： MPI 提供了一系列的集合通信操作，例如广播（broadcast）、归约（reduce）和散射（scatter）等，以便更方便地在处理器之间交换数据。
- 4.进程管理： MPI 不仅用于通信，还可以用于管理并行计算环境中的进程。它包括了一些用于创建、结束和管理进程的函数。
- 5.跨平台性： MPI是一个跨平台的标准，支持多种计算机体系结构和操作系统。这使得使用MPI编写的并行程序在不同的硬件和操作系统上都能够运行。

- MPI的一个主要优势是其灵活性和可扩展性，使得开发人员能够更容易地将其应用于各种并行计算环境。MPI的实现通常包括在编译时链接的库，而且有多个MPI库的实现可供选择，如Open MPI、MPICH等。

通信：集群间--局域网络

- 我们不会用到这么庞大的东西



Neil Gunther's

Universal Law of Computational Scalability

Relative capacity C(*N*) of a computational platform:

$$C_{(N)} = \frac{N}{1 + \alpha\,(N-1) + \beta\,N\,(N-1)}$$

$0 <= \alpha, \beta < 1$

α Level of contention

β Coherency delay (latency for data to become consistent)

www.perfdynamics.com

The problem with clustering is that it follows a law of diminishing returns: adding a second server will less than double your capacity, and in practice people have clusters of 2, 3 or 4 machines at most (Neil Gunther is a famous Australian consultant/academic specializing on performance)

3.指令集:CPU在运行什么?

序号	架构	特点
1	X86	英特尔和AMD的“专属”，在PC市场上独霸多年，地位不可撼动
2	ARM	在移动端和便捷设备上有着不可替代的优势
3	MIPS	在网关、机顶盒等市场上非常受欢迎
4	RISC-V	虽然出来不久，但在智能穿戴产品上的应用广泛，前景广阔

- 指令集，就是CPU中用来计算和控制计算机系统的一套指令的集合，而每一种新型的CPU在设计时就规定了一系列与其他硬件电路相配合的指令系统。而指令集的先进与否，也关系到CPU的性能发挥，它也是CPU性能体现的一个重要标志。
- 从CPU发明到现在，有非常多种架构，从我们熟悉的X86、ARM，到不太熟悉的RISC-V，MIPS、IA64，它们之间的差距都非常大。但是如果从最基本的逻辑角度来分类的话，它们可以被分为两大类，即所谓的“复杂指令集”与“精简指令集”系统，也就是经常看到的“CISC”与“RISC”。

- 高级语言 C/C++...
- 汇编语言
- 指令集 RISC-V...
- 处理器架构 √

Open RISC-V Reference Card ①

Base Integer Instructions: RV32I and RV64I

Category	Name	Fmt	RV32I Base	+RV64I
Shifts	Shift Left Logical	R	SLL rd,rs1,rs2	SLLW rd,rs1,rs2
	Shift Left Log. Imm.	I	SLLI rd,rs1,shamt	SLLIW rd,rs1,shamt
	Shift Right Logical	R	SRL rd,rs1,rs2	SRLW rd,rs1,rs2
	Shift Right Log. Imm.	I	SRLI rd,rs1,shamt	SRLIW rd,rs1,shamt
	Shift Right Arithmetic	R	SRA rd,rs1,rs2	SRAW rd,rs1,rs2
	Shift Right Arith. Imm.	I	SRAI rd,rs1,shamt	SRAIW rd,rs1,shamt
Arithmetic	ADD	R	ADD rd,rs1,rs2	ADDW rd,rs1,rs2
	ADD Immediate	I	ADDI rd,rs1,imm	ADDIW rd,rs1,imm
	SUBtract	R	SUB rd,rs1,rs2	SUBW rd,rs1,rs2
	Load Upper Imm	U	LUI rd,imm	
	Add Upper Imm to PC	U	AUIPC rd,imm	
Logical	XOR	R	XOR rd,rs1,rs2	
	XOR Immediate	I	XORI rd,rs1,imm	
	OR	R	OR rd,rs1,rs2	
	OR Immediate	I	ORI rd,rs1,imm	
	AND	R	AND rd,rs1,rs2	
	AND Immediate	I	ANDI rd,rs1,imm	
Compare	Set <	R	SLT rd,rs1,rs2	
	Set < Immediate	I	SLTI rd,rs1,imm	
	Set < Unsigned	R	SLTU rd,rs1,rs2	
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm	
Branches	Branch =	B	BEQ rs1,rs2,imm	
	Branch ≠	B	BNE rs1,rs2,imm	
	Branch <	B	BLT rs1,rs2,imm	
	Branch ≥	B	BGE rs1,rs2,imm	
	Branch < Unsigned	B	BLTU rs1,rs2,imm	
	Branch ≥ Unsigned	B	BGEU rs1,rs2,imm	
Jump & Link	J&L	J	JAL rd,imm	
	Jump & Link Register	I	JALR rd,rs1,imm	
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
Environment	CALL	I	ECALL	
	BREAK	I	EBREAK	

RV Privileged Instructions

Category	Name	Fmt	RV mnemonic
Trap	Mach-mode trap return	R	MRET
	Supervisor-mode trap return	R	SRET
Interrupt	Wait for Interrupt	R	WFI
MMU	Virtual Memory FENCE	R	SFENCE.VMA rs1,rs2

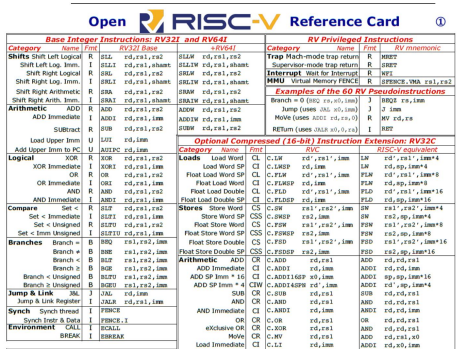
Examples of the 60 RV Pseudoinstructions

Branch = 0 (BEQ rs,x0,imm)	J	BEQZ rs,imm
Jump (uses JAL x0,imm)	J	J imm
MoVe (uses ADDI rd,rs,0)	R	MV rd,rs
RETurn (uses JALR x0,0,ra)	I	RET

Optional Compressed (16-bit) Instruction Extension: RV32C

Category	Name	Fmt	RVC	RISC-V equivalent
Loads	Load Word	CL	C.LW rd',rs1',imm	LW rd',rs1',imm*4
	Load Word SP	CI	C.LWSP rd,imm	LW rd,sp,imm*4
	Float Load Word SP	CL	C.FLW rd',rs1',imm	FLW rd',rs1',imm*8
	Float Load Word	CI	C.FLWSP rd,imm	FLW rd,sp,imm*8
	Float Load Double	CL	C.FLD rd',rs1',imm	FLD rd',rs1',imm*16
	Float Load Double SP	CI	C.FLDSP rd,imm	FLD rd,sp,imm*16
Stores	Store Word	CS	C.SW rs1',rs2',imm	SW rs1',rs2',imm*4
	Store Word SP	CSS	C.SWSP rs2,imm	SW rs2,sp,imm*4
	Float Store Word	CS	C.FSW rs1',rs2',imm	FSW rs1',rs2',imm*8
	Float Store Word SP	CSS	C.FSWSP rs2,imm	FSW rs2,sp,imm*8
	Float Store Double	CS	C.FSD rs1',rs2',imm	FSD rs1',rs2',imm*16
	Float Store Double SP	CSS	C.FSDSP rs2,imm	FSD rs2,sp,imm*16
Arithmetic	ADD	CR	C.ADD rd,rs1	ADD rd,rd,rs1
	ADD Immediate	CI	C.ADDI rd,imm	ADDI rd,rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP x0,imm	ADDI sp,sp,imm*16
	ADD SP Imm * 4	CIW	C.ADDI4SPN rd',imm	ADDI rd',sp,imm*4
	SUB	CR	C.SUB rd,rs1	SUB rd,rd,rs1
	AND	CR	C.AND rd,rs1	AND rd,rd,rs1
	AND Immediate	CI	C.ANDI rd,imm	ANDI rd,rd,imm
	OR	CR	C.OR rd,rs1	OR rd,rd,rs1
	eXclusive OR	CR	C.XOR rd,rs1	AND rd,rd,rs1
	MoVe	CR	C.MV rd,rs1	ADD rd,rs1,x0
	Load Immediate	CI	C.LI rd,imm	ADDI rd,x0,imm

- 高级语言 C/C++...
- 汇编语言：
- 指令集 RISC-V...
- 处理器架构 √



Open RISC-V Reference Card ①

Base Integer Instructions: RV32I and RV64I

Category	Name	Fmt	RV32I Base	+RV64I
Shifts	Shift Left Logical	R	SLL rd,rs1,rs2	SLLW rd,rs1,rs2
	Shift Left Log. Imm.	I	SLLI rd,rs1,shamt	SLLIW rd,rs1,shamt
	Shift Right Logical	R	SRL rd,rs1,rs2	SRLW rd,rs1,rs2
	Shift Right Log. Imm.	I	SRLI rd,rs1,shamt	SRLIW rd,rs1,shamt
	Shift Right Arithmetic	R	SRA rd,rs1,rs2	SRAW rd,rs1,rs2
	Shift Right Arith. Imm.	I	SRAI rd,rs1,shamt	SRAIW rd,rs1,shamt
Arithmetic	ADD	R	ADD rd,rs1,rs2	ADDW rd,rs1,rs2
	ADD Immediate	I	ADDI rd,rs1,imm	ADDIW rd,rs1,imm
	SUBtract	R	SUB rd,rs1,rs2	SUBW rd,rs1,rs2
	Load Upper Imm	U	LUI rd,imm	
	Add Upper Imm to PC	U	AUIPC rd,imm	
Logical	XOR	R	XOR rd,rs1,rs2	
	XOR Immediate	I	XORI rd,rs1,imm	
	OR	R	OR rd,rs1,rs2	
	OR Immediate	I	ORI rd,rs1,imm	
	AND	R	AND rd,rs1,rs2	
	AND Immediate	I	ANDI rd,rs1,imm	
Compare	Set <	R	SLT rd,rs1,rs2	
	Set < Immediate	I	SLTI rd,rs1,imm	
	Set < Unsigned	R	SLTU rd,rs1,rs2	
	Set < Imm Unsigned	I	SLTIU rd,rs1,imm	
Branches	Branch =	B	BEQ rs1,rs2,imm	
	Branch ≠	B	BNE rs1,rs2,imm	
	Branch <	B	BLT rs1,rs2,imm	
	Branch ≥	B	BGE rs1,rs2,imm	
	Branch < Unsigned	B	BLTU rs1,rs2,imm	
	Branch ≥ Unsigned	B	BGEU rs1,rs2,imm	
Jump & Link	J&L	J	JAL rd,imm	
	Jump & Link Register	I	JALR rd,rs1,imm	
Synch	Synch thread	I	FENCE	
	Synch Instr & Data	I	FENCE.I	
Environment	CALL	I	ECALL	
	BREAK	I	EBREAK	

RV Privileged Instructions

Category	Name	Fmt	RV mnemonic
Trap	Mach-mode trap return	R	MRET
	Supervisor-mode trap return	R	SRET
Interrupt	Wait for Interrupt	R	WFI
MMU	Virtual Memory FENCE	R	SFENCE.VMA rs1,rs2

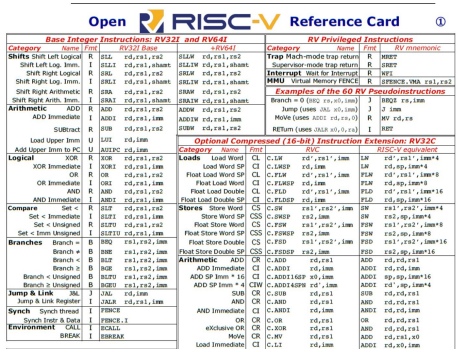
Examples of the 60 RV Pseudoinstructions

Branch = 0 (BEQ rs,x0,imm)	J	BEQZ rs,imm
Jump (uses JAL x0,imm)	J	J imm
MoVe (uses ADDI rd,rs,0)	R	MV rd,rs
RETurn (uses JALR x0,0,ra)	I	RET

Optional Compressed (16-bit) Instruction Extension: RV32C

Category	Name	Fmt	RVC	RISC-V equivalent	
Loads	Load Word	CL	C.LW	rd',rs1',imm	LW rd',rs1',imm*4
	Load Word SP	CI	C.LWSP	rd,imm	LW rd,sp,imm*4
	Float Load Word SP	CL	C.FLW	rd',rs1',imm	FLW rd',rs1',imm*8
	Float Load Word	CI	C.FLWSP	rd,imm	FLW rd,sp,imm*8
	Float Load Double	CL	C.FLD	rd',rs1',imm	FLD rd',rs1',imm*16
	Float Load Double SP	CI	C.FLDSP	rd,imm	FLD rd,sp,imm*16
Stores	Store Word	CS	C.SW	rs1',rs2',imm*4	SW rs1',rs2',imm*4
	Store Word SP	CSS	C.SWSP	rs2,imm	SW rs2,sp,imm*4
	Float Store Word	CS	C.FSW	rs1',rs2',imm	FSW rs1',rs2',imm*8
	Float Store Word SP	CSS	C.FSWSP	rs2,imm	FSW rs2,sp,imm*8
	Float Store Double	CS	C.FSD	rs1',rs2',imm	FSD rs1',rs2',imm*16
	Float Store Double SP	CSS	C.FSDSP	rs2,imm	FSD rs2,sp,imm*16
Arithmetic	ADD	CR	C.ADD	rd,rs1	ADD rd,rd,rs1
	ADD Immediate	CI	C.ADDI	rd,imm	ADDI rd,rd,imm
	ADD SP Imm * 16	CI	C.ADDI16SP	x0,imm	ADDI sp,sp,imm*16
	ADD SP Imm * 4	CIW	C.ADDI4SPN	rd',imm	ADDI rd',sp,imm*4
	SUB	CR	C.SUB	rd,rs1	SUB rd,rd,rs1
	AND	CR	C.AND	rd,rs1	AND rd,rd,rs1
	AND Immediate	CI	C.ANDI	rd,imm	ANDI rd,rd,imm
	OR	CR	C.OR	rd,rs1	OR rd,rd,rs1
	eXclusive OR	CR	C.XOR	rd,rs1	XOR rd,rd,rs1
	MoVe	CR	C.MV	rd,rs1	ADD rd,rs1,x0
	Load Immediate	CI	C.LI	rd,imm	ADDI rd,x0,imm

在汇编语言中，用助记符（Mnemonics）代替机器指令的操作码，用地址符号（Symbol）或标号（Label）代替指令或操作数操作数的地址。在不同的设备中，汇编语言对应着不同的机器语言指令集，通过汇编过程转换成机器指令。普遍地说，特定的汇编语言和特定的机器语言指令集是一一对应的,不同平台之间不可直接移植。



- 高级语言 C/C++...

编译器↓

- 汇编语言：
- 指令集 RISC-V...

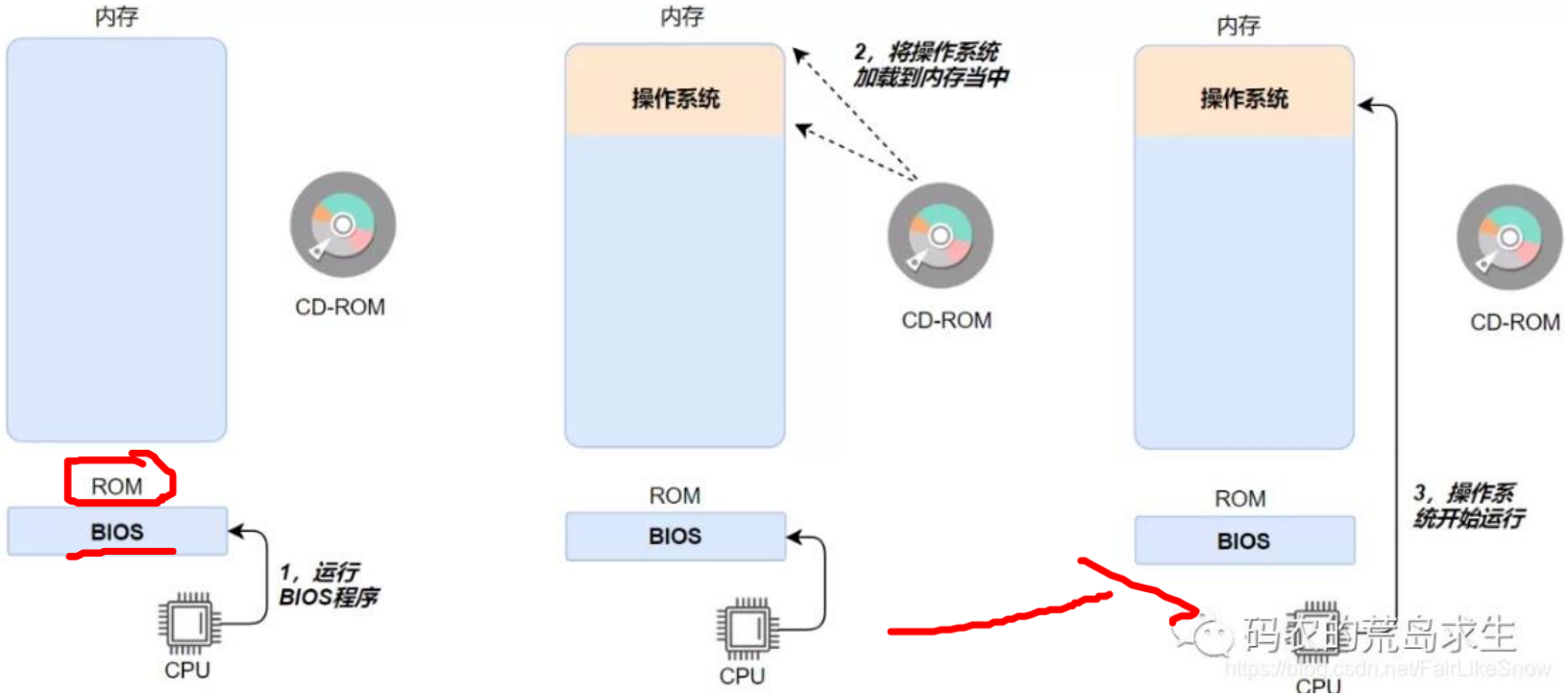
- 处理器架构 √

虽然汇编语言读起来方便了，但也有缺陷。首先汇编语言操作起来还是挺麻烦的。其次汇编语言对应一条条指令集，所以当指令集改变时，就得修改相应汇编语言，导致其可移植性很差，不能跨平台使用，如ARM的汇编语言与Intel X86的就不同。
这时人们就想开发一种更方便操作，超越指令集的语言，于是有了C，C++等高级语言。但处理器只能识别二进制码，那怎么能识别高级语言呢？于是人们开发了编译器，依照如下顺序，将高级语言翻译成二进制码：
高级语言 → 汇编语言 → 二进制机器码。

二次编译思想->面向对象程序设计
字节码中间码思想

3.操作系统：怎么控制CPU跑我们的程序？

- 操作系统其实就是一个大的C程序，本质上和我们自己写的C程序没什么区别，用户程序要想运行起来需要被操作系统从磁盘加载到内存中，那么操作系统是如何运行起来的呢？
- 我们可以自己想一下这个问题，计算机在关机状态下也就是不通电的状态下内存是不能保存内容的，因此一般情况下操作系统和我们的程序一样都是保存在磁盘当中(没有磁盘的计算设备，例如嵌入式设备，会保存在ROM中，稍后会有解释)，那么操作系统要想运行起来必然需要被什么东西加载到内存当中。
- 当我们按下计算机的开机按键后，这时的计算机什么都做不了，因为这时操作系统还没有运行起来，内存中什么都没有。由于现在内存中没有任何程序供CPU执行，因此CPU只能跳转到一种特殊的存储介质中，这种特殊的存储介质就是ROM(Read-Only Memory)，这种存储介质在断电后保存的内容不会丢失，CPU开始执行ROM中保存的程序，这个程序就是大家熟悉的BIOS(Basic Input/Output System)，基本输入输出系统。
- BIOS之所以称之为基本输入输出系统，就是因为BIOS中有管理硬件设备的程序，类似硬件的驱动程序，如果你重装过操作系统使用过BIOS就很容易理解了，你在设置BIOS时就可以使用键盘鼠标来操作，但是请注意此时操作系统还没有运行起来，因此是BIOS来管理的这些硬件设备，当操作系统运行起来后会使用自己的驱动程序来进行设备管理，此时就不再依赖BIOS了。
- BIOS程序运行后就开始在磁盘中搜索操作系统映像(Operating System Image，也就是操作系统)，可以搜索的位置包括floppy disk，磁盘，CD-ROM等，搜索的顺序基于BIOS的设置，这个大家应该比较清楚，我们在重装操作系统时要设置BIOS的搜索顺序。
- 一般情况下像Windows、Linux、MacOS这样的大型操作系统其映像都是放在磁盘当中的(就像我们写程序一样，编译成可执行程序后存放在磁盘当中)。当BIOS找到可以使用的操作系统映像后，开始把操作系统映像加载到内存当中，就好比操作系统把我们的程序从磁盘加载到内存一样，加载完毕后调用操作系统初始化函数，比如启动的是Linux系统，那么调用start_kernel()函数，该函数对Linux内核进行初始化，之后操作系统就运行起来了，因此作为程序员如果你好奇操作系统是如何初始化的，那么该阶段完成的就是操作系统的初始化任务。
- ——————————————————
-
- 原文链接：https://blog.csdn.net/FairLikeSnow/article/details/115606890



BIOS 通过指定好的路径导入操作系统

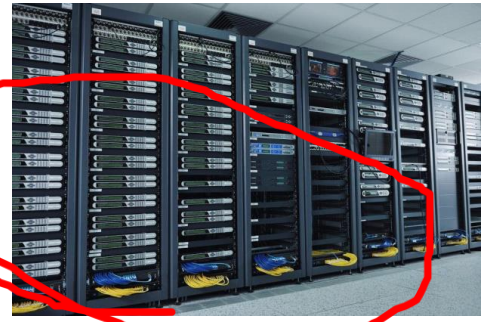
某些病毒：把路径改入自己的路径

操作系统会做什么？

- 操作系统主要包括以下几个方面的功能：
- ①进程管理，其工作主要是进程调度，在单用户单任务的情况下，处理器仅为一个用户的一个任务所独占， 进程管理的工作十分简单。但在多道程序或多用户的情况 下，组织多个作业或任务时，就要解决处理器的调度、 分配和回收等问题 。
- ②存储管理分为几种功能：存储分配、存储共享、存储保护 、存储扩张。
- ③设备管理分有以下功能：设备分配、设备传输控制 、设备独立性。
- ④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。
- ⑤作业管理是负责处理用户提交的任何要求。

计算单元里每个单元都是一个操作系统

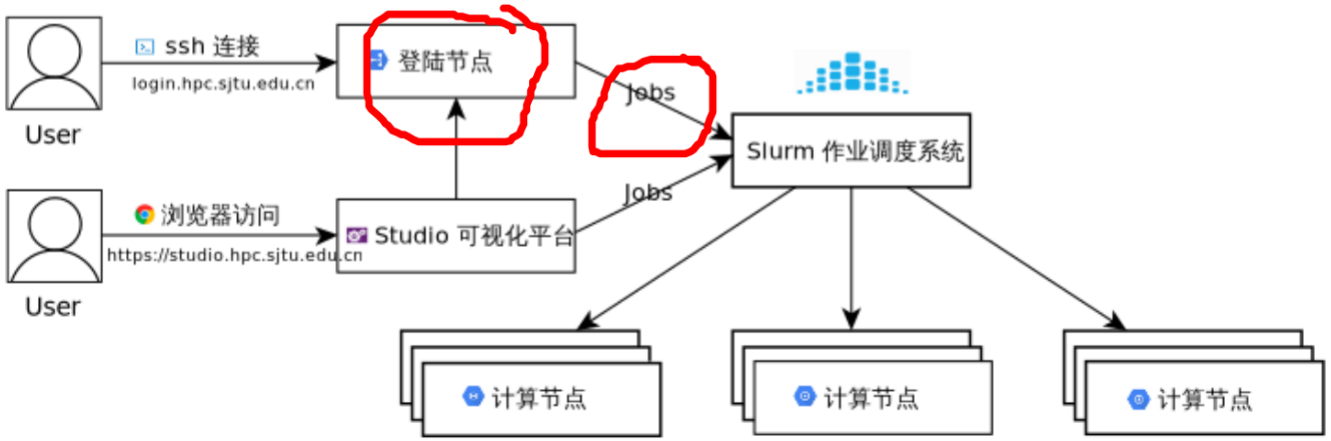
4.作业调度系统



SLURM （Simple Linux Utility for Resource Management）是一种可扩展的工作负载管理器，已被全世界的国家超级计算机中心广泛采用。 它是免费且开源的，根据GPL通用公共许可证发行。

本文档将协助您通过 Slurm 管理作业。 在这里可以找到更多的工作样本。

如果我们可以提供任何帮助，请随时联系 HPC 邮箱。



启明lsf作业调度系统

科学与工程计算中心
Center for Computational Science and Engineering

超级计算机

使用手册

更新时间：2023 年 3月

2. 172.18.6.10 (ssc-iscc)

```
#!/bin/bash
#BSUB -J HPCG-GPU-xiaoyc
#BSUB -q 4a100-40
#BSUB -n 2
#BSUB -e %J.err
#BSUB -o %J.out
#BSUB -gpu "num=1"
#BSUB -R "affinity[core(1)] span[ptile=2]"

# wrong! this is in cpu!!!!!!!!!!!!!!!!!!!!!!
hostfile=`echo $LSB_DJOB_HOSTFILE`
NP=`cat $hostfile | wc -l`

export FAKEROOT=/work/ssc-iscc/zhaojh
source /work/share/intel/oneapi/setvars.sh intel64
module load icc/2022.1.0
module load mpi/2021.6.0
module load mkl/latest
module load compiler/2022.1.0

mpirun -machinefile $hostfile -np $NP $FAKEROOT/hpl-2.3/bin/Linux_Intel64/xhpl
~
~
```

https://hpc.sustech.edu.cn/ref/cluster_User_Manual.pdf

5.环境

```
export FAKEROOT=/work/ssc-iscc/zhaojh
source /work/share/intel/oneapi/setvars.sh intel64
module load icc/2022.1.0
module load mpi/2021.6.0
module load mkl/latest
module load compiler/2022.1.0
```

- 编译环境就是一个你能编译的地方，开发环境就是一个你能开发的地方，测试环境就是一个你能测试的地方。这里环境就是一种其他一切条件就绪，可以让你开发的地方。这个地方可以是一个配置好了网络的实验室，可以是实验室时一台安装了某些软件（IDE及配套软件）的台式机，也可以是操作系统里一个配置好了一些变量（如PATH, LD_LIBRARY_PATH）让你可以执行特定任务的shell。所以环境是大家对完成某种任务所需要一切外部条件的统称。

- 由于用户可能需要使用不同的软件环境，配置不同的环境变量，因而在“太乙”和“启明2.0”上安装了“module软件”来进行管理，用户方便环境变量的设置
- 从而提高移植软件效率。因为”太乙”的cpu为intel Xeon Gold,建议使用Intel编译器。

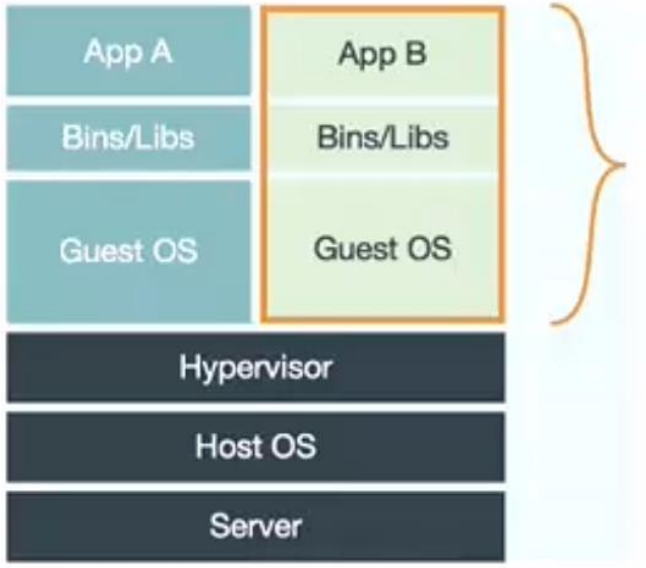
- 如：输入javac命令，本质是调用PATH中java jdk的编译器

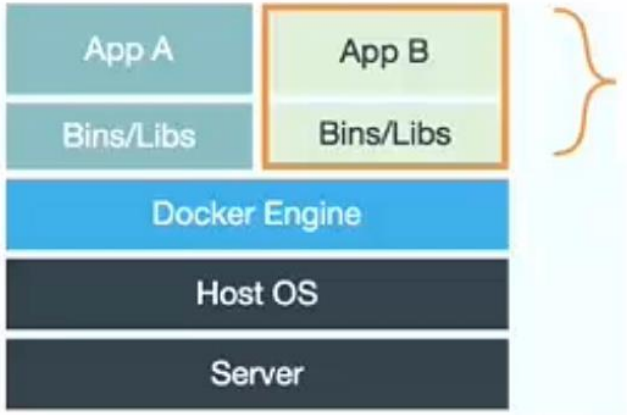
5.容器

容器也是一个应用会占用资源，所以
我们只是简单介绍

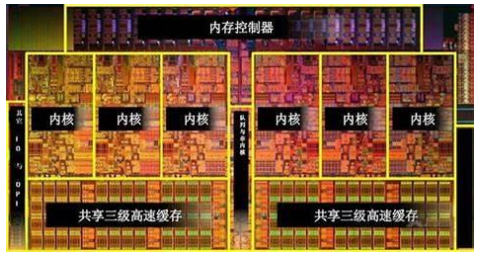
环境的承载器

docker



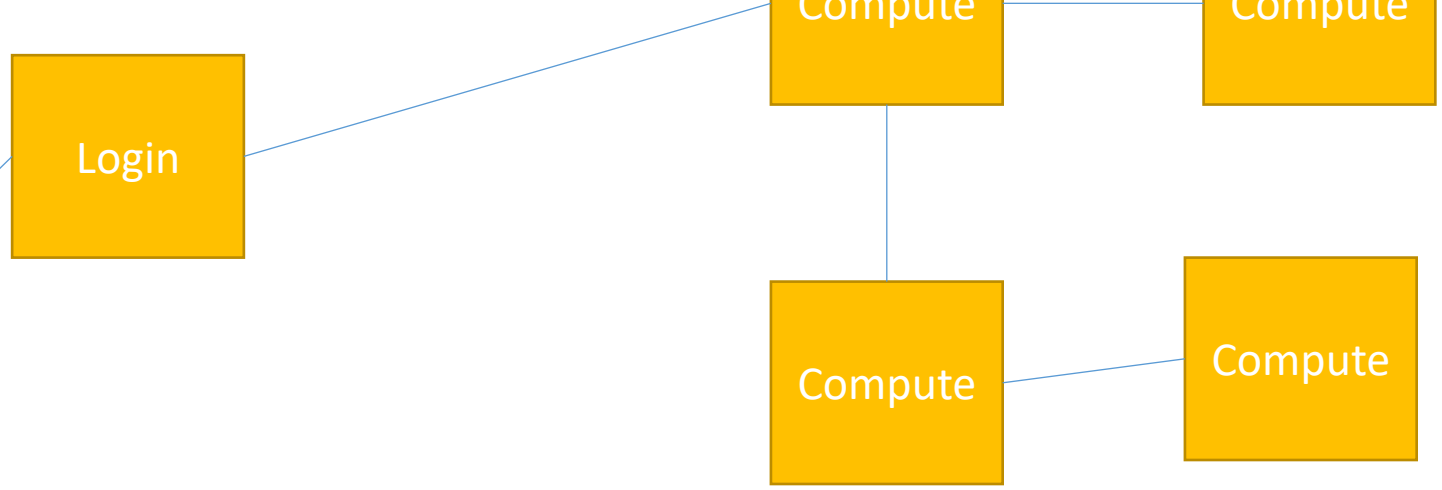



INTEL CORE i5
i5-8500T
SR3XD 2.10GHZ
L834F431


内存控制器
内核
内核
内核
内核
内核
内核
共享三级高速缓存
共享三级高速缓存

集群建设

更好的集群建设
还请学习图论(



2.HPL基础知识

- 1.Top 500
- 2.数学原理
- 3.程序原理
- 4.参数介绍

HPL

- HPL（The High-Performance Linpack Benchmark）是测试高性能计算集群系统浮点性能的基准程序。HPL通过对高性能计算集群采用高斯消元法求解一元N次稠密线性代数方程组的测试，评价高性能计算集群的浮点计算能力。

- 浮点计算峰值是指计算机每秒可以完成的浮点计算次数，包括理论浮点峰值和实测浮点峰值。理论浮点峰值是该计算机理论上每秒可以完成的浮点计算次数，主要由CPU的主频决定。理论浮点峰值 = CPU主频×CPU核数×CPU每周期执行浮点运算的次数。

数学原理

LU 分解

3.1 HPL algorithm introduction

HPL solve non-singular dense linear equation sets by using a Gaussian elimination method. We take a non-singular dense linear equation set

$$Ax = b \tag{3.3}$$

Where A and b are known, x is the vector to be solved.

The basic idea of the Gaussian elimination method is to first perform LU decomposition on the matrix A, where L is the unit lower triangular matrix and U is the non-singular upper triangular matrix [1]. Let $Ux = y$, $Ly = b$, solve $Ux = y$. The decomposition of matrix A is expressed as

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} = LU \tag{3.4}$$

In the formula, A_{ij} and L_{ij} are known, while U_{ij} is to be found $(i, j = 1, 2)$. It can be obtained from formula 3.4 that

$$A_{11} = L_{11}U_{11} \tag{3.5}$$

In formula 3.5, L_{11} and U_{11} are computed by decomposing the principal column elements, so L_{21} and U_{12} can then be obtained by

$$L_{21} = A_{21}U_{11} \tag{3.6}$$

$$U_{12} = A_{12}L_{11}^{-1} \tag{3.7}$$

In addition, formula 3.7 gives us the conclusion that

$$A_{22} - L_{21}U_{12} = L_{22}U_{22} \tag{3.8}$$

It can be seen from the algorithm above that the main operations in HPL are concentrated in matrix operations, which including matrix multiplications and additions, also, its inversions, so we can know that matrix operation is the main factor affecting the performance of HPL, which provides direction for our subsequent optimization.

浮点计算性能测试程序 HPL

HPL 测试通常求解一个稠密线性方程组 $Ax = b$ 所花费的时间来评价计算机的浮点计算性能。为了保证测评结果的公平性，HPL 不允许修改基本算法（采用 LU 分解的高斯消元法），即必须保证总浮点计算次数不变。对 $N \times N$ 的矩阵 A，求解 $Ax = b$ 的总浮点计算次数为 $(2/3 \times N^3 - 2 \times N^2)$。因此，只要给出问题规模 N，测的系统计算时间 T，则 HPL 将测试该系统的浮点性能值为：

$$\frac{\text{计算量：}(2/3 \times N^3 - 2 \times N^2)}{\text{计算时间：}T}$$

HPL using a Gaussian elimination method to solve a system of linear equations. When the problem size is N, the number of floating-point operations is

$$2/3 \times N^3 - 2 \times N^2 \tag{3.1}$$

This value is a certainty, so as long as the problem size N is given and the system computing time T is measured, the performance parameters of the machine can be calculated quantitatively $R_{peak floating-point operations/sec}$

$$R_{peak} = \frac{2/3 \times N^3 - 2 \times N^2}{T} \tag{3.2}$$

On the basis of HPL hotspot function analysis and source code algorithm analysis, our team weighs the key adjustment parameters and formulates a set of evaluation system based on the ranking of parameter influence factors, which is used to find the optimal path for HPL parameter tuning. In addition, we also optimized some of the algorithms and the HPL ported on the GPU platform, which make it more efficient than the average HPL-GPU version.

3.原理

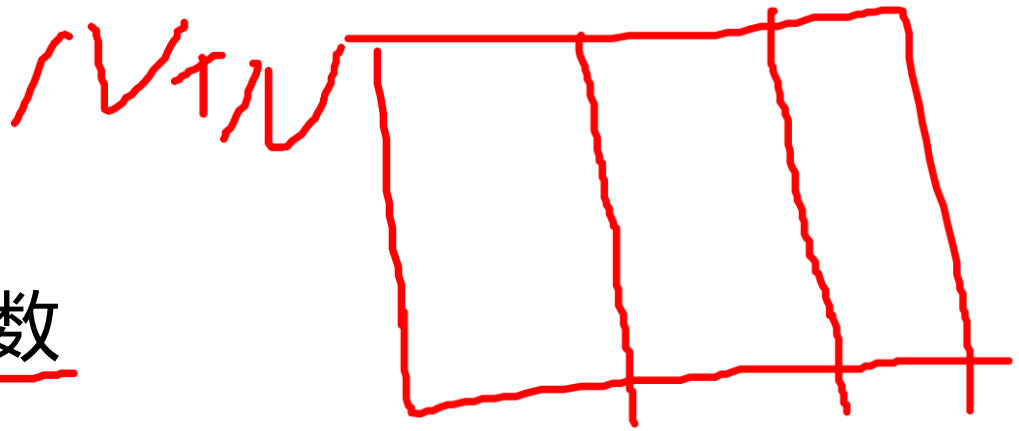
- 安装+编译软件：C compiler Cmake
- 运行：

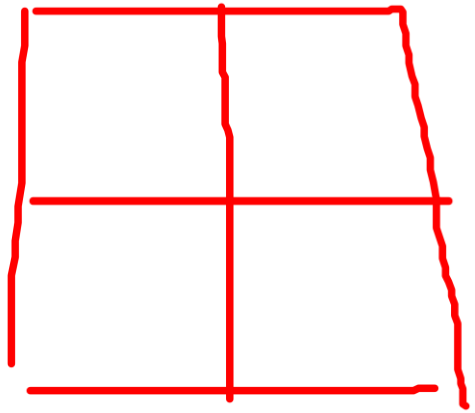
HPL 的核心计算是矩阵乘（耗时通常在 90%以上），矩阵乘法采用**分块算法**实现，其分块会根据CPU的数量、分布由执行者决定。其分块的大小对计算性能影响大，需综合系统 CPU 缓存大小等因素，通过小规模问题的实测，选择最佳的分块矩阵值。

HPL 采用 MPI 进行并行计算，其中计算的进程以二维网格方式分布，需要设定处理器阵列排列方式和网格尺寸，这同样需要小规模数据测定获得最佳方案。

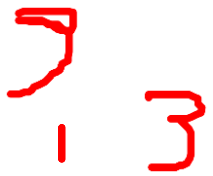
LU 分解参数、MPI、BLAS数学库、编译选项、操作系统等众多其他因素同样对最终测试结果有影响，具体情况需要参考相关文献。

HPL 的安装需要编译器、并行环境 MPI 和基本线性代数函数库（BLAS）支持，其中需要注意 BLAS 库的选择。当前常用的 BLAS 库有： GOTO、OpenBLAS、Atlas、MKL、ACML 等多个版本，不同系统上不同实现的性能可能会有较大差异，需要参照相关文献和实际测试版本。





- N 矩阵的维数
- NBs
- P,Q



- PMAP.....

- P and Q - the number of rows and columns in the process grid, respectively.

  P*Q must be the number of MPI processes that HPL is using.

  Choose P ≤ Q.

- NB - the block size of the data distribution.

  The table below shows recommended values of NB for different Intel® processors:

3.HPL所需库

- MPI通信库
- BLAS数学库
- C编译器

- openMP

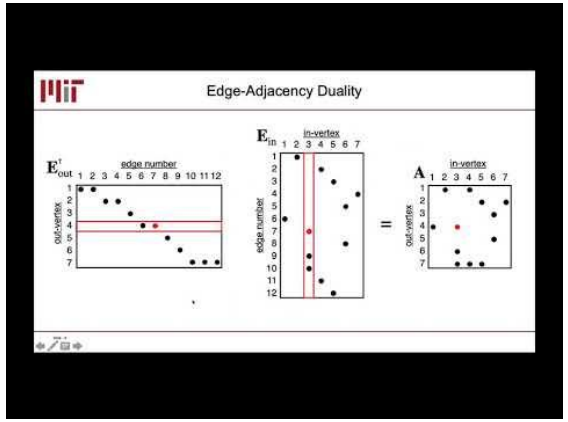
节点通信实现：MPI

```
#
# ----------------------------------------------------------------------
# - Message Passing library (MPI) ------------------------------------
# ----------------------------------------------------------------------
# MPinc tells the  C  compiler where to find the Message Passing library
# header files,  MPlib  is defined  to be the name of  the library to be
# used. The variable MPdir is only used for defining MPinc and MPlib.
#
MPdir        = /work/share/intel/oneapi/mpi/latest
MPinc        = -I$(MPdir)/include
MPlib        = $(MPdir)/lib/libmpicxx.a
#
```

- MPI（Message Passing Interface）是一种用于编写并行程序的标准通信协议和库。它定义了一组规范，使得在多个处理器之间进行消息传递变得更加容易。MPI主要用于并行计算环境，如超级计算机集群，其中多个独立的计算节点需要协同工作以解决大规模的计算问题。
- MPI的作用包括：

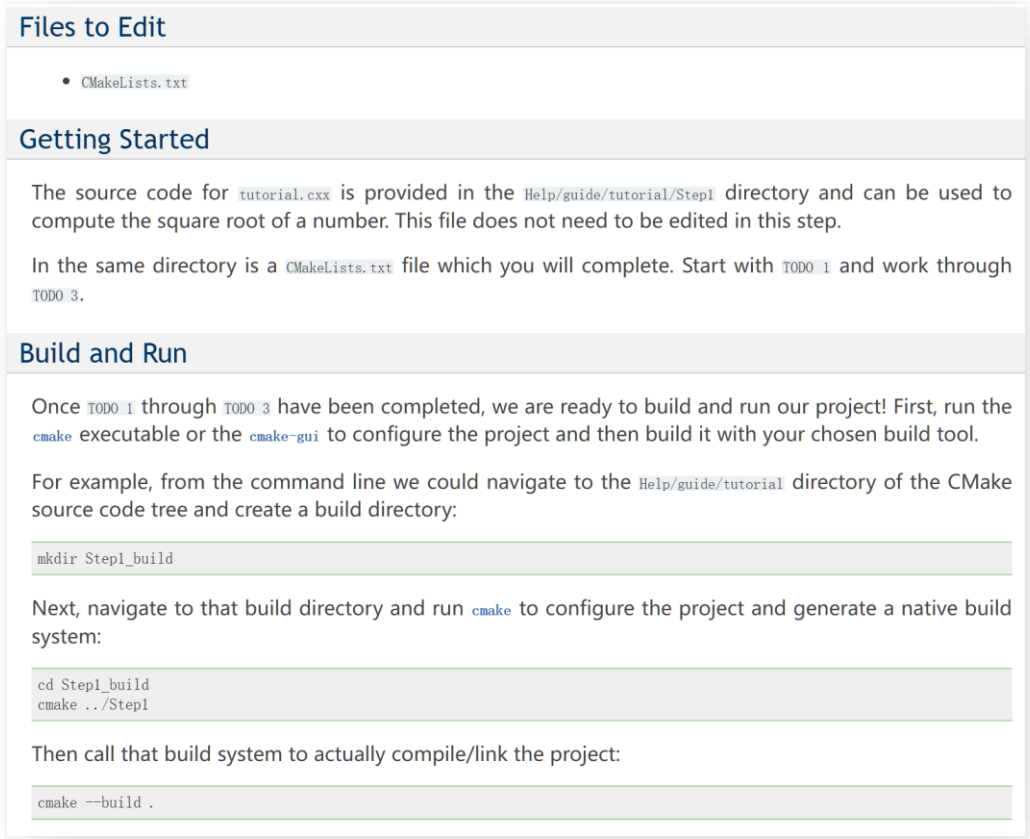
- 1.消息传递： MPI 提供了一套标准的消息传递操作，允许不同的处理器在执行过程中相互交换信息。这种消息传递模型对于分布式内存环境非常重要，因为在这种环境中，各个处理器拥有自己的本地内存，而需要通过消息传递来共享数据。
- 2.进程间同步： MPI 提供了各种同步操作，使得不同处理器上运行的进程能够协同工作，同步地执行任务。这对于确保程序正确性和避免竞争条件非常重要。
- 3.集合通信： MPI 提供了一系列的集合通信操作，例如广播（broadcast）、归约（reduce）和散射（scatter）等，以便更方便地在处理器之间交换数据。
- 4.进程管理： MPI 不仅用于通信，还可以用于管理并行计算环境中的进程。它包括了一些用于创建、结束和管理进程的函数。
- 5.跨平台性： MPI是一个跨平台的标准，支持多种计算机体系结构和操作系统。这使得使用MPI编写的并行程序在不同的硬件和操作系统上都能够运行。

- MPI的一个主要优势是其灵活性和可扩展性，使得开发人员能够更容易地将其应用于各种并行计算环境。MPI的实现通常包括在编译时链接的库，而且有多个MPI库的实现可供选择，如Open MPI、MPICH等。



- BLAS（basic linear algebra subroutine） 是一系列基本线性代数运算函数1的接口（interface）标准． 这里的线性代数运算是指例如矢量的线性组合，矩阵乘以矢量， 矩阵乘以矩阵等． 接口在这里指的是诸如哪个函数名实现什么功能， 有几个输入和输出变量， 分别是什么．
- BLAS 被广泛用于科学计算和工业界， 已成为业界标准． 在更高级的语言和库中， 即使我们不直接使用 BLAS 接口， 它们也是通过调用 BLAS 来实现的（如 Matlab 中的各种矩阵运算）．
- BLAS 原本是用 Fortran 语言写的， 但后来也产生了 C 语言的版本 cBLAS，接口与 Fortran 的略有不同（例如使用指针传递数组）， 但大同小异．
- 注意 BLAS 是一个接口的标准而不是某种具体实现（implementation）． 简单来说， 就是不同的作者可以各自写出不同版本的 BLAS 库， 实现同样的接口和功能， 但每个函数内部的算法可以不同． 这些不同导致了不同版本的 BLAS 在不同机器上运行的速度也不同．

Cmake



Files to Edit

- `CMakeLists.txt`

Getting Started

The source code for `tutorial.cxx` is provided in the `Help/guide/tutorial/Step1` directory and can be used to compute the square root of a number. This file does not need to be edited in this step.

In the same directory is a `CMakeLists.txt` file which you will complete. Start with `TODO 1` and work through `TODO 3`.

Build and Run

Once `TODO 1` through `TODO 3` have been completed, we are ready to build and run our project! First, run the `cmake` executable or the `cmake-gui` to configure the project and then build it with your chosen build tool.

For example, from the command line we could navigate to the `Help/guide/tutorial` directory of the CMake source code tree and create a build directory:

```
mkdir Step1_build
```

Next, navigate to that build directory and run `cmake` to configure the project and generate a native build system:

```
cd Step1_build
cmake ../Step1
```

Then call that build system to actually compile/link the project:

```
cmake --build .
```

- Cmake 是一个构建工具，帮助您为项目生成安装规则。您可以使用官方网站上的预编译二进制文件在 Windows、macOS 和 Linux 平台上安装。安装过程包括提取源文件、创建 build 目录、运行 CMake 配置 build 树并使用选定的 build 工具来构建软件。
- https://cmake.org/cmake/help/latest/guide/tutorial/index.html
- https://cmake.org/getting-started/

4.HPL编译

- Cmake配置：
- 配置好目录

- 环境配置
- make arch=Linux_Intel64

```
Loading compiler-rt/2022.1.0
  ERROR: compiler-rt/2022.1.0 cannot be loaded due to a conflict.
    HINT: Might try "module unload compiler-rt" first.
[ssc-iscc@b04u17l hpl-2.3]$ module list
Currently Loaded Modulefiles:
 1) tbb/latest   2) mpi/latest   3) compiler-rt/latest   4) mkl/latest   5) icc/2022.1.0
[ssc-iscc@b04u17l hpl-2.3]$ ls
AUTHORS    ChangeLog          Make.top     NEWS    TUNING        compile       configure.ac  lib
BUGS       HISTORY            Makefile     README  acinclude.m4  config.guess  depcomp       makes
COPYING    INSTALL            Makefile.am  THANKS  aclocal.m4    config.sub    include       man
COPYRIGHT  Make.Linux_Intel64  Makefile.in  TODO    bin           configure     install-sh    missin
[ssc-iscc@b04u17l hpl-2.3]$ make arch=Linux_Intel64
```

1.Linux_Intel64

2.main
3.mkl
4.mpi
5.cc:202.1

6.
load 202.1
source 202.1

-qopenmp

5.HPL运行与调优

- 1.N
- 2.NBs
- 3.P,Q

看文章

- More:
- 数学库 通信库更替
- PFact等地方更改

- 常用的 HPL 优化策略如下：
- 1.选择尽可能大的 N，在系统内存耗尽之前，N越大，HPL 性能越高。
- 2.选择合适的分块方法，给每个CPU适当的计算量
- 3.修改并行进程的数量
- 最新版的 HPL 可以从 HPL官网 获得

6.进阶：更多HPL学习

上海交大超算平台用户手册 Documentation (sjtu.edu.cn)

- https://docs.hpc.sjtu.edu.cn/index.html

- 超算习堂
- https://www.easyhpc.net/lab/1