第一章
并行计算导论

哈尔滨工业大学
张伟哲

2025, Fall Semester

目录

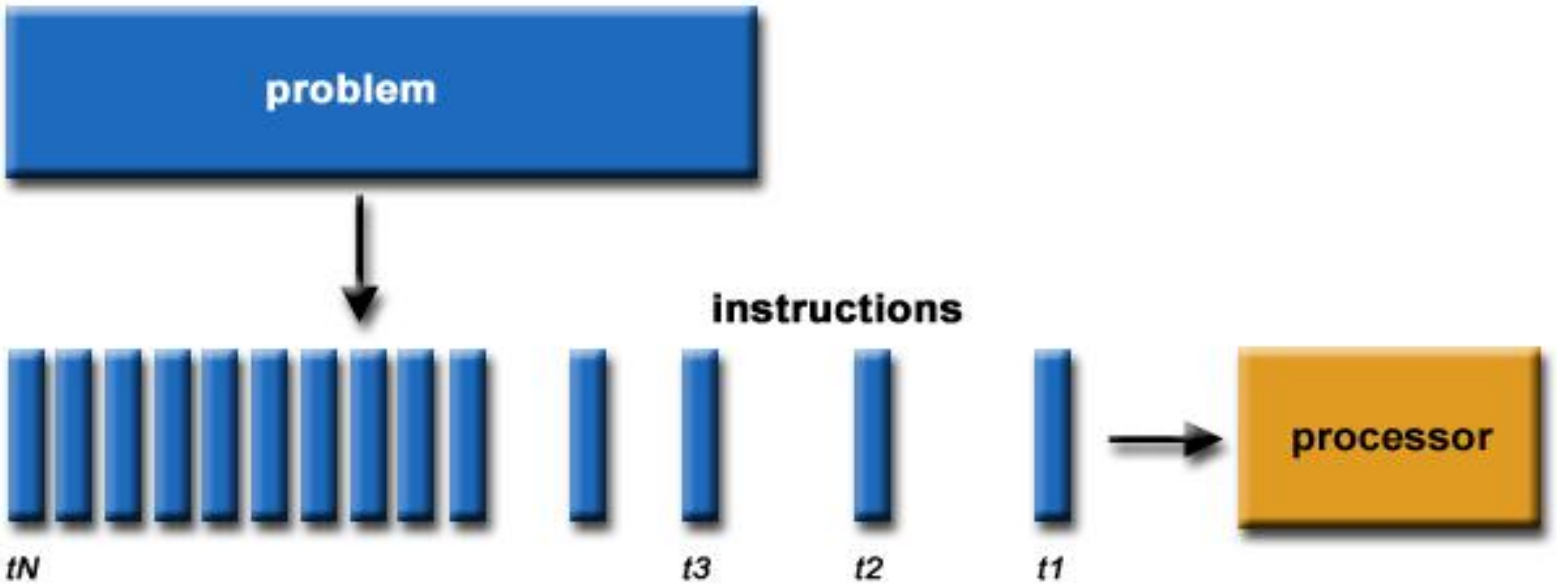
- 什么是并行计算
- 为什么需要并行计算
- 并行计算发展
- 并行计算面临的挑战

目录

- **什么是并行计算**
- 为什么需要并行计算
- 并行计算发展
- 并行计算面临的挑战

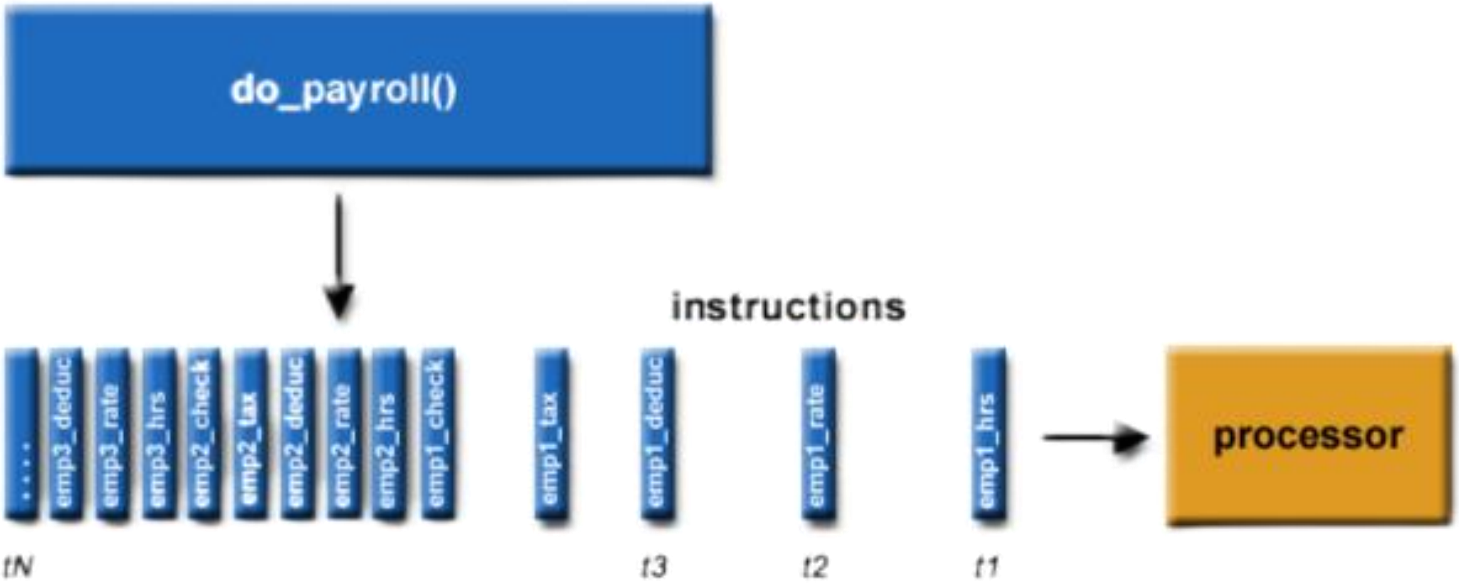
什么是并行计算?

- 传统情况下，程序是串行的(Serial Computing)
  - 问题被分解成一系列离散的指令
  - 这些指令顺序执行
  - 单个处理器上执行
  - 任意时刻只能有一条指令再执行



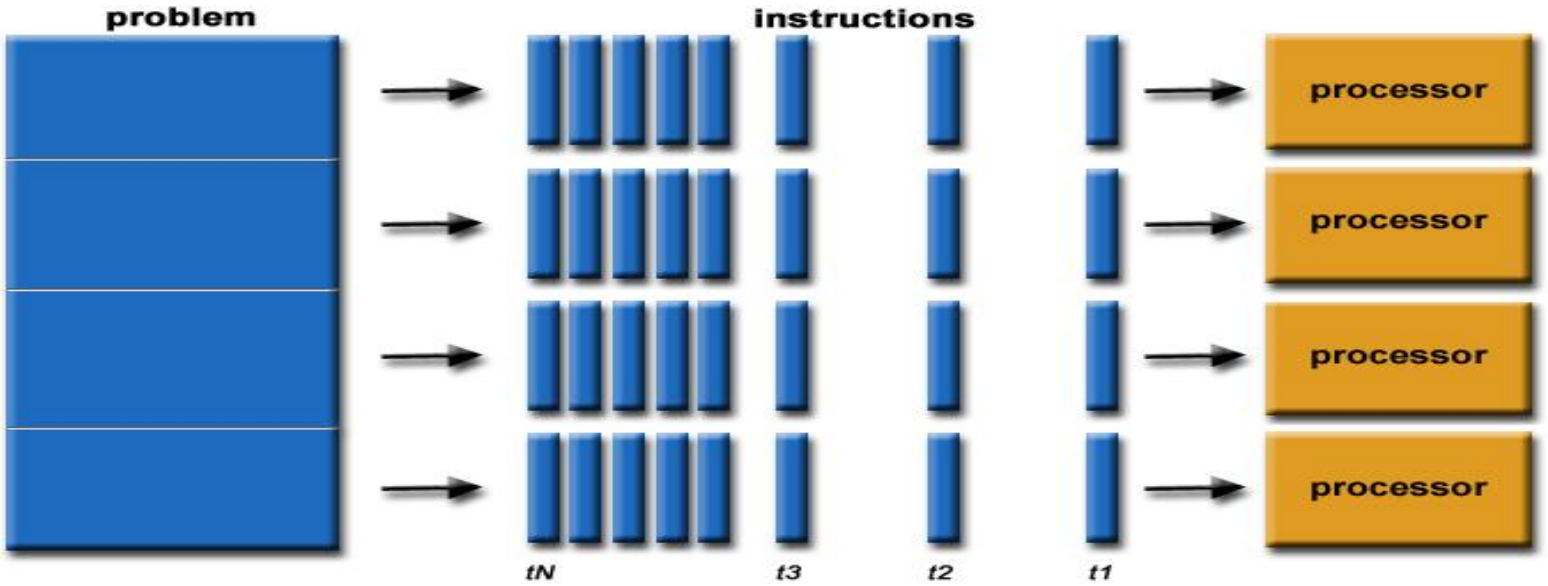
什么是并行计算?

- 传统情况下，程序是串行的(Serial Computing)
  - 问题被分解成一系列离散的指令
  - 这些指令顺序执行
  - 单个处理器上执行
  - 任意时刻只能有一条指令再执行



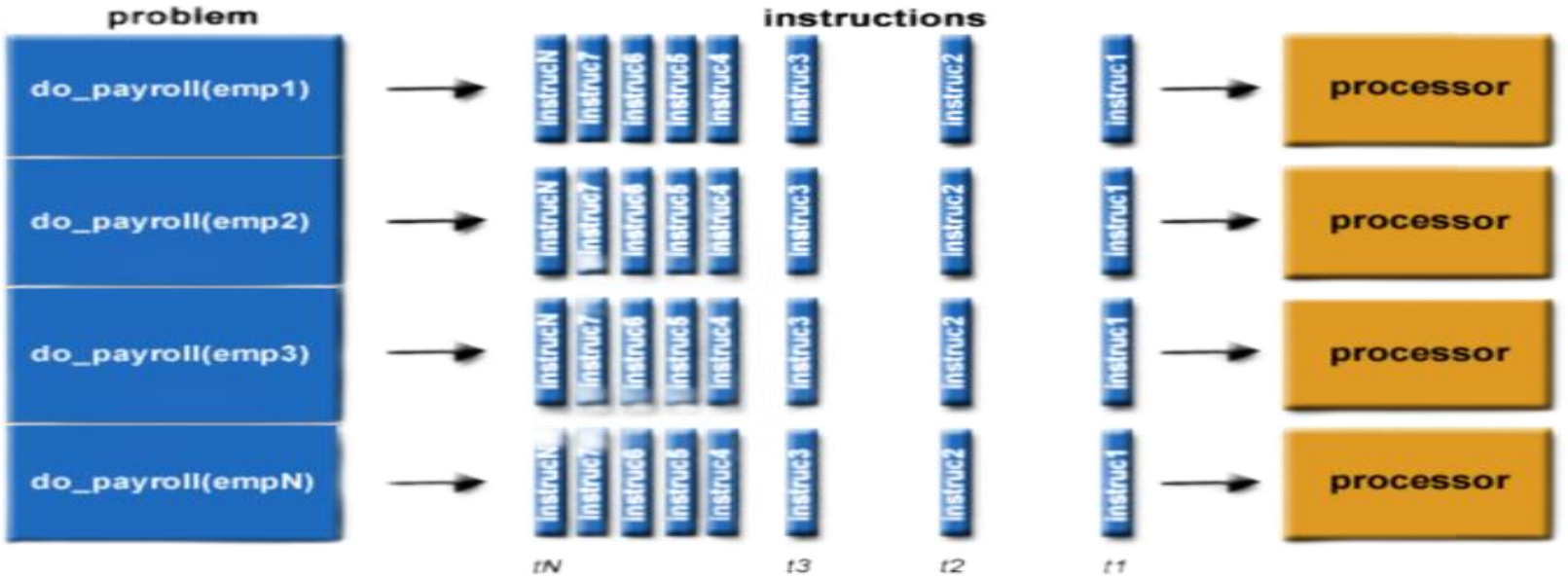
什么是并行计算?

- 并行计算(Parallel Computing): 同时使用多种计算资源解决计算问题
  - 问题被分为离散的部分
  - 每个部分进一步分解为离散的指令
  - 每个部分指令同时在不同处理器执行
  - 总体控制和协调机制



什么是并行计算?

- 并行计算(Parallel Computing): 同时使用多种计算资源解决计算问题
  - 问题被分为离散的部分
  - 每个部分进一步分解为离散的指令
  - 每个部分指令同时在不同处理器执行
  - 总体控制和协调机制



什么是并行计算?

- **身边的并行计算**

华为　小米　魅族　苹果




骁龙888，麒麟9000，Exynos 8895， A14

…　2核、4核

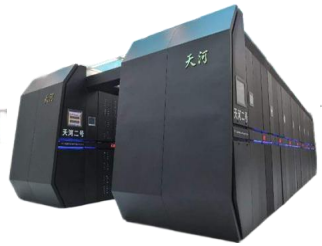
英特尔® 酷睿，苹果M1

2核、4核、8核

…

华为　小米　苹果




神威　天河　富岳

…

神威26010众核处理器、Matrix-2000、A64FX

1064万核、498万核、763万核

目录

- 什么是并行计算
- **为什么需要并行计算**
- 并行计算发展
- 并行计算面临的挑战

为什么需要并行计算?

- **并行计算更适合模拟和理解复杂的现实世界现象**

星系的形成

行星的运动

气候的变化

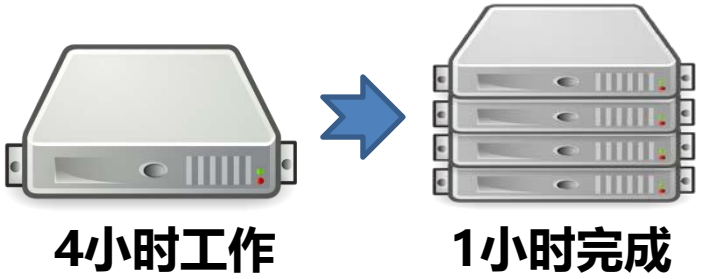
高峰期交通

板块运动

天气

为什么需要并行计算?

- **节约大量的时间&金钱**
  - 为一个任务分配更多的资源 → 节约时间
  - 并行计算机用更廉价的部件组合而成 → 节约金钱



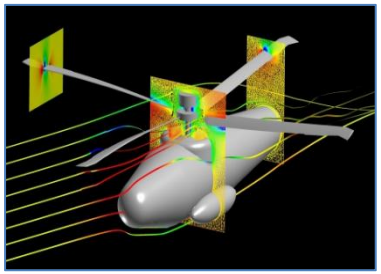
4小时工作　　1小时完成

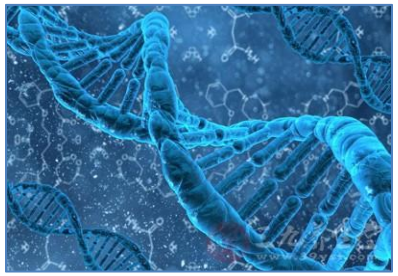
	DUAL XEON CPU server	DGX-1 GPU server (8 GPUs)
FLOPS	3TF	170TF
Mem BW	76GB/s	768GB/s
Alexnet Train Time	150 Hr	2Hr
Train in 2Hr	>250Nodes	1Node
Price	>$774000	$129,000

为什么需要并行计算?

- **解决更大规模/更复杂问题**
  - 计算机内存限制，串行处理大规模问题不切实际
  - 需要petaflops和petabytes计算资源，甚至“E级计算”
  - 示例：石油勘探

- 数学问题：Ax=b
- 工区：20km×20km×10km
- 网格：50m×50m×20m
- 网格数量：8千万
- 数据量：TB级
- 100台服务器（节点）
- 执行时间：>10小时





为什么需要并行计算?

- **训练更大的AI模型**
  - 单机/普通数据中心算力和存储不足以训练大模型
  - 需要petaflops和petabytes计算资源，甚至“E级计算”
  - 示例: 清华大学“八卦炉”模型

- 超算平台：新一代神威超级计算机
- 模型参数：174万亿
- 训练性能：> 1 EFLOPS (混合精度)

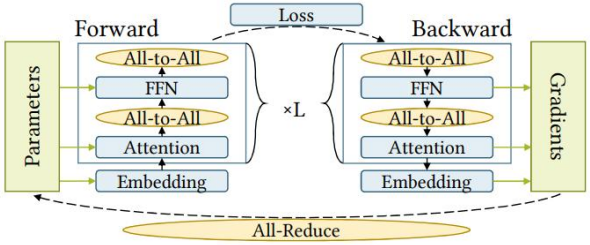


Figure 1. Simplified computing process of the proposed model.

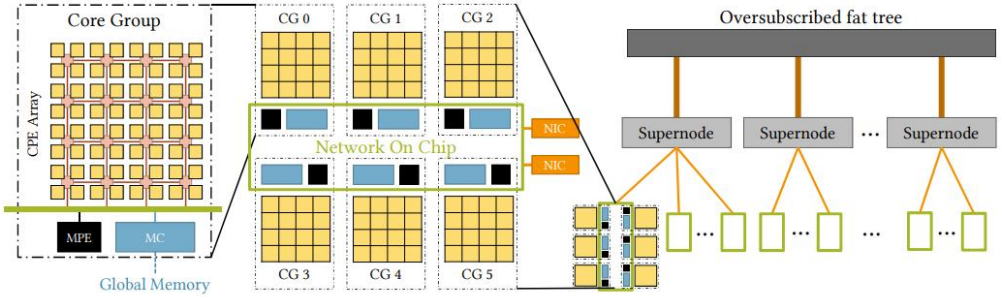
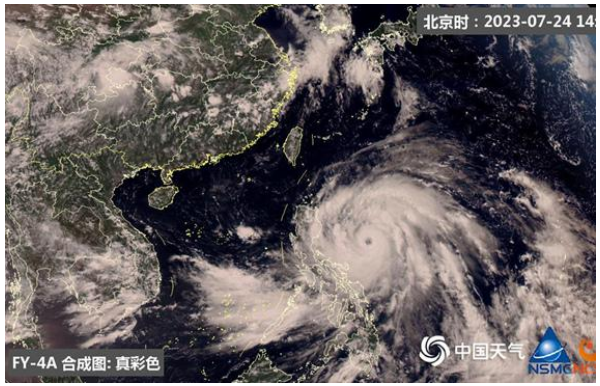


Figure 2. Architecture of the *New Generation Sunway Supercomputer*.

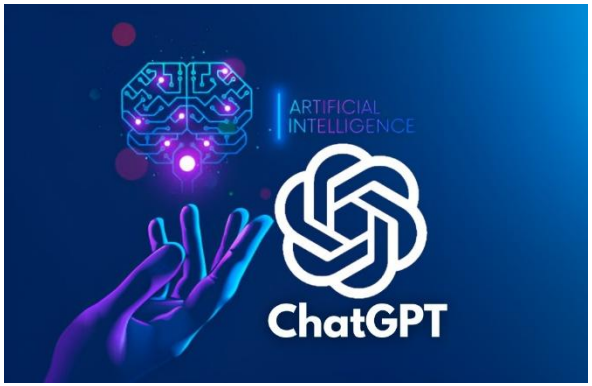
为什么需要并行计算?

- **典型并行计算应用**

天体物理仿真

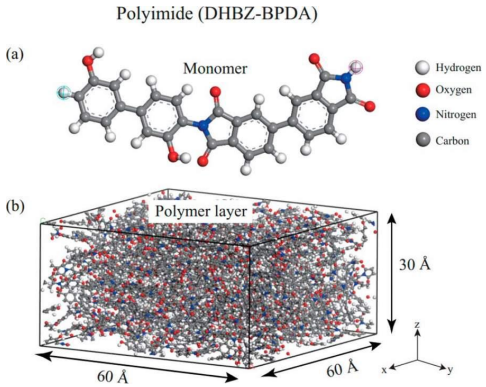


天气预测

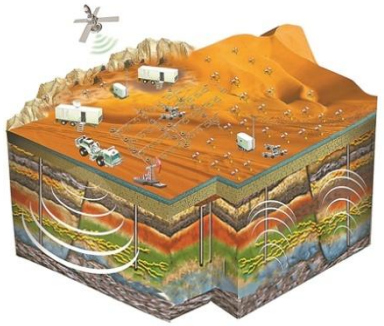


人工智能大模型

核爆模拟



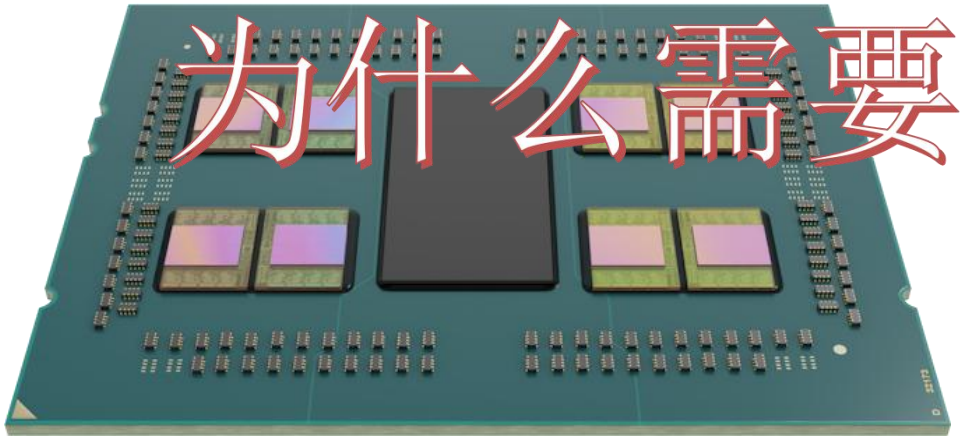
分子动力学仿真



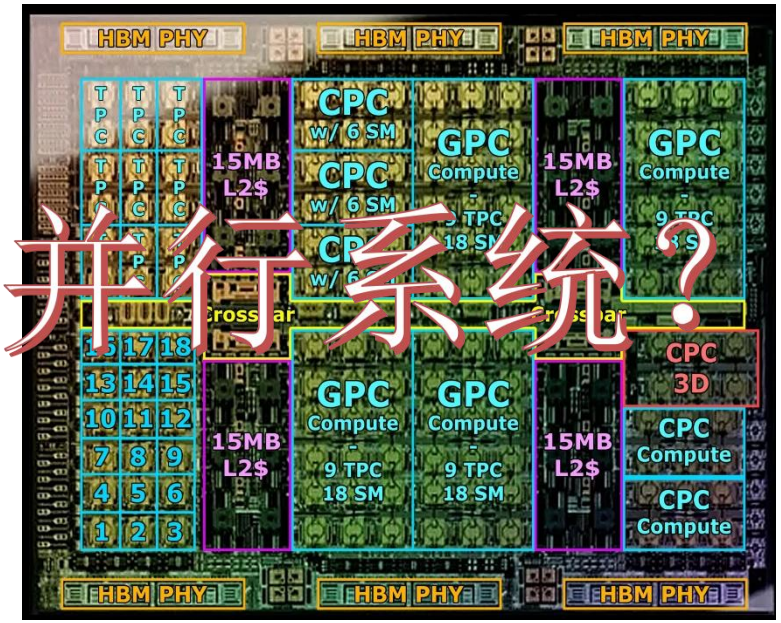
物探(矿物勘探)

为什么需要并行计算?

- **理解底层硬件架构，充分利用硬件性能**



64 Cores AMD EPYC 7003系列CPU

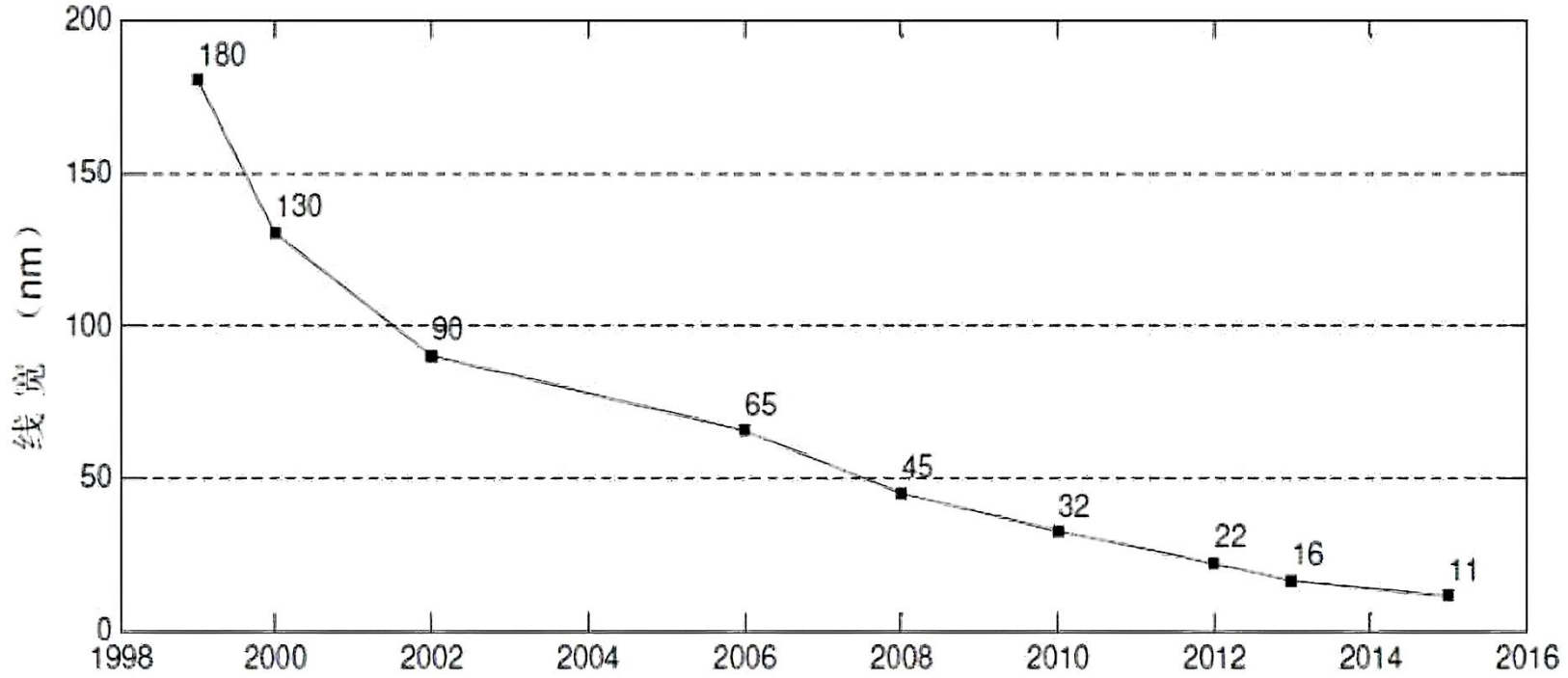


18432 Cores NVIDIA H100 GPU

为什么需要并行系统？

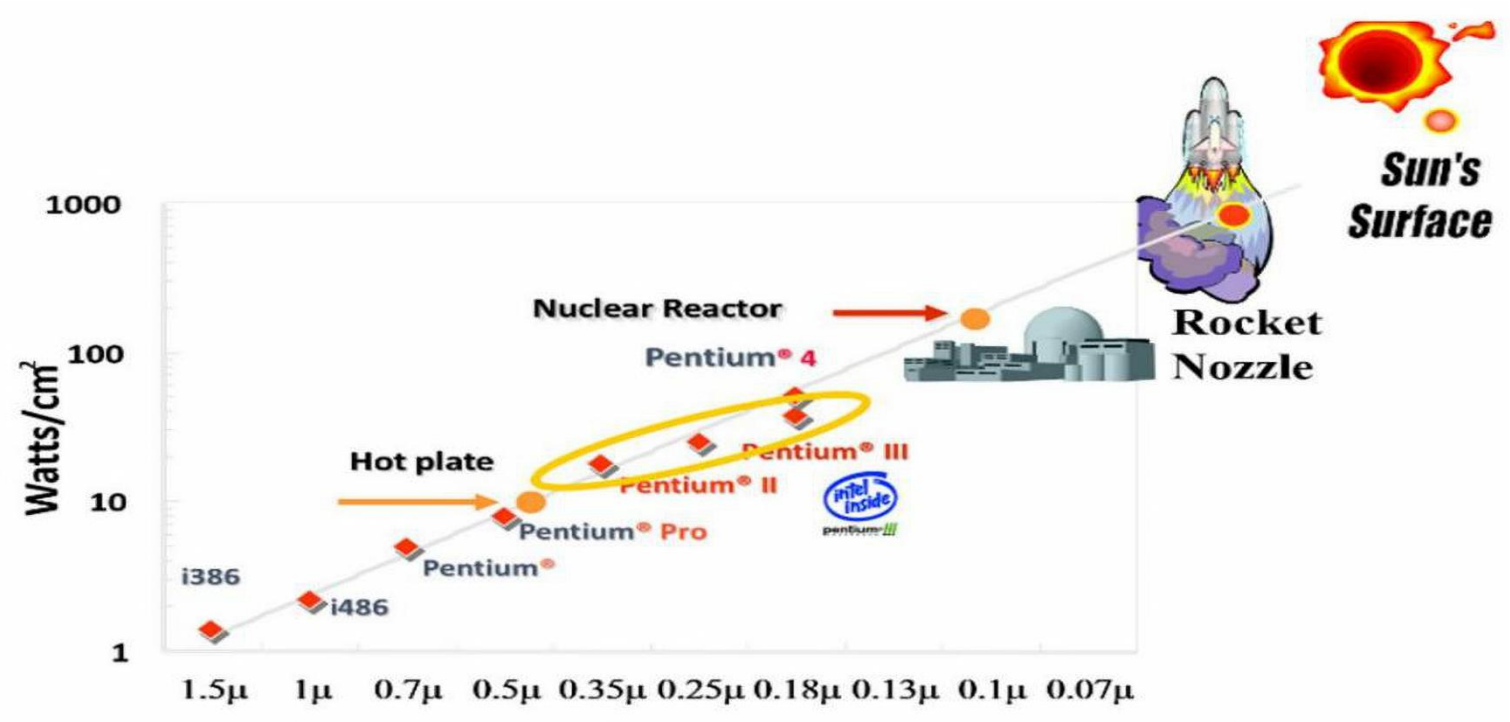
为什么需要并行计算?

- **晶体管线宽不断降低**
  - 晶体管密度增加，频率增加，芯片性能提高
  - 台积电、三星：5nm
  - 中芯国际：14nm



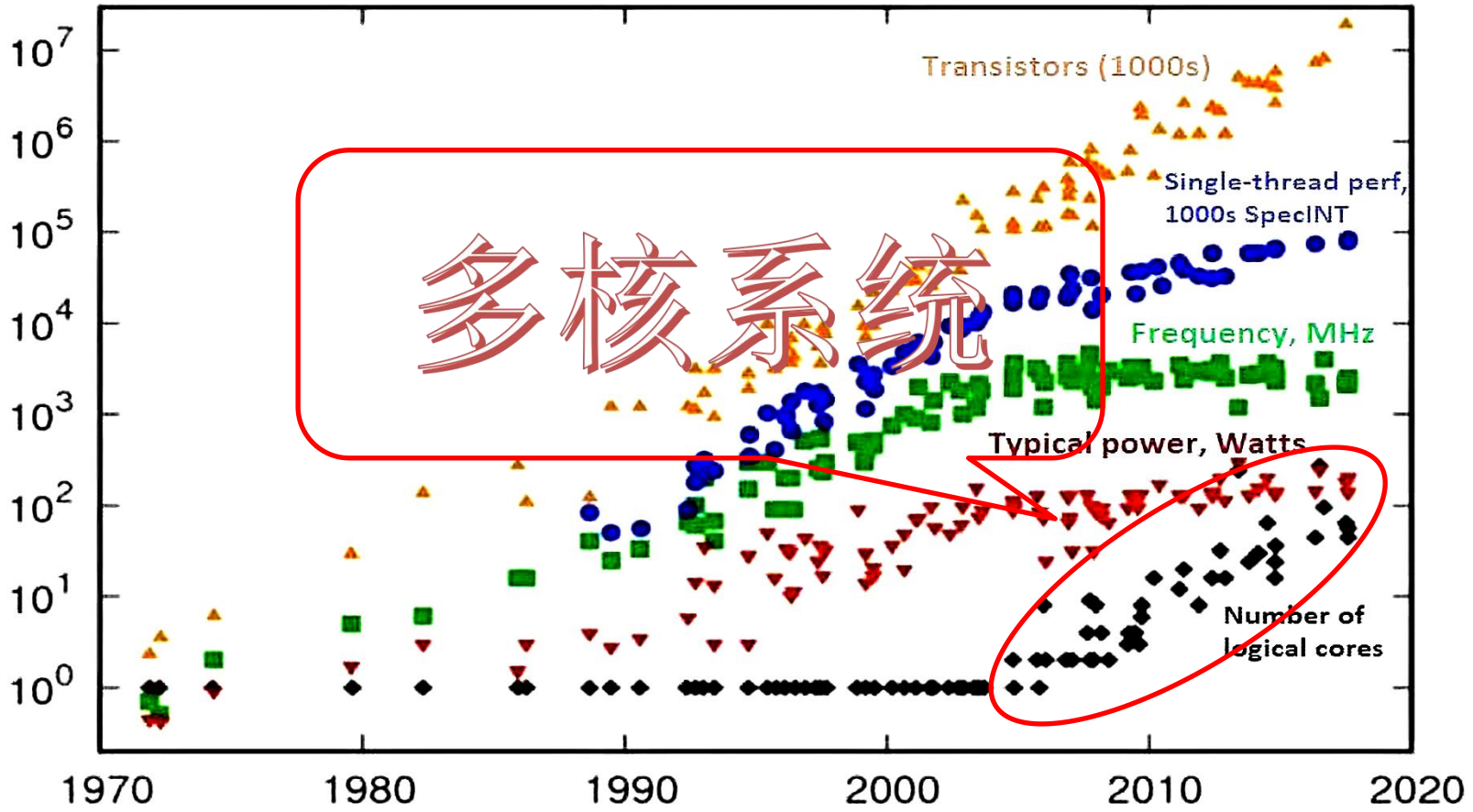
为什么需要并行计算?

- 芯片功耗密度不断增加
  - 温度上升影响芯片的可靠性
  - 45nm工艺，每百个芯片每月发生一次故障
  - 16nm工艺，每百个芯片每天发生一次故障



为什么需要并行计算?

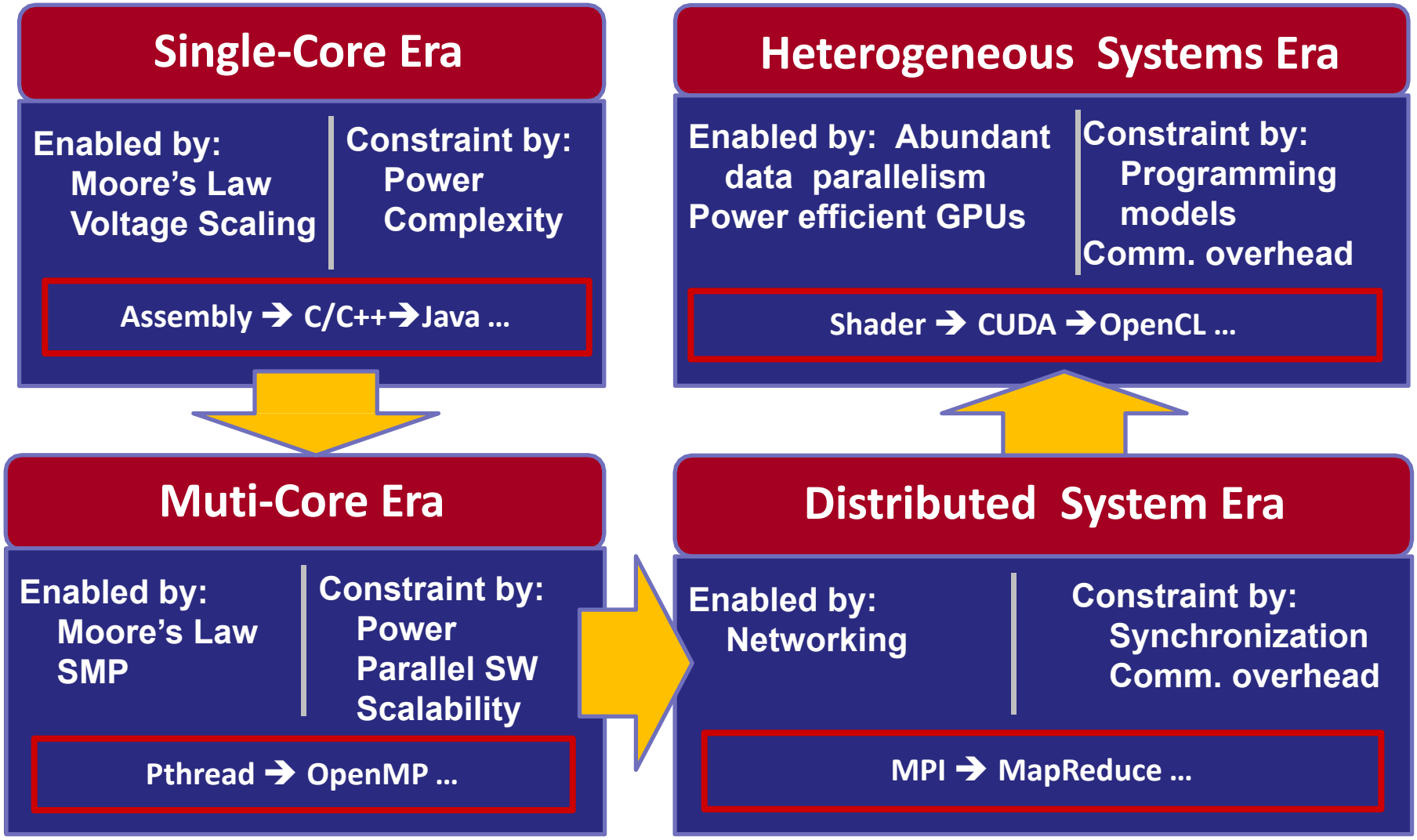
- **单处理器的性能提升变缓**
  - **2002之前：50%/Year；2002之后：<20%/Year**



目录

- 什么是并行计算
- 为什么需要并行计算
- **并行计算发展**
- 并行计算面临的挑战

并行计算发展


Single-Core Era
Enabled by:
Moore's Law
Voltage Scaling
Constraint by:
Power
Complexity
Assembly → C/C++→Java …
Heterogeneous Systems Era
Enabled by: Abundant data parallelism
Power efficient GPUs
Constraint by:
Programming models
Comm. overhead
Shader → CUDA →OpenCL …
Muti-Core Era
Enabled by:
Moore's Law
SMP
Constraint by:
Power
Parallel SW
Scalability
Pthread → OpenMP …
Distributed System Era
Enabled by:
Networking
Constraint by:
Synchronization
Comm. overhead
MPI → MapReduce …

并行计算发展

- **始于70年代**
  - **1946年第一台计算机 ENIAC (Electronic Numerical Integrator And Computer)**

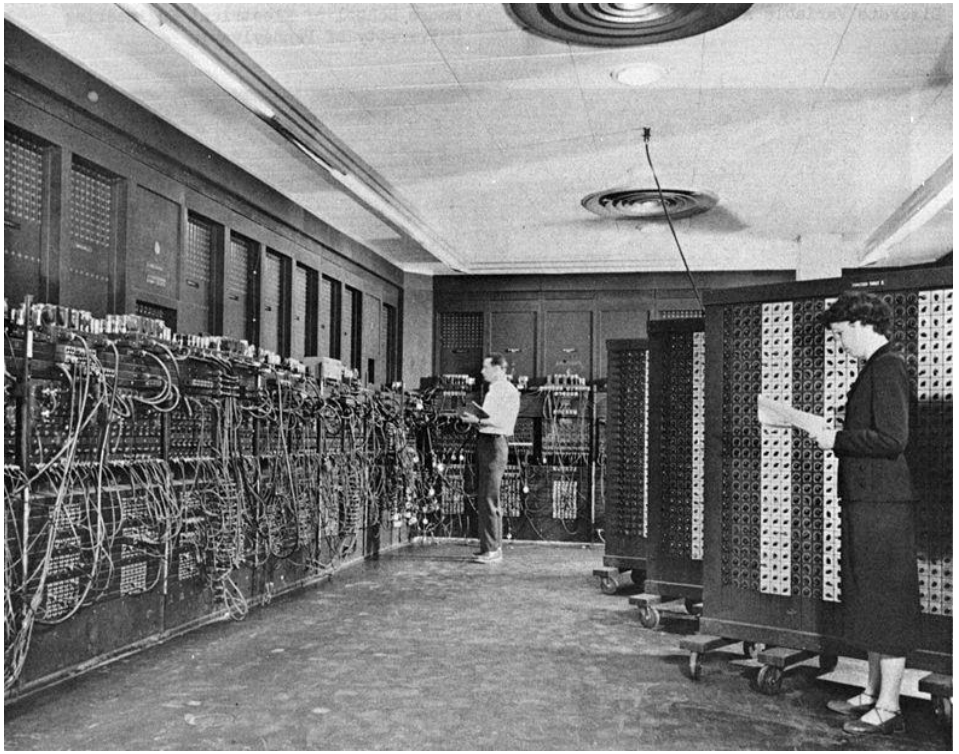
①、占地170平方

②、重约 30 吨

③、5000 次加法/秒或500次乘法/秒

④、15分钟换一个零件

⑤、主要用于弹道计算和氢弹研制



并行计算发展

- **始于70年代**
  - **1972年第一台并行计算机 ILLIAC IV（伊利诺依大学）**

①、60年代末开始建造

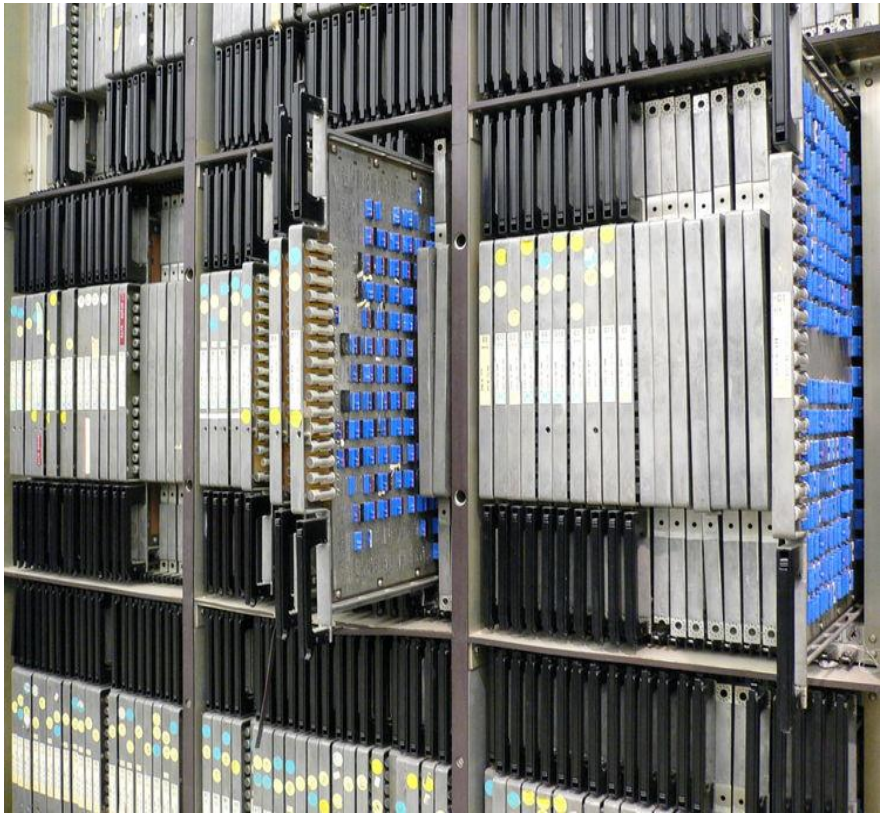
②、72年建成, 74年运行第一个完整程序, 76年运行第一个应用程序

③、64个处理器，是当时性能最高CDC7600机器的2-6倍

④、公认的1981年前最快

⑤、1982年退役

⑥、可扩展性好，可编程性差



并行计算发展

- 始于70年代
  - 向量机 Cray-1

①、一般将 Cray-1 投入运行的 1976 年称为“超级计算元年”

②、编程方便，但可扩展性差

③、以 Cray 为代表的向量机称雄超级计算机界十几载

收藏于 Deutsches Museum 德意志博物馆的 Cray-1原型

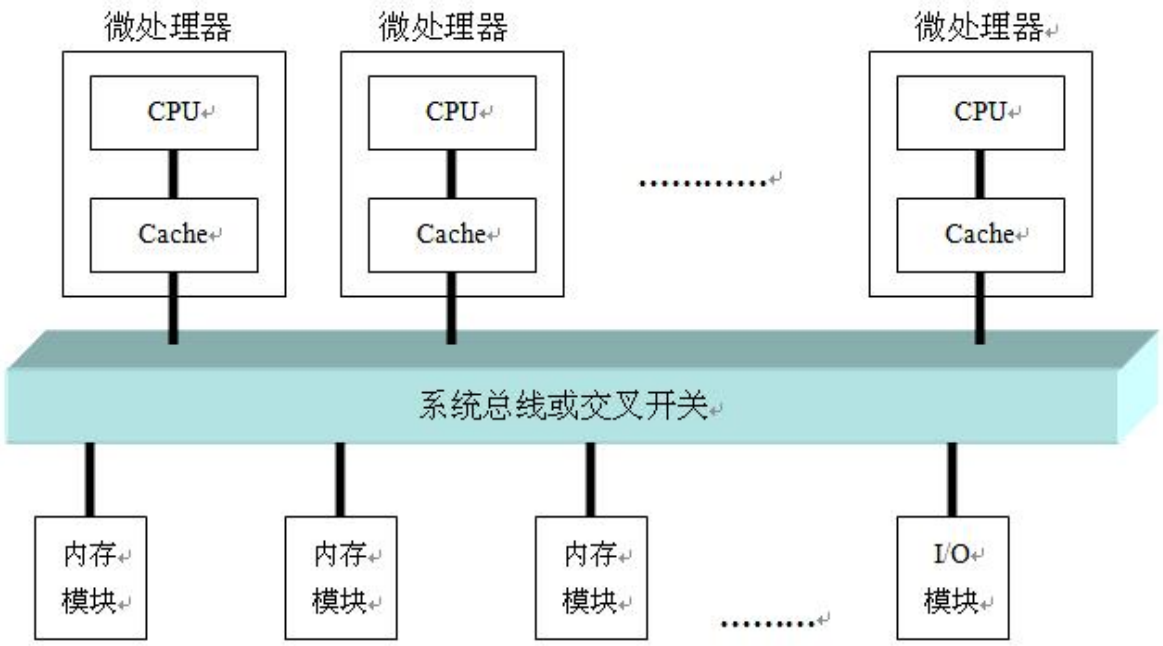
并行计算发展

- 80年代：百家争鸣
  - 早期：以 MIMD 并行计算机的研制为主
    - Denelcor HEP (1982年)第一台商用 MIMD 并行计算机
    - IBM 3090 80 年代普遍为银行所采用
    - Cray X-MP Cray 研究公司第一台 MIMD 并行计算机

西摩·克雷 Seymour Cray (1925-1996)，
电子工程学学士，应用数学硕士，
超级计算机之父， Cray研究公司的创始人，
亲手设计了Cray机型的全部硬件与操作系统，
作业系统由他用机器码编写完成。1984年时，
公司占据了超级计算机市场 70%的份额。
1996年Cray研究公司被SGI收购，2000年被
出售给Tera计算机公司，成立Cray公司。

并行计算发展

- 80年代：百家争鸣
  - 中期：共享存储多处理机 SMP
    - SMP：在一个计算机上汇集一组处理器，各处理器对称共享内存及计算机的其他资源，由单一操作系统管理，极大提高整个系统的数据处理能力



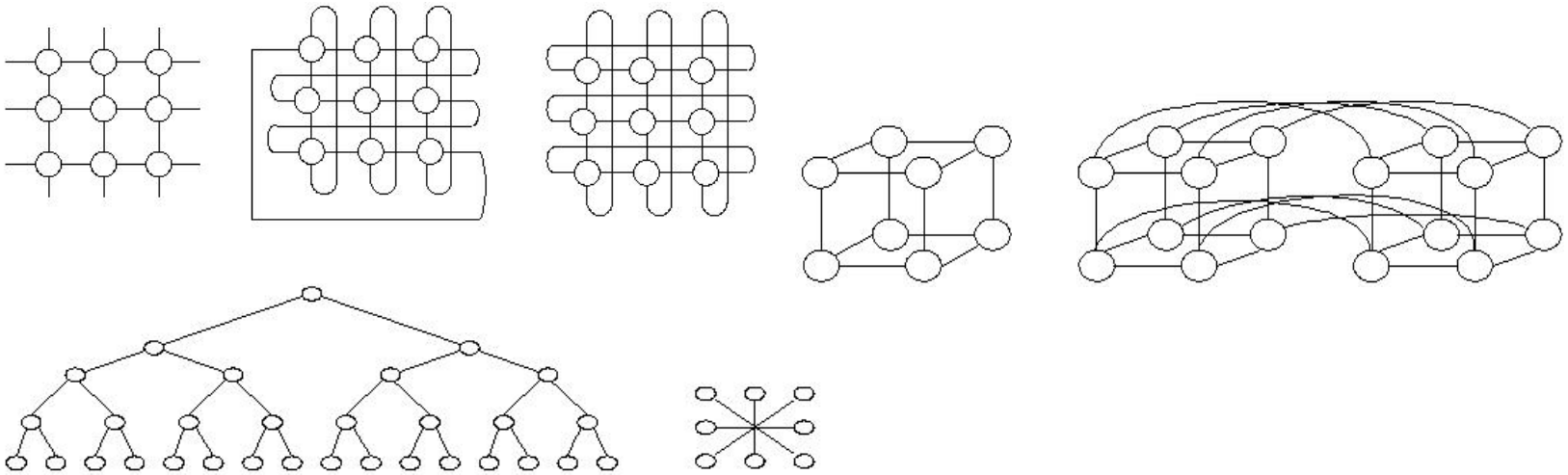
扩展性较差

可靠性较差

内存访问瓶颈

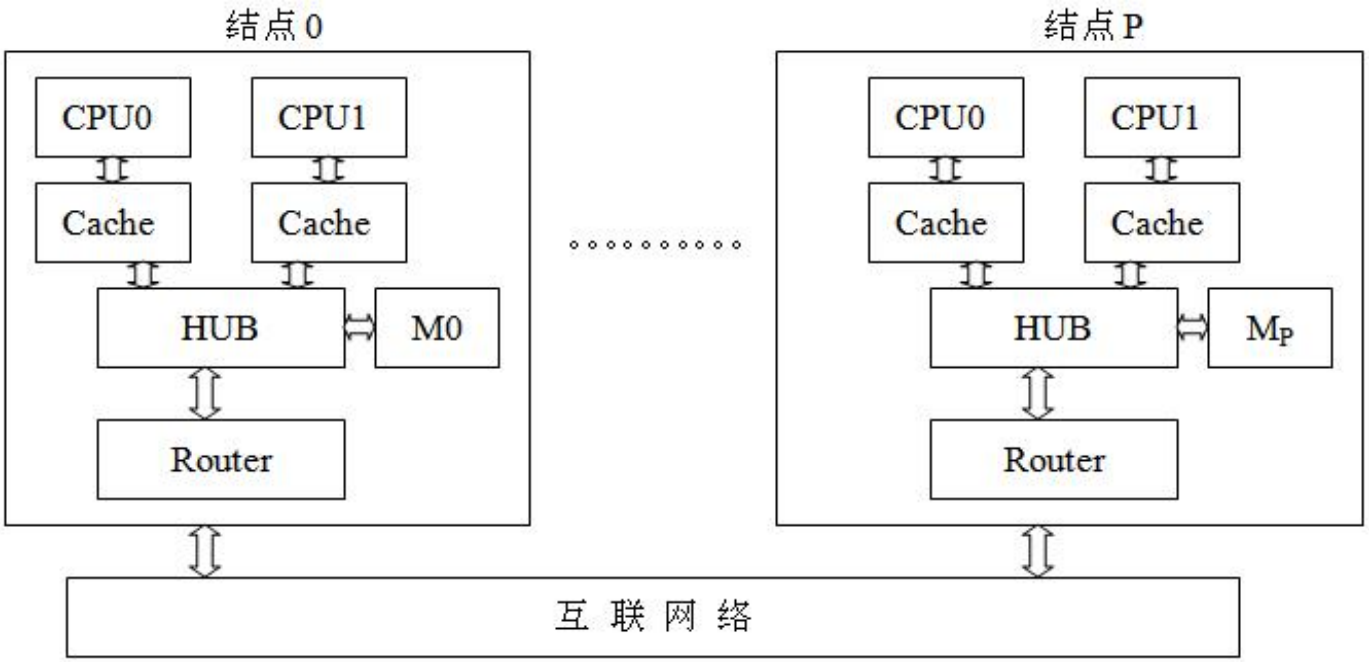
并行计算发展

- 80年代：百家争鸣
  - 后期：具有强大计算能力的并行机
    - 通过二维Mesh连接的Meiko（Sun）系统
    - 超立方体连接的 MIMD 并行机：nCUBE-2、iPSC/80
    - 共享存储向量多处理机 Cray Y-MP
    - ……



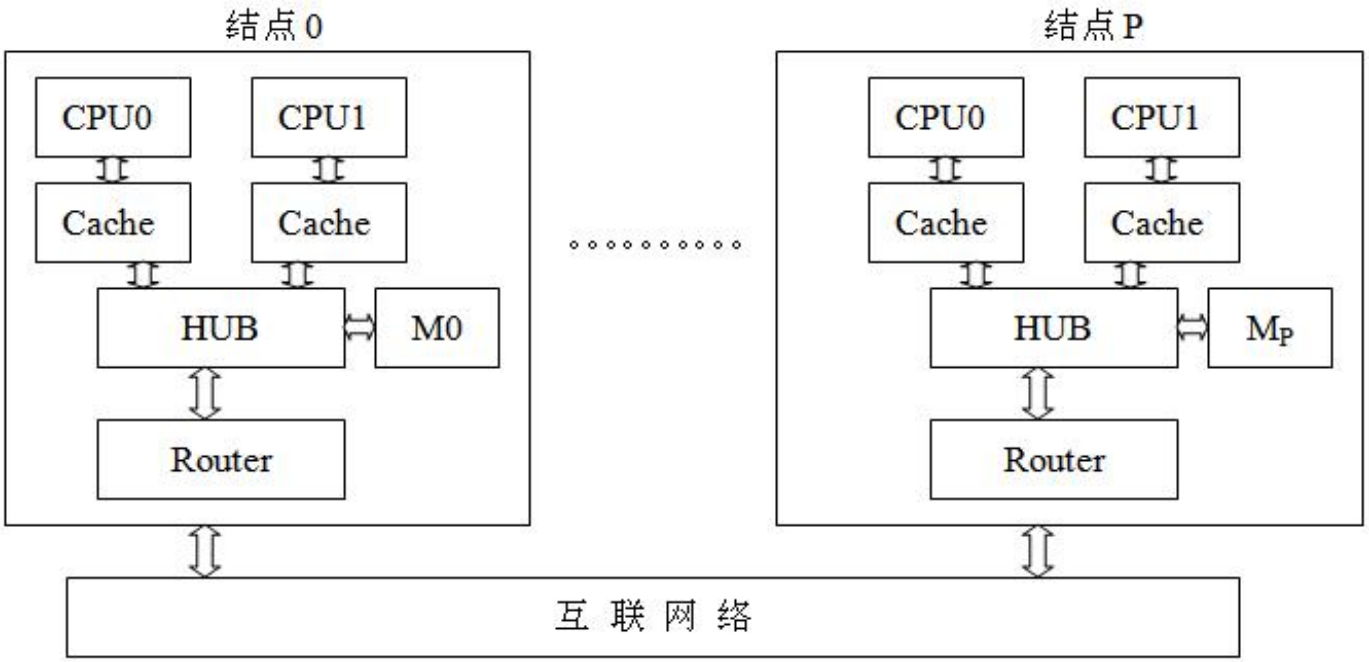
并行计算发展

- 90 年代：体系结构框架趋于统一
  - DSM （Distributed Shared Memory）分布式共享存储
    - 以结点为单位，每个结点有一个或多个CPU
    - 专用的高性能互联网络连接（Myrinet, Infiniband, … ）



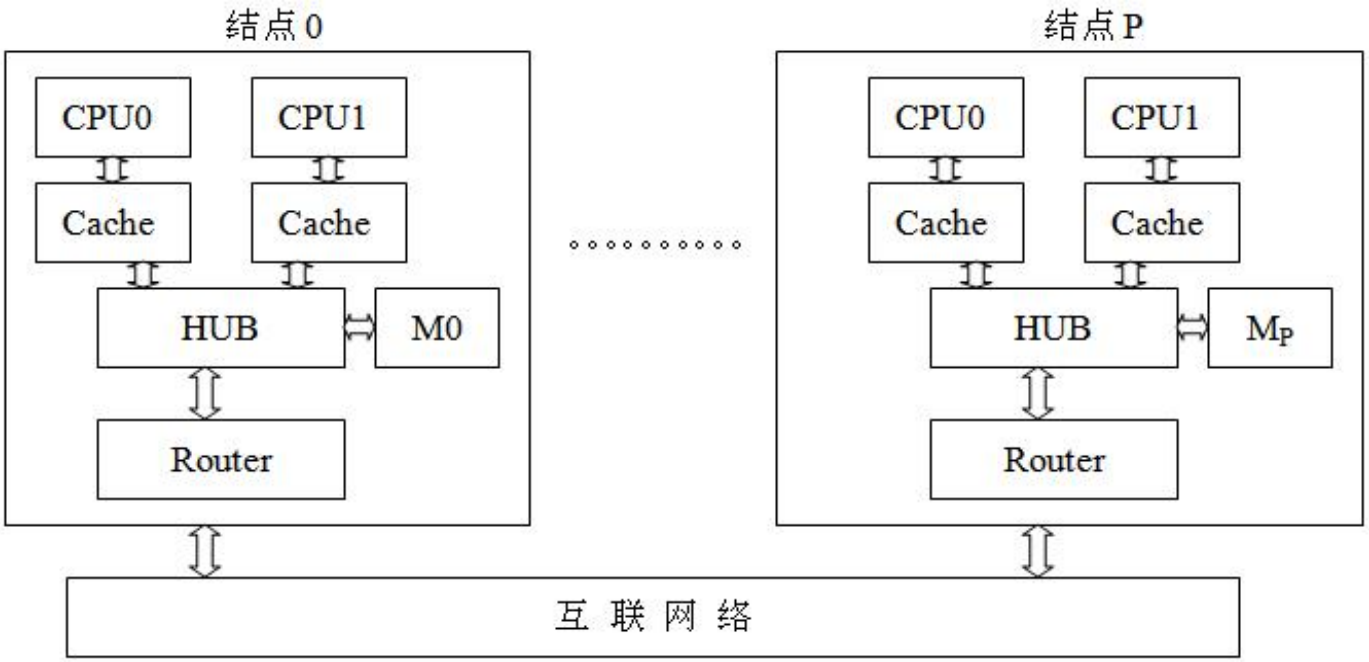
并行计算发展

- 90 年代：体系结构框架趋于统一
  - DSM （Distributed Shared Memory）分布式共享存储
    - 分布式存储：内存模块局部在每个结点中
    - 单一的操作系统



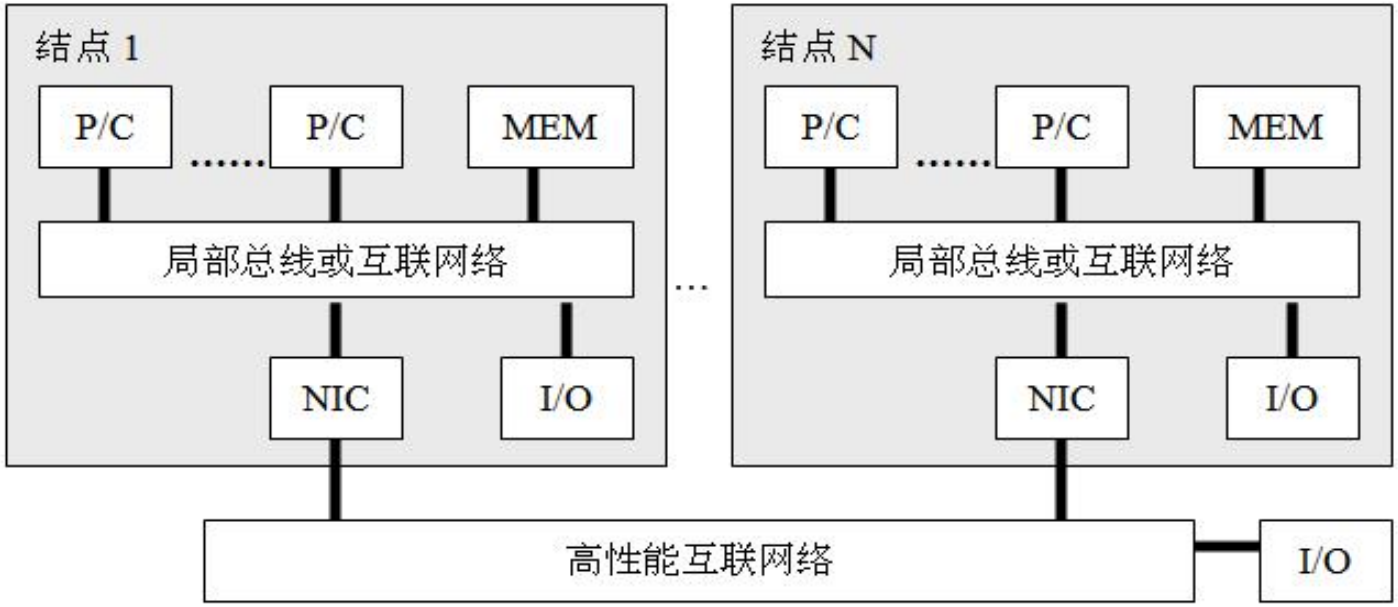
并行计算发展

- 90 年代：体系结构框架趋于统一
  - DSM （Distributed Shared Memory）分布式共享存储
    - 单一的内存地址空间
    - 可扩展到上百个结点



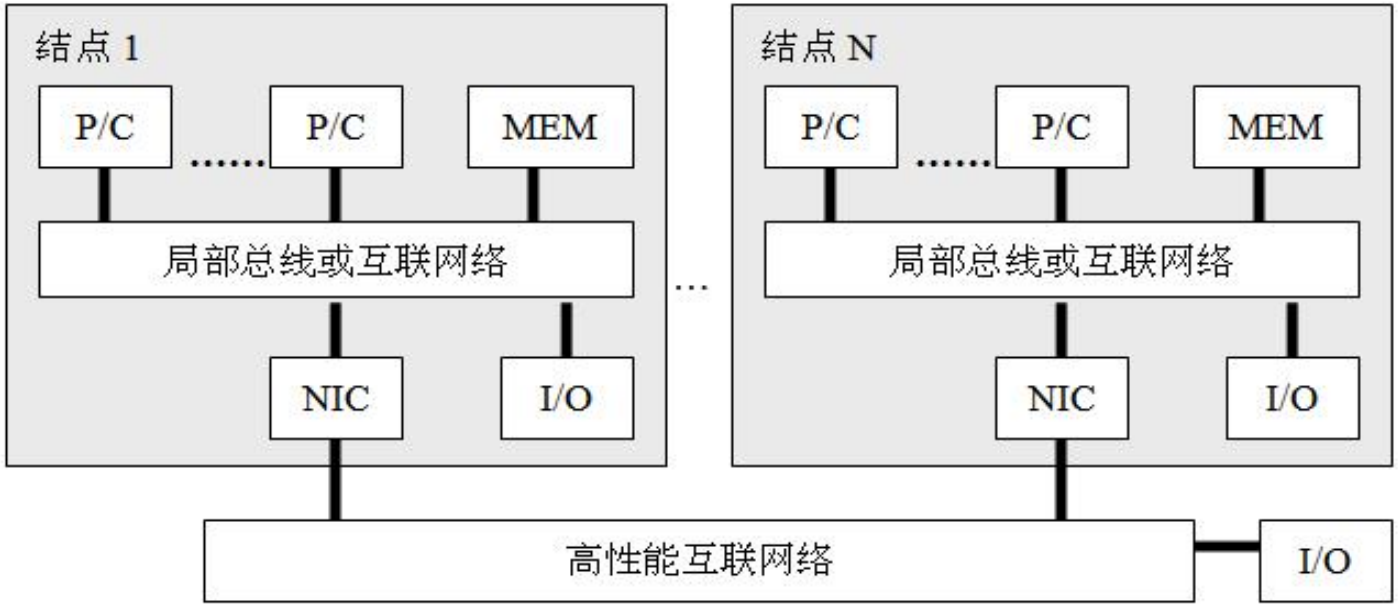
并行计算发展

- 90 年代：体系结构框架趋于统一
  - MPP（Massively Parallel Processing）大规模并行处理结构
    - 每个结点相对独立，有一个或多个微处理器
    - 每个结点均有自己的操作系统



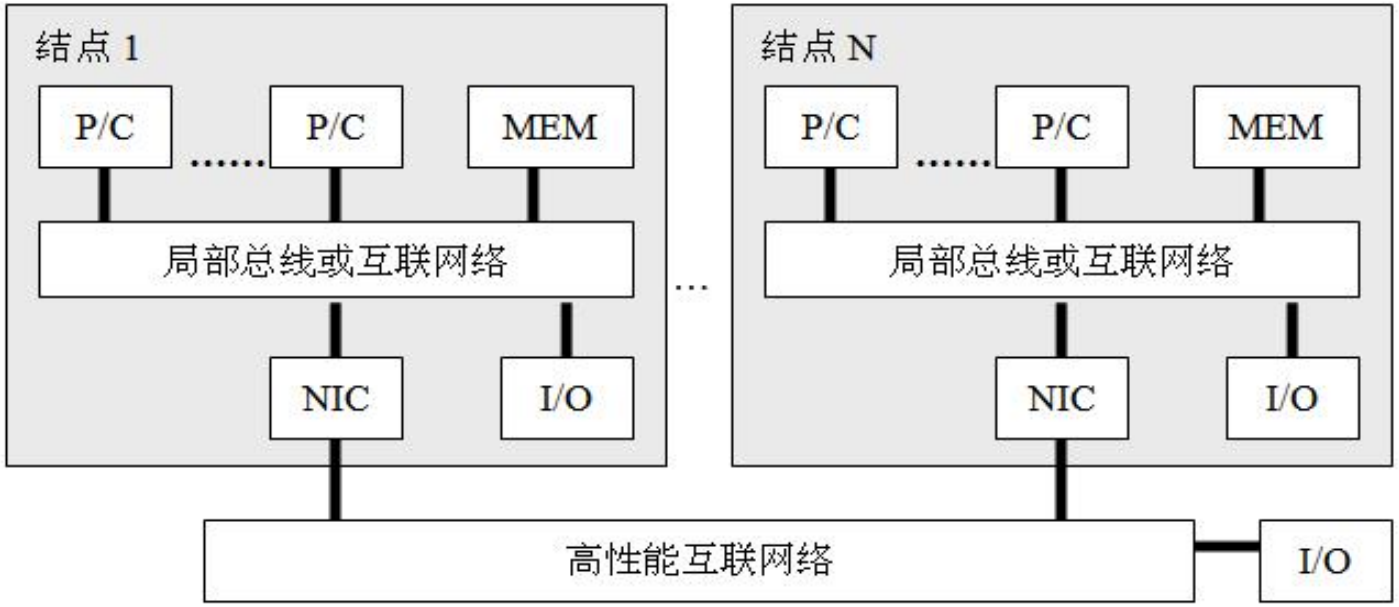
并行计算发展

- 90 年代：体系结构框架趋于统一
  - MPP（Massively Parallel Processing）大规模并行处理结构
    - 各个结点自己独立的内存，避免内存访问瓶颈
    - 各个结点只能访问自己的内存模块



并行计算发展

- 90 年代：体系结构框架趋于统一
  - MPP（Massively Parallel Processing）大规模并行处理结构
    - 扩展性较好

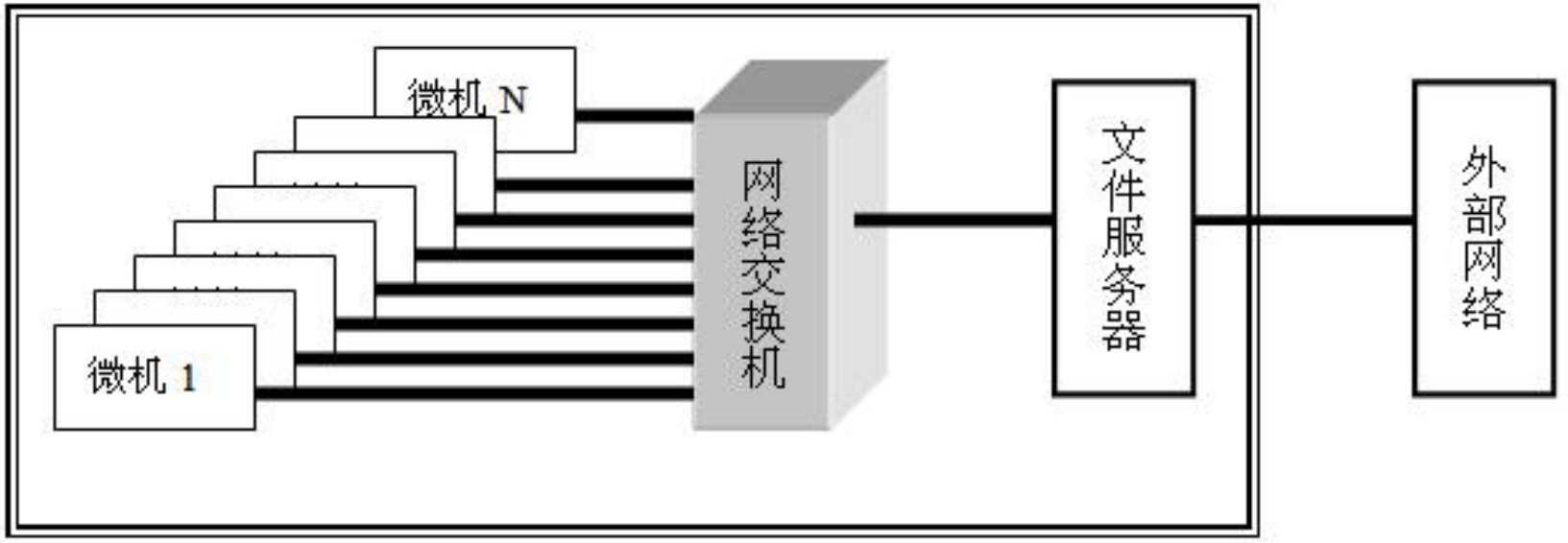


并行计算发展

- 90 年代：体系结构框架趋于统一
  - NOW（Network of Workstations）工作站机群
    - 每个结点都是一个完整的工作站，有独立的硬盘与UNIX系统
    - 结点间通过低成本的网络（如千兆以太网）连接
    - 每个结点安装消息传递并行程序设计软件，实现通信、负载平衡等
    - 投资风险小、结构灵活、可扩展性强、通用性好、异构能力强，被大量中小型计算用户和科研院校所采用
    - 也称为 COW（Cluster of Workstations）
    - NOW（COW）与 MPP 之间的界线越来越模糊

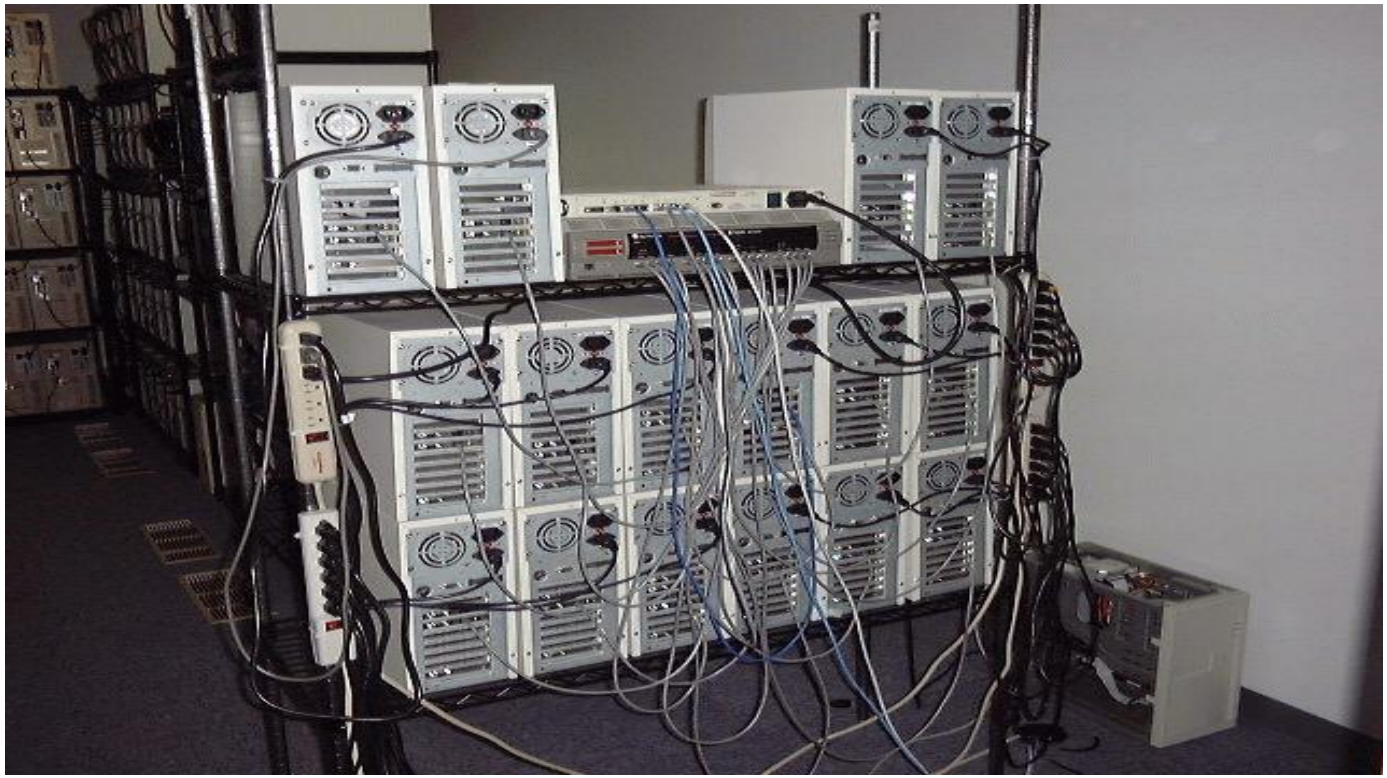
并行计算发展

- 90 年代：体系结构框架趋于统一
  - NOW（Network of Workstations）工作站机群
    - NOW的典型代表：Beowulf cluster 微机机群
    - 性能价格比极高



并行计算发展

- 90 年代：体系结构框架趋于统一
  - NOW（Network of Workstations）工作站机群
    - 第一台 Beowulf 机群

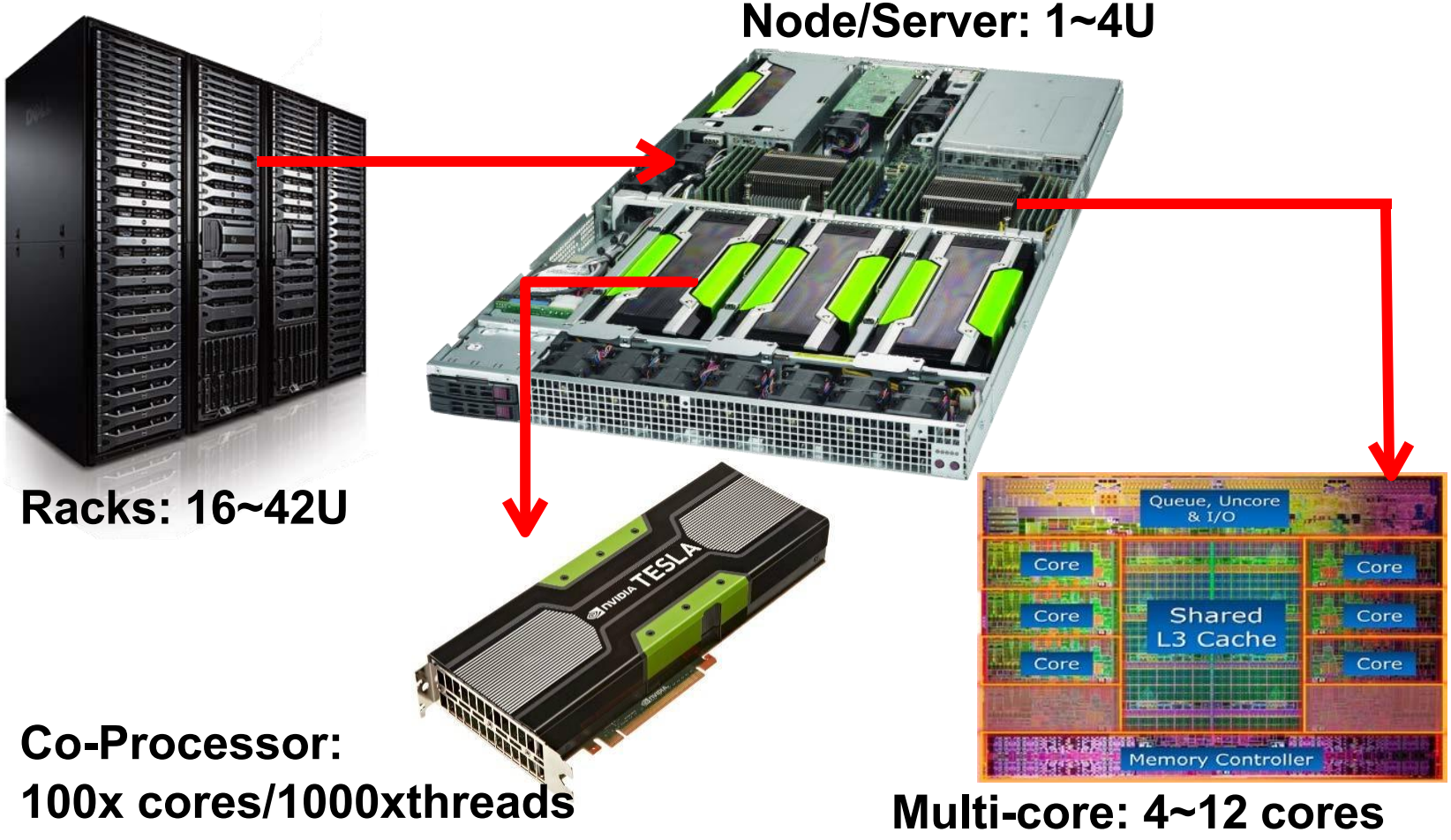


并行计算发展

- **2000 年至今：前所未有大踏步发展**
  - **Cluster 机群**
    - 每个结点含多个商用处理器，结点内部共享存储
    - 采用商用机群交换机通过总线连接结点，结点分布存储
    - 各结点采用Linux操作系统、GNU编译系统和作业管理系统
  - **Constellation 星群**
    - 每个结点是一台子并行机
    - 采用商用机群交换机通过总线连接结点，结点分布存储
    - 各个结点运行专用的操作系统、编译系统和作业管理系统
  - **MPP**
    - 专用高性能网络，大多为政府直接支持

并行计算发展

- 目前典型的并行系统组成（引入异构架构）



并行计算发展

- **并行计算发展水平的标志：超级计算机**
  - 体量巨大、造价高昂的设备，拥有数以万计的处理器，旨在执行专业性强、计算密集型的任务
  - 它的性能是以每秒浮点运算（FLOPS）来衡量的，而不是以每秒百万条指令（MIPS）来衡量的
  - Top500排名（1993年开始），每年发布两次

前沿(Frontier)

① Top500排名第一，1.194 EFlop/s (1E=10^18)

② 9402个AMD EPYC CPU

③ 437608个AMD MI250X GPU

③ 37.608PB总内存，700PB存储

美国前沿（Frontier）超级计算机

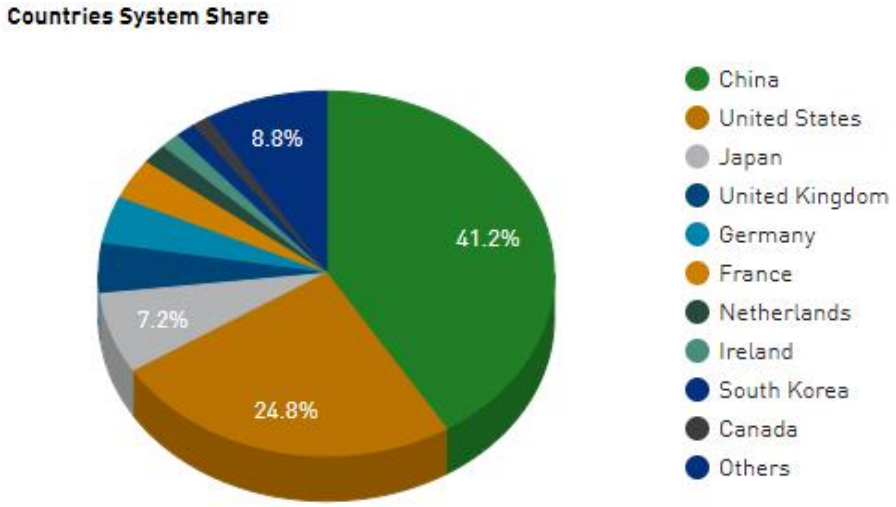
并行计算发展

- **并行计算发展水平的标志：超级计算机**
  - 最新的硬件技术
  - 定制的系统配置
  - 优化的软件和函数库
  - 巨大的资金投入和能耗成本

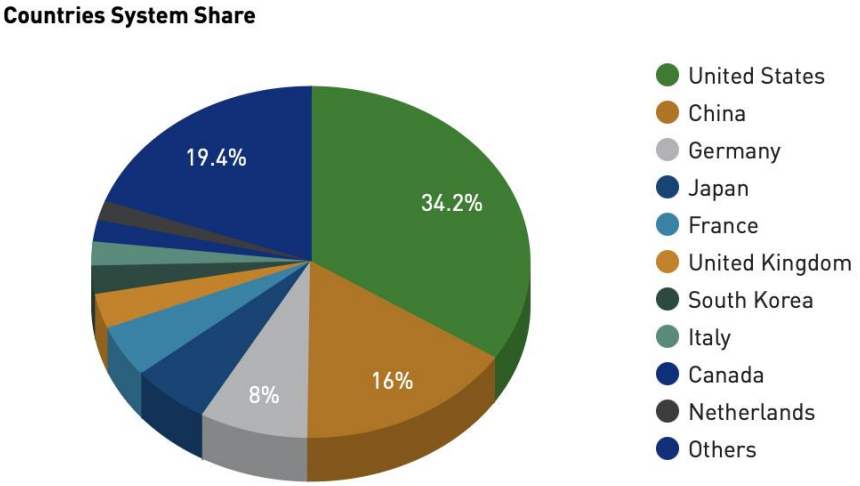
科技军备竞赛：国家高新技术发展水平的重要标志

并行计算发展

- Top500趋势：国家对比
  - 美国超算数量第一，中国超算数量第二



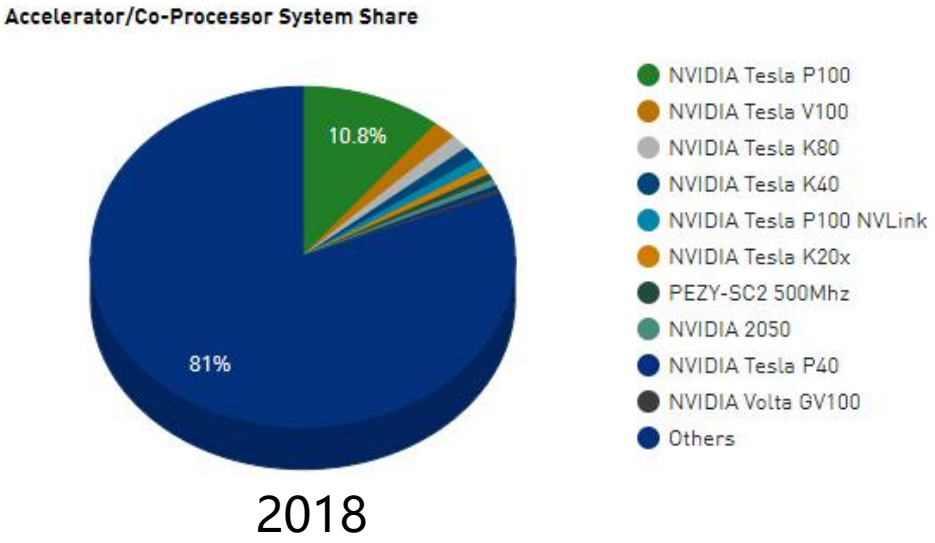
2018



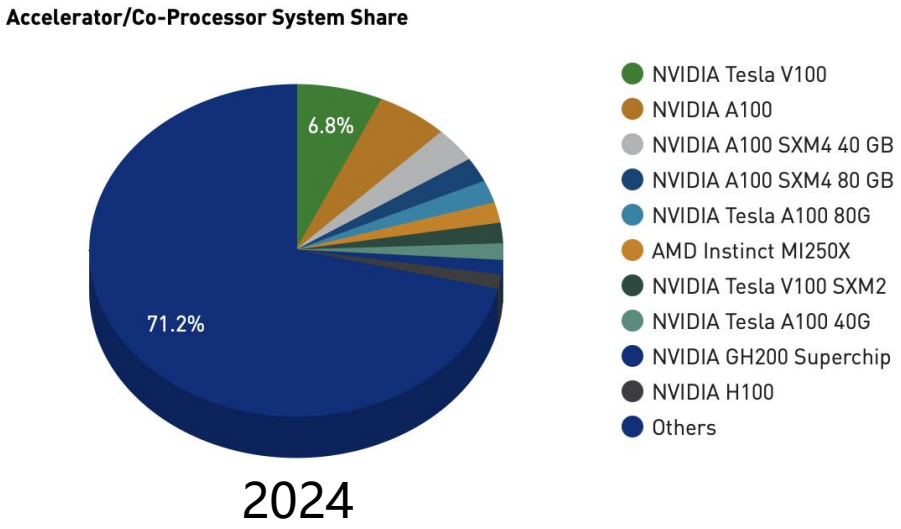
2024

并行计算发展

- **Top500趋势：算力架构对比**
  - **伴随AI等应用的强势崛起，异构算力成最火爆的概念**
  - **NVIDIA GPU、Intel协处理器、国防科大Matrix-2000、曙光DCU**



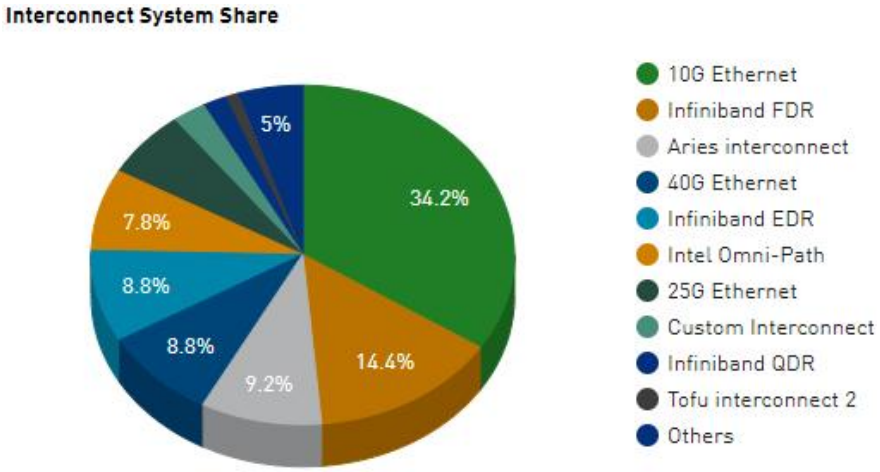
2018



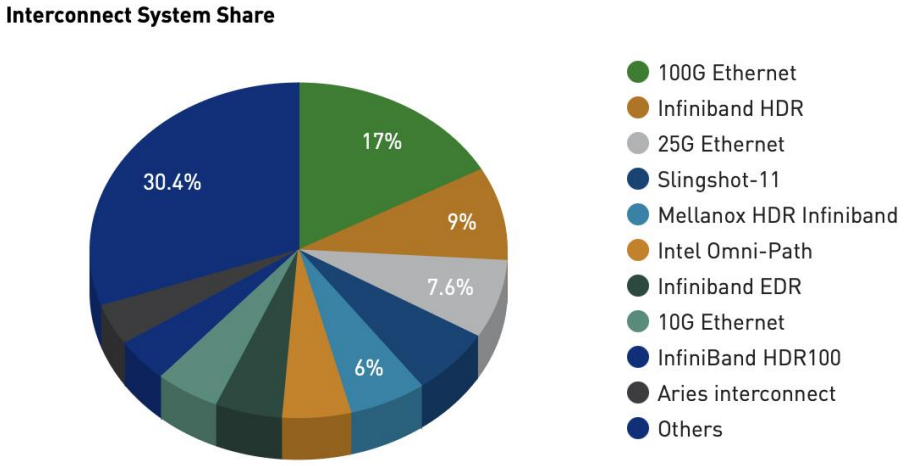
2024

并行计算发展

- Top500趋势：互联网络对比
  - 以太网交换机和网卡包含RDMA、智能网络编排
  - 以太网相对于Infiniband和各类定制网络，性价比更高



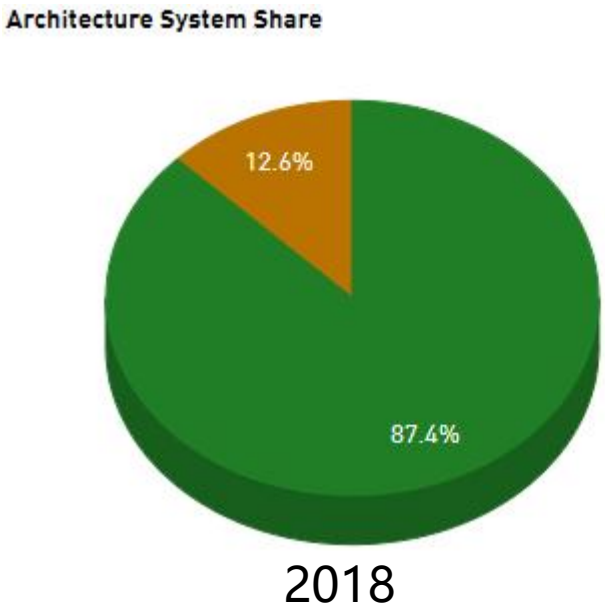
2018



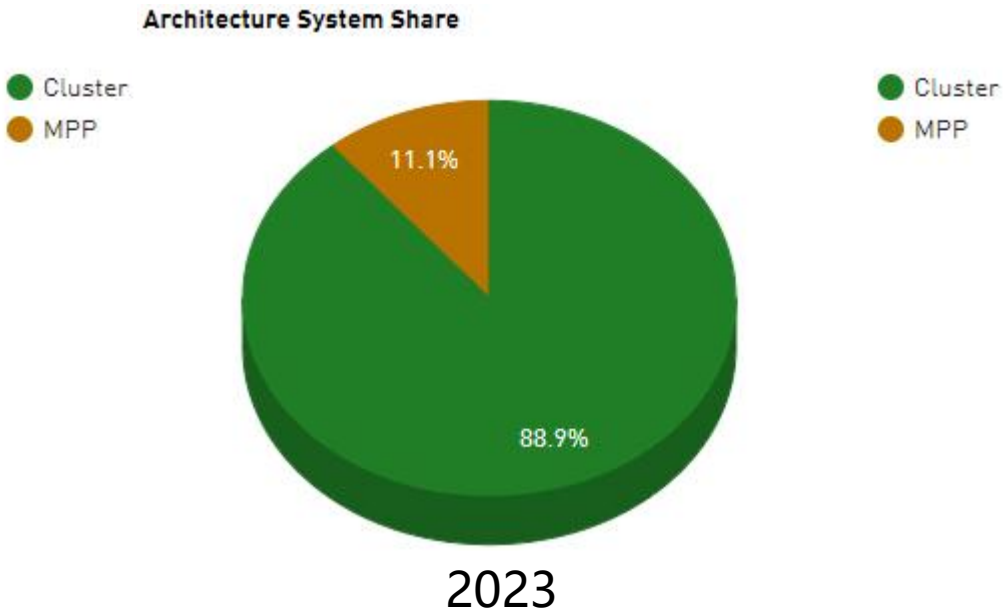
2024

并行计算发展

- Top500趋势：超算架构对比
  - Cluster架构占据主导地位
  - 功能和架构限制之下，MPP架构超算占比逐渐降低



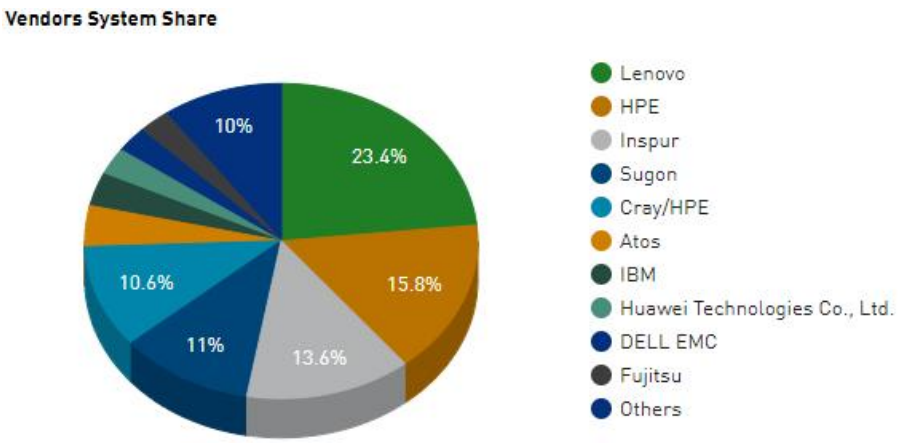
2018



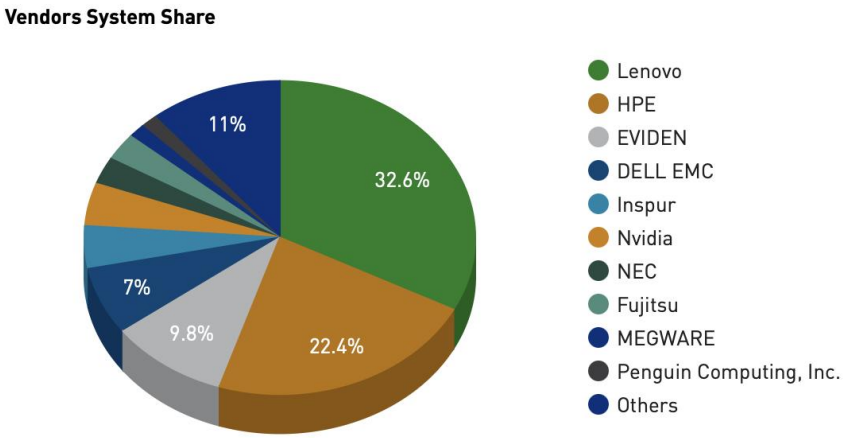
2023

并行计算发展

- Top500趋势：制造商对比
  - 承接来自国家、科研机构和顶尖企业的需求
  - 榜单中超算制造商，中国品牌市场份额快速提升



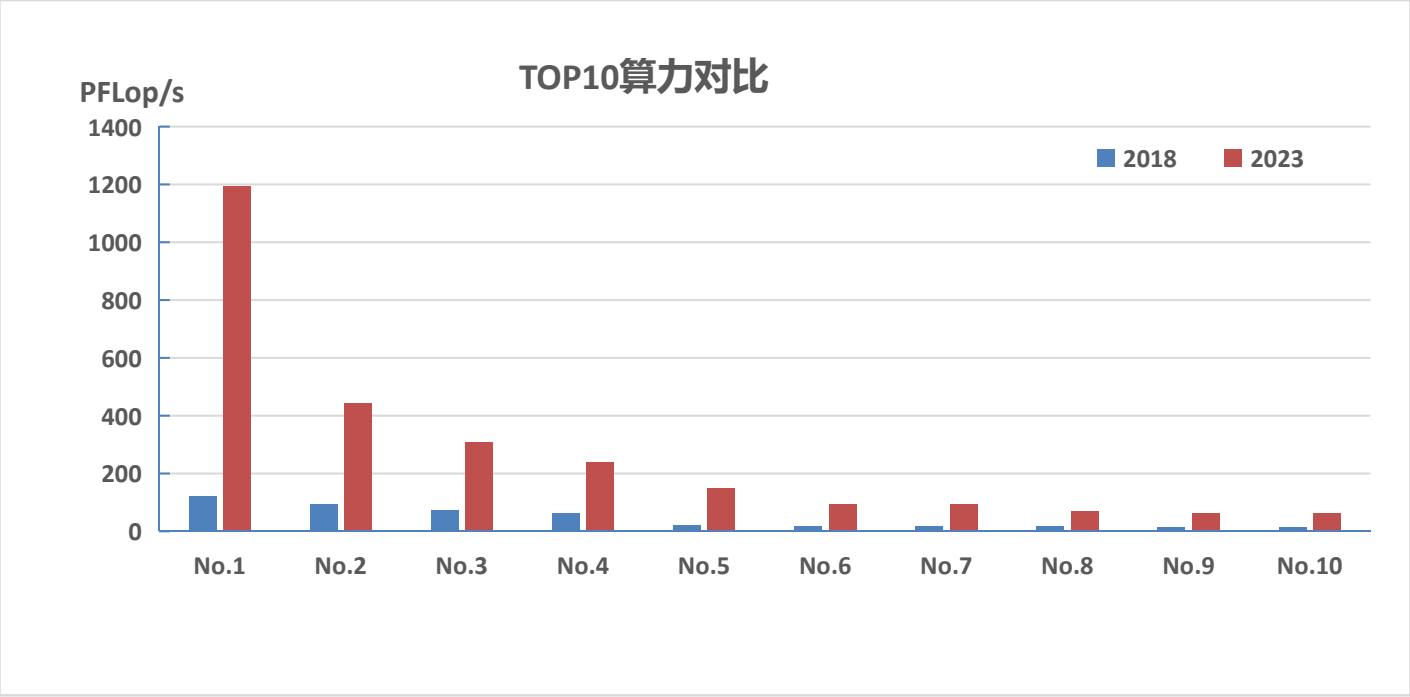
2018



2024

并行计算发展

- Top500趋势：百亿亿次（EFlop）
  - 5年跨度的两张TOP10榜单对比，4-10倍的算力增长
  - E级计算时代已经到来



并行计算发展

- **全球各大经济体，E级超算计划**

天河三号：采用Matrix 3000片上异构众核处理器, 已投入使用, FP64算力>1EFLop/s

神威E级：采用SW39000片上异构众核处理器, 已投入使用, FP64算力>1EFLop/s

曙光E级：采用x86架构的海光处理器和曙光DCU

JUPITER: 欧盟的超算机构EuroHPC JU计划在德国Jülich建造基于GPU的E级超算, 还在招标中

欧洲第二台E级超算(还未命名)计划部署在法国

Frontier：由AMD和HPE CRAY共同研发制造，用户同样为美国能源部, 已投入使用

Aurora：由Intel和HPE CRAY共同研发制造，用户为美国能源部阿贡实验室

El Capitan：由AMD和HPE CRAY共同研发制造，用户为美国能源部劳伦斯利弗莫尔实验

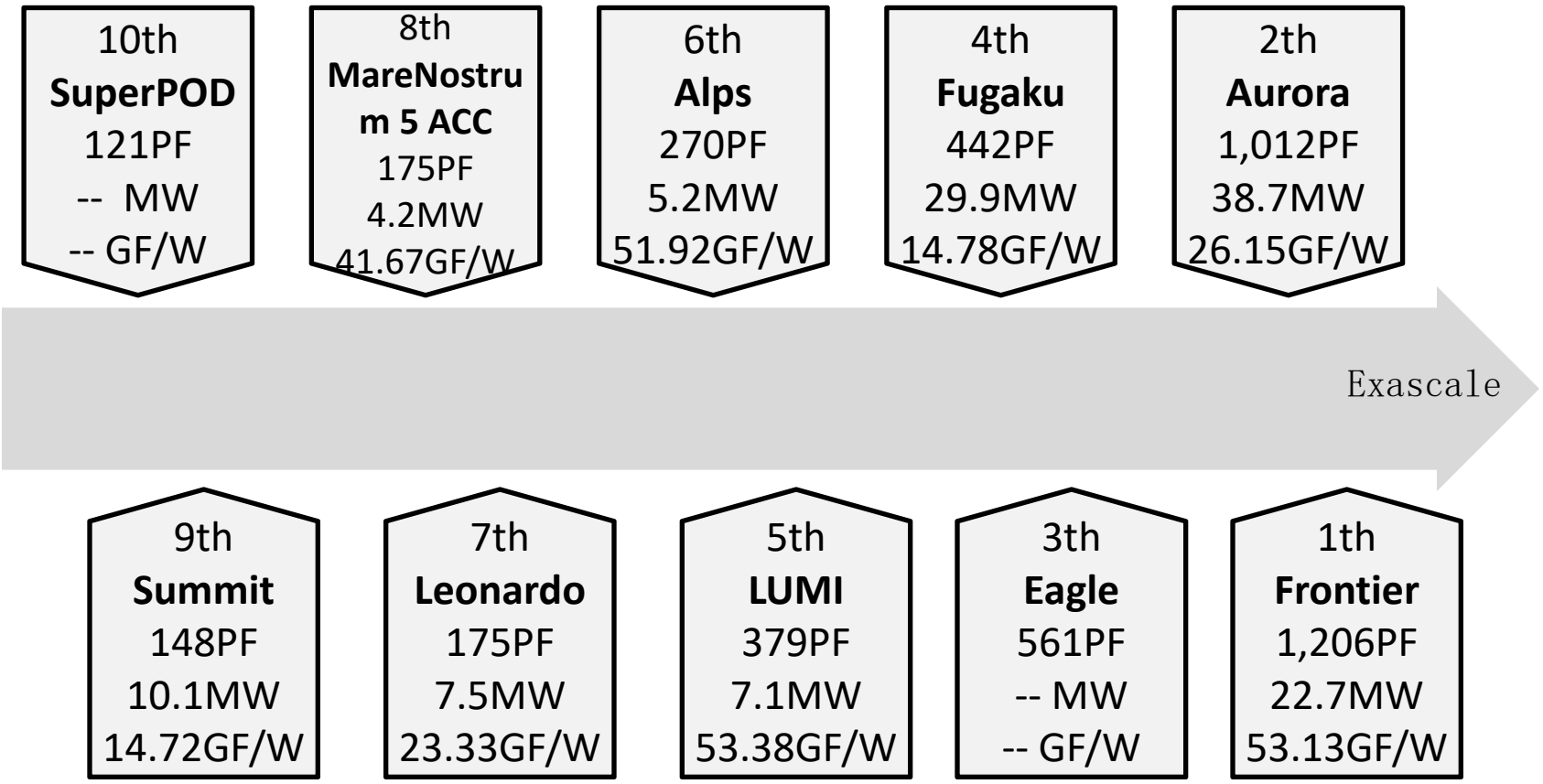
Post-K（后“京”）：作为日本超算“京”的后续产品，Post-K将采用目前已经成功部署的富士通A64FX处理器。Post-K计算节点原型已经开发完成，I/O及计算节点有48个核心外加4个辅助核心

目录

- 什么是并行计算
- 为什么需要并行计算
- 并行计算发展
- **并行计算面临的挑战**

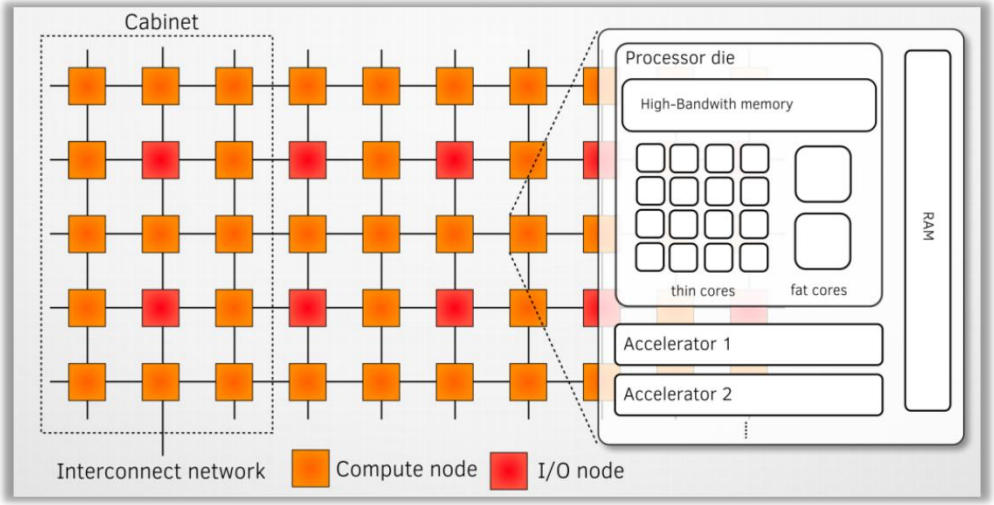
并行计算面临的挑战

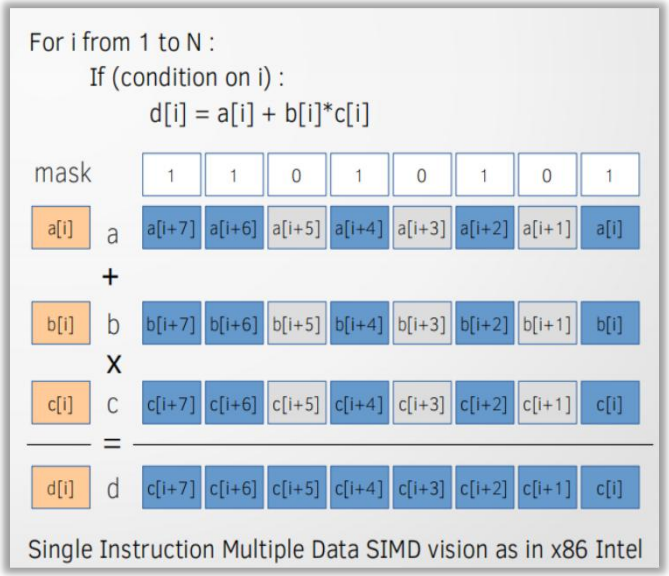
- **功耗（Power）**
  - 超算耗电功率巨大, 如何降低功率提高能效, 是算力增长的主要挑战



并行计算面临的挑战

- **应用性能（Performance）**
  - 追求应用可获得的性能而不是峰值性能
  - 实际应用性能经常在10%甚至5%的峰值之下





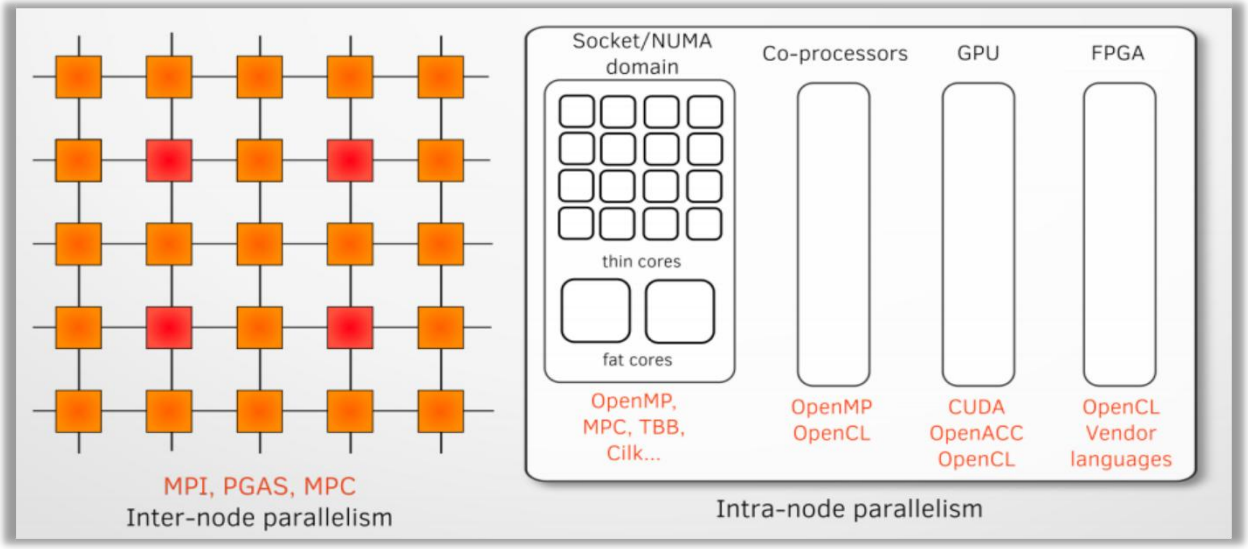
节点间并行：Complex software stack required to handle huge traffic and failure (losseless compression, resilence, huge pages, RMA...)

节点内并行：Heterogeneous computing with different accelerators, memory space and bandwidth

向量化：Capability to compute simultaneously a full vector of data with the same set of operations

并行计算面临的挑战

- **可编程性（Programmability）**
  - **大规模并行和异构体系结构给并行编程带来巨大困难**
  - **并行程序编程难，调试难，性能不确定**



①、MPI + X (OpenMP, Open ACC...)

②、Limit data movements, avoid global communications, use shared memory within a node

①、Use non-blocking comunication with computation

②、Dedicate cores to the communications, diagnostic processing, I/O

③、Use accelerator in symmetric mode to exploit the full node computational power

并行计算面临的挑战

- **可靠性（Resilience）**
  - 巨大的系统规模使得系统的平均无故障时间大大缩短，甚至在1小时以下
  - 如何完成长时间不间断运行的应用？

卡内基梅隆大学根据美国洛斯阿拉莫斯国家实验室（Los Alamos National Labs，LANL）22 台超级计算机长达 9 年的故障数进行统计

Category	Hardware	Software	Network	Environment	Human	Unknown
Failure Percentage	30-60%	5-24%	<3%	<3%	<3%	20-30%
Downtime Percentage	40-80%	3-25%	<2%	<5%	<3%	<10%

硬件故障和软件故障比例较大，由网络、环境因素和人为因素故障比例较小

由于超级计算机本身结构复杂，故障成因较多，同时故障类别本身鉴别起来较为困难，因此，有相当一部分故障没有给出明确故障原因

Reference

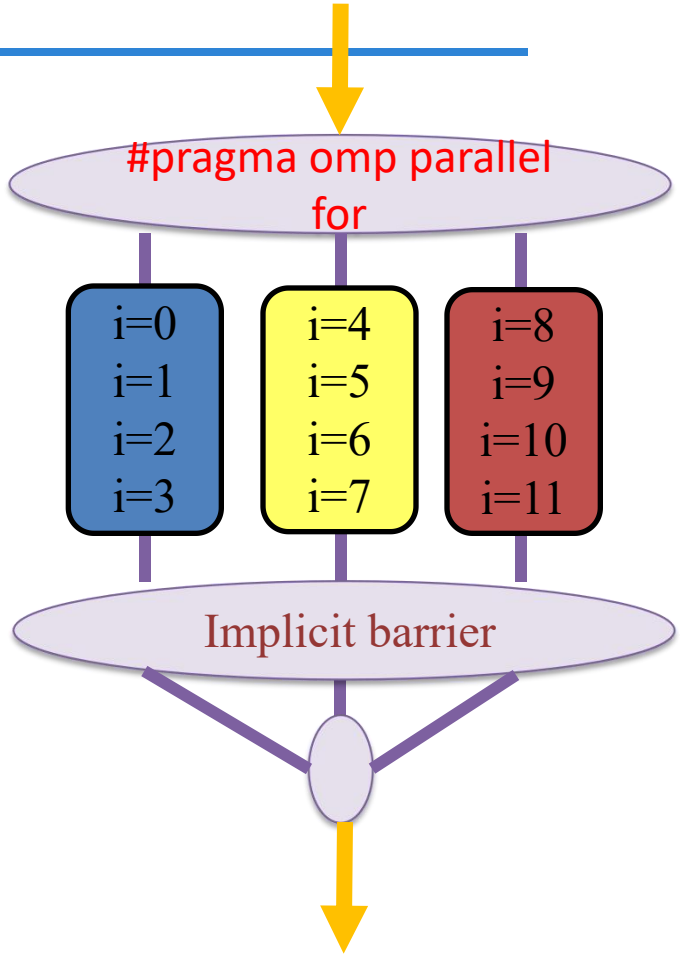
- Parallel Programming course slides from Prof. Jerry Chou, National Tsing Hua University. http://lms.nthu.edu.tw/sys/read_attach.php?id=1530893
- TOP500: https://www.top500.org/
- Blaise Barney, Lawrence Livermore National Laboratory, Introduction to Parallel Computing, https://computing.llnl.gov/tutorials/parallel_comp/

并行编程实践

- OpenMP示例

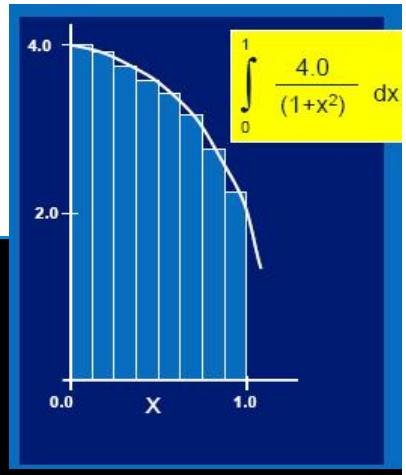
```
for (int i = 0; i < 12; i++)
        c[i] = a[i] + b[i];
```

```
#pragma omp parallel for
for (int i = 0; i < 12; i++)
        c[i] = a[i] + b[i];
```



并行编程实践

- OpenMP示例




```
[zqybegin@ecs-zqy openmp]$ vim serial_pi.c
[zqybegin@ecs-zqy openmp]$ gcc serial_pi.c -o serial_pi.bin
[zqybegin@ecs-zqy openmp]$ ./serial_pi.bin
The value of PI is  3.141592653590
The time to calculate PI was 7.006011 seconds
```

```
[zqybegin@ecs-zqy openmp]$ ./openmp_pi.bin
The value of PI is  3.141592653590
The time to calculate PI was 1.768647 seconds
```

```c
#include <stdio.h>
#include <time.h>

static long num_steps=2000000000;
double step, pi;

void main()
{
        clock_t start, stop;
        int i;
        double x, sum = 0.0;

        step = 1.0/(double) num_steps;
        start = clock();
        for (i=0; i< num_steps; i++){
                x = (i+0.5)*step;
                sum = sum + 4.0/(1.0 + x*x);
        }
        pi = step * sum;
        stop = clock();
        printf("Pi = %15.12f\n",pi);
        printf("The time to calculate PI was %f seconds\n",((double)(stop - start)/CLOCKS_PER_SEC));
}
```

```c
#include <stdio.h>
#include <time.h>
#include <omp.h>

static long num_steps=2000000000;
double step, pi;

void main()
{
        double start, stop;
        int i;
        double x, sum = 0.0;

        step = 1.0/(double) num_steps;
        start = omp_get_wtime();
        #pragma omp parallel for private(x) reduction(+:sum)
        for (i=0; i< num_steps; i++){
                x = (i+0.5)*step;
                sum = sum + 4.0/(1.0 + x*x);
        }
        pi = step * sum;
        stop = omp_get_wtime();
        printf("Pi = %15.12f\n",pi);
        printf("The time to calculate PI was %f seconds\n",((double)(stop - start)));
}
```

```
gcc serial_pi.c -o serial_pi.bin
```

```
gcc -fopenmp parallel_pi.c -o
openmp_pi.bin
```

并行编程实践

■ MPI示例

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <mpi.h>

int prime_part ( int id, int p, int n );

int main ( int argc, char *argv[] )
{
        MPI_Init(&argc, &argv);
        int id, p, n;
        MPI_Comm_rank(MPI_COMM_WORLD, &id);
        MPI_Comm_size(MPI_COMM_WORLD, &p);
        if(id == 0)
        {
                printf("Input N Value: ");
                fflush(stdout);
                scanf("%d", &n);
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

        int total = 0;
        int total_part = 0;

        double time1 = MPI_Wtime();
        total_part = prime_part(id, p, n);
        MPI_Reduce(&total_part, &total, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
        double time2 = MPI_Wtime();
        if(id == 0)
        {
                printf ( "Between 2 and %d, there are %d primes\n", n, total );
                printf ( "Computation Time %.10lf\n",time2 - time1 );
        }
        MPI_Finalize();

        return 0;
}
```

```c
int prime_part(int id, int p, int n)
{
        int i;
        int j;
        int prime;
        int total_part;

        total_part = 0;

        for ( i = id * 2 + 1; i <= n; i += p*2 )
        {
                prime = 1;

                for ( j = 2; j < i; j++ )
                {
                        if ( i % j == 0 )
                        {
                                prime = 0;
                                break;
                        }
                }
                if ( prime )
                {
                        total_part = total_part + 1;
                }
        }

        return total_part;
}
```

```
mpicc prime-mpi.c -o prime-mpi.bin
```

```
mpirun -np 6 ./prime-mpi.bin
```

并行编程实践

■ MPI示例

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <mpi.h>

int prime_part ( int id, int p, int

int main ( int argc, char *argv[] )
{
        MPI_Init(&argc, &argv);
        int id, p, n;
        MPI_Comm_rank(MPI_COMM_WORLD
        MPI_Comm_size(MPI_COMM_WORLD
        if(id == 0)
        {
                printf("Input N Valu
                fflush(stdout);
                scanf("%d", &n);
        }
        MPI_Bcast(&n, 1, MPI_INT, 0,

        int total = 0;
        int total_part = 0;

        double time1 = MPI_Wtime();
        total_part = prime_part(id,
        MPI_Reduce(&total_part, &tot
        double time2 = MPI_Wtime();
        if(id == 0)
        {
                printf ( "Between 2
                printf ( "Computatio
        }
        MPI_Finalize();

        return 0;
}
```

运行时间（T/s）统计表

进程数 / N	1	2	4	6
100000	1.053	0.528	0.270	0.179
200000	3.959	1.996	1.008	0.678
400000	14.951	7.504	3.778	2.534
800000	56.615	28.434	14.268	9.661

加速比（S_p）统计表

进程数 / N	1	2	4	6
100000	1.00	1.99	3.90	5.88
200000	1.00	1.98	3.93	5.84
400000	1.00	1.99	3.96	5.90
800000	1.00	1.99	3.97	5.86

```
2 )

t + 1;
```

```
ime-mpi.bin
```

```
mpirun -np 6 ./prime-mpi.bin
```