并行计算基础

崔 涛

tcui@lsec.cc.ac.cn

科学与工程计算国家重点实验室
中科院计算数学与科学工程计算研究所

线性方程组

《九章算术》[1]卷8第1题

有上禾三秉，中禾二秉，下禾一秉，实三十九斗；
上禾二秉，中禾三秉，下禾一秉，实三十四斗；
上禾一秉，中禾二秉，下禾三秉，实二十六斗。
问上、中、下禾实一秉各几何？

$$\begin{cases} 3x + 2y + z &= 39 \\ 2x + 3y + z &= 34 \\ x + 2y + 3z &= 26 \end{cases} \tag{1}$$

[1] 《九章算术》是中国古代数学专著,大约于东汉初年（公元1世纪）成书

线性方程组求解

直接法

经过有限次运算后可求得方程组的精确解的方法。 example

1. 张苍、刘徽《九章算术》，“直除法”[a]
2. 高斯消元法(1800年)

[a] 比欧洲早了1700多年

迭代法

从解的某个近似出发，通过构造无穷序列去逼进精确解的方法。（有限步内不一定能得到精确解）

1. Jacobi迭代 example
2. Gauss-Seidel迭代 example

计算量与存储量分析

高斯消元法

- 计算量（乘除法）$\mathcal{O}(n^3)$ [2]
- 存储量$\mathcal{O}(n^2)$

一般迭代法

- 计算量（乘除法）$\mathcal{O}(n^2 * k)$ [3]
- 存储量: $\mathcal{O}(n)$（稀疏矩阵）; $\mathcal{O}(n^2)$（稠密矩阵）

[2] n表示未知量个数

[3] k表示迭代步数

计算机主要技术指标[4]

运算速度

理论峰值速度=**CPU**主频×每个时钟周期执行的浮点运算次数
常见**CPU**每时钟周期执行的浮点运算次数:

处理器类型	flops/cycle	处理器类型	flops/cycle
Intel Pentium 4 IA32	1	Intel Pentium Xeon	2
Intel Itanium 2	4	AMD Opteron	4
SGI MIPS	2	SUN Ultra-Sparc	2
HP Alpha	2	IBM PowerPC	4

例子

普通家用3.4GHz Pentium4电脑理论峰值：$3.4 \times 2 = 6.8$Gflops。
当线性方程组$n = 10^9$,求解时间

- 直接法：$\mathcal{O}(10^{17})$秒
- 迭代法：$\mathcal{O}(10^8 * k)$秒

[4] 技术指标包括：字长、存储容量、运算速度、外设、接口标准与类型、软件配置、可靠性

计算机主要技术指标[5]

存储容量

在64位计算机中，一个双精度浮点数占用8 byte（字节），整型数占用4 byte（字节）。

例子

普通家用电脑一般内存容量为2 GB 当线性方程组$n = 10^9$,需要内存

- 直接法：$\mathcal{O}(10^9)$GB
- 迭代法：$\mathcal{O}(10)$GB（稀疏矩阵）、$\mathcal{O}(10^9)$GB（稠密矩阵）

[5] 技术指标包括：字长、存储容量、运算速度、外设、接口标准与类型、软件配置、可靠性

问题与挑战

全球气候预报

在24小时内完成48小时天气预报，至少需要计算635万个网格点，内存需求大于1TB,计算性能要求高达25Tflops/s。

芯片设计

假设我们计算区域为$10\mu m \times 10\mu m \times 10\mu m$,光波波长为$50nm$。每个波长至少需要用5个网格点刻画，因此网格规模为1,000,000,000。未知量个数为$\mathcal{O}(10^9)$

高性能计算目的

高性能计算目的：算的规模更大，算的速度更快

问题与挑战

全球气候预报

在24小时内完成48小时天气预报，至少需要计算635万个网格点，内存需求大于1TB,计算性能要求高达25Tflops/s。

芯片设计

假设我们计算区域为$10\mu m \times 10\mu m \times 10\mu m$,光波波长为$50nm$。每个波长至少需要用5个网格点刻画，因此网格规模为1,000,000,000。未知量个数为$\mathcal{O}(10^9)$

高性能计算目的

高性能计算目的：算的规模更大，算的速度更快

并行计算误区

- 必须：认为所有应用都要并行
- 万能：所有的算法都能从并行计算中得到好处
- 酷：为了并行而并行

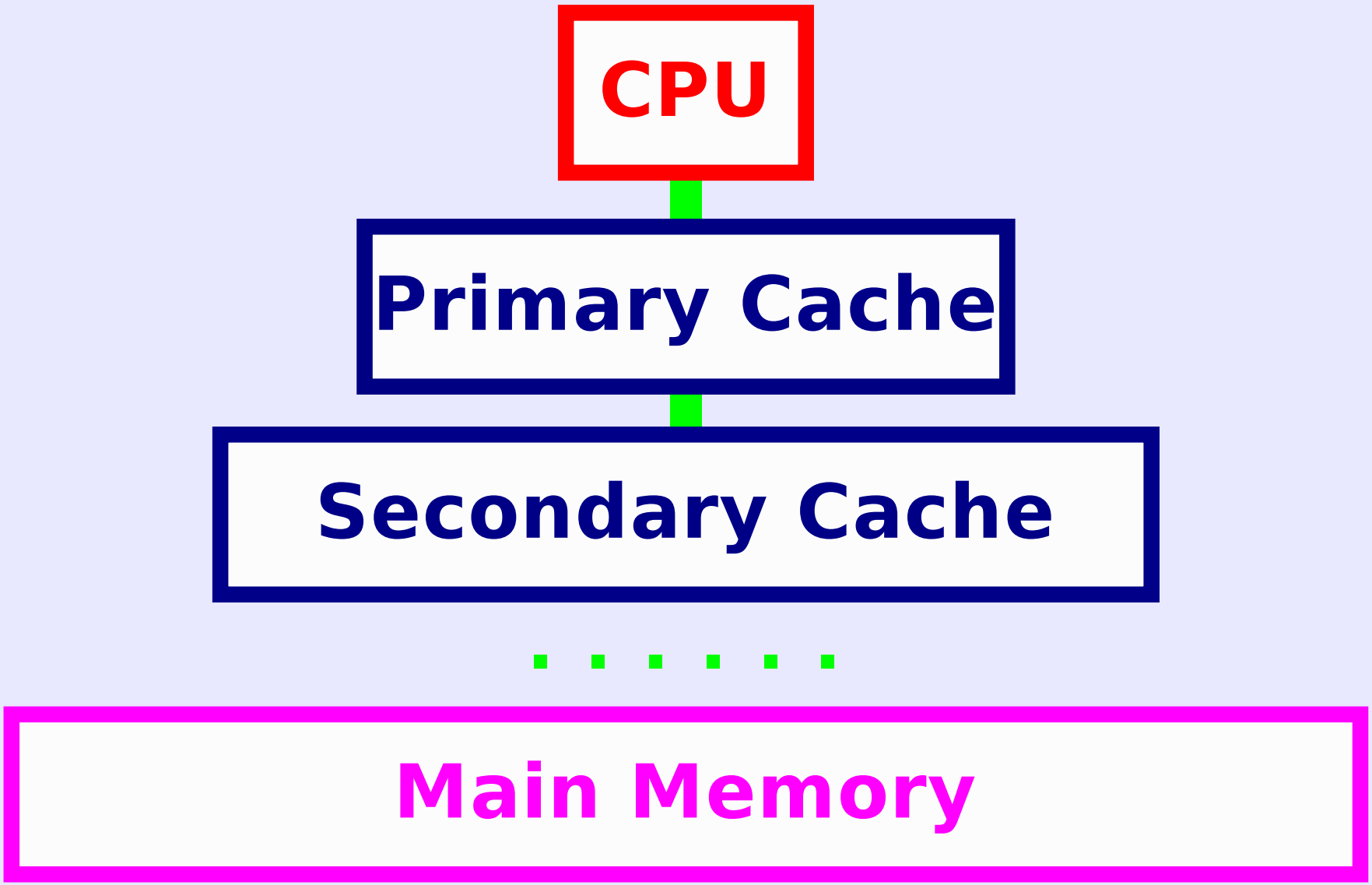
并行计算的主要研究方向

- 高性能计算机系统(硬件):
  - 计算机
  - 微电子
  - ... ...
- 并行算法:
  - 数学
  - 物理
  - 计算机
  - ... ...
- 并行程序设计,应用以及支撑软件:
  - 数学
  - 计算机
  - ... ...
- 并行计算性能评测

Agenda

1. 高性能并行计算机系统简介
2. 并行算法
3. 并行编程模式
4. Unix程序开发简介

微处理器的存储结构



Cache结构对程序性能的影响

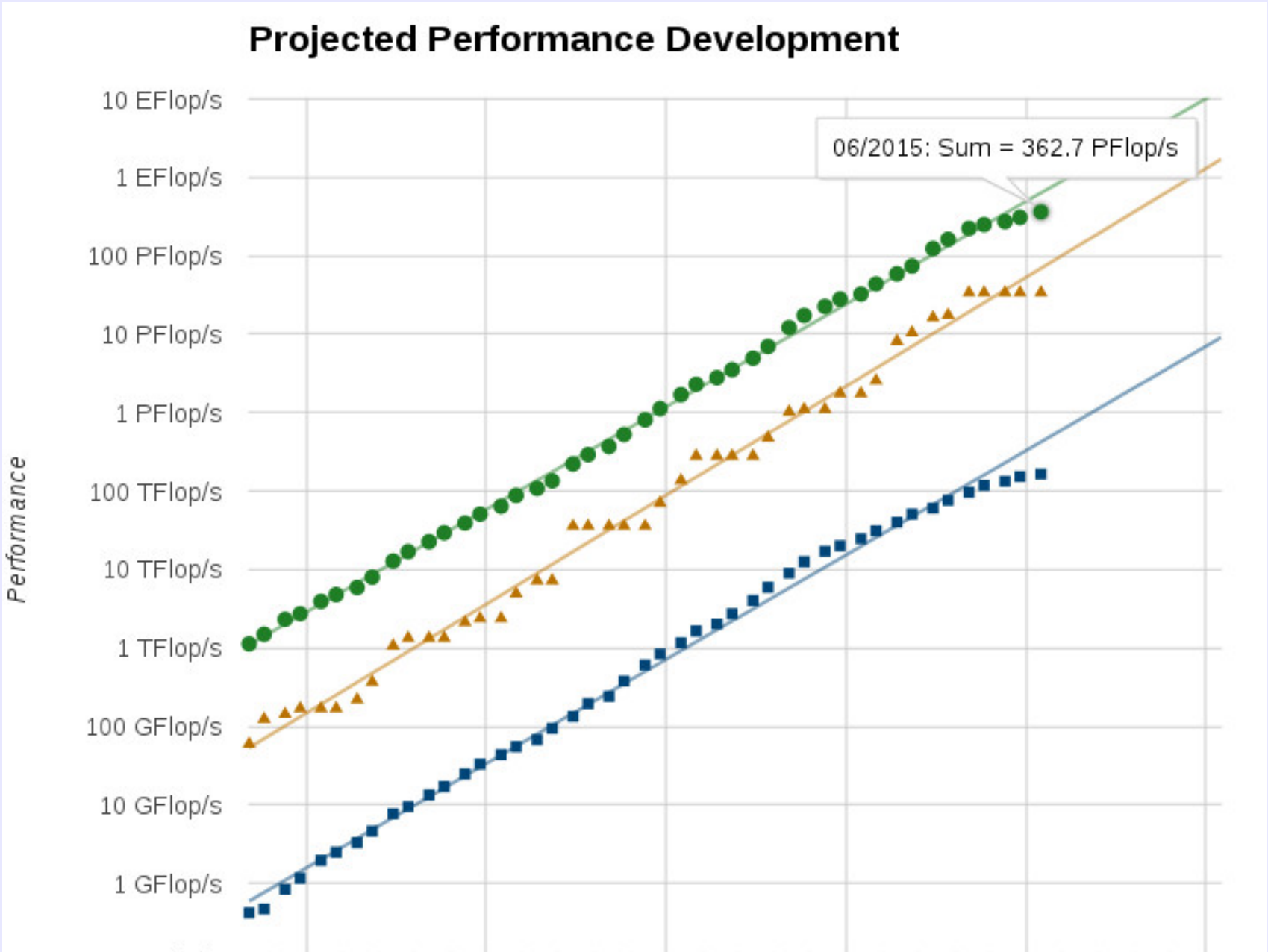
矩阵乘法中不同循环顺序对程序性能的影响.

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j} \qquad i,j = 1,\ldots,n$$

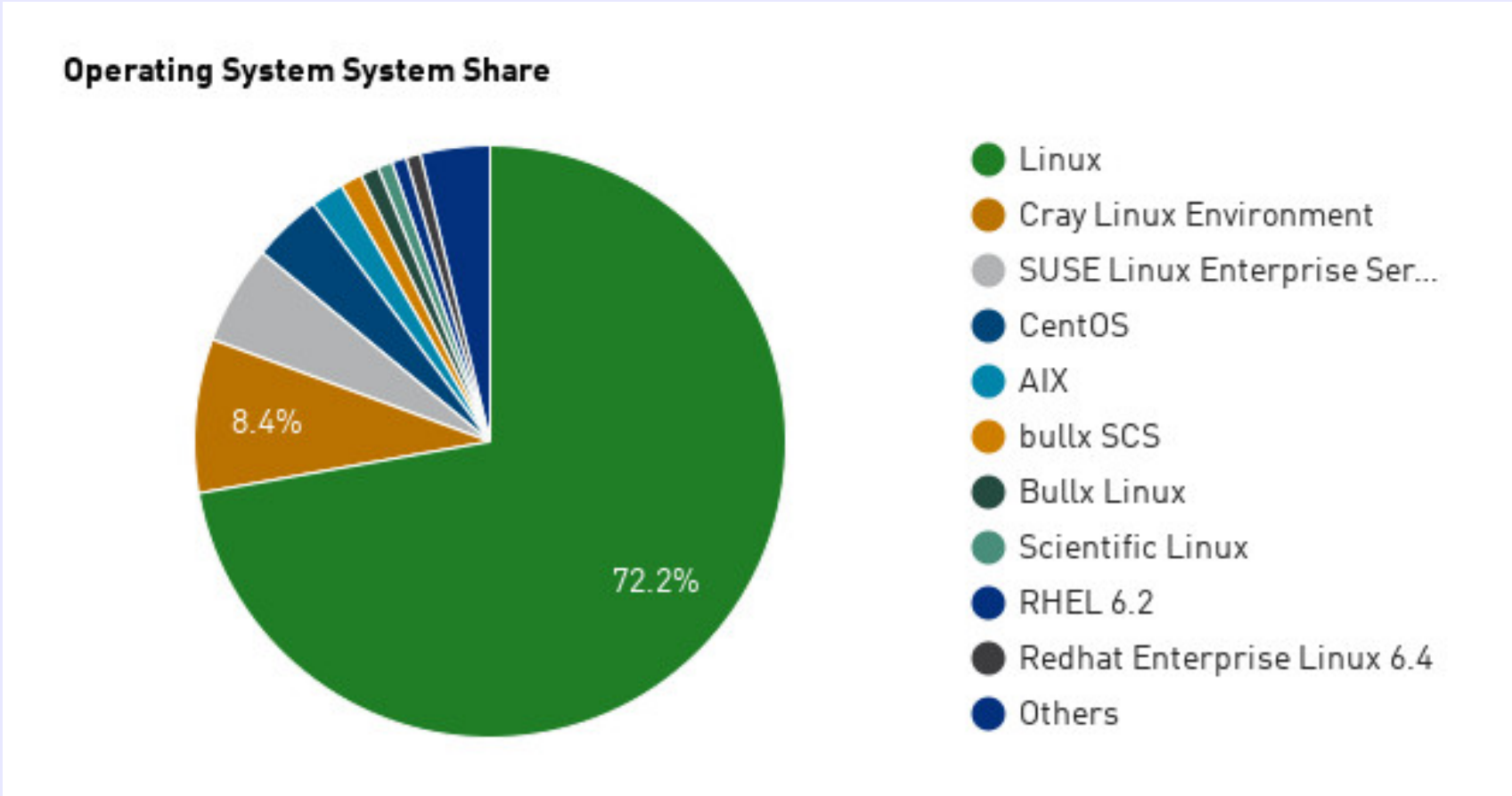
```
DO J=1,N
   DO I=1,N
      C(I,J) = 0.DO
   ENDDO
ENDDO
DO I=1,N
   DO J=1,N
      DO K=1,N
         C(I,J) = C(I,J) + A(I,K) * B(K,J)
      ENDDO
   ENDDO
ENDDO
```

example: matmul.c

高性能计算机的发展[6]


Projected Performance Development
06/2015: Sum = 362.7 PFlop/s
Performance
10 EFlop/s
1 EFlop/s
100 PFlop/s
10 PFlop/s
1 PFlop/s
100 TFlop/s
10 TFlop/s
1 TFlop/s
100 GFlop/s
10 GFlop/s
1 GFlop/s

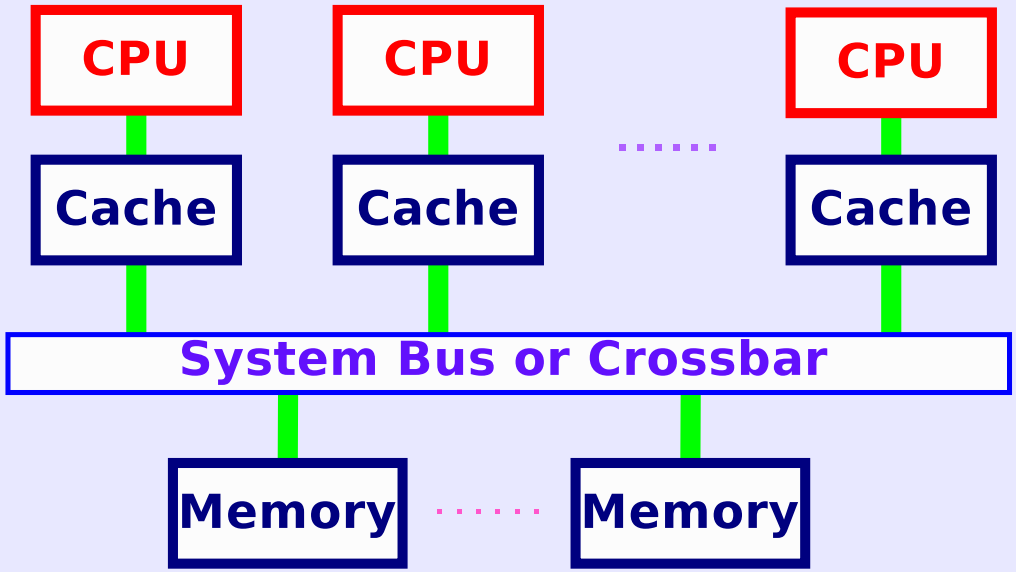
高性能计算机的操作系统[7]



[7] 资料来源http://www.top500.org

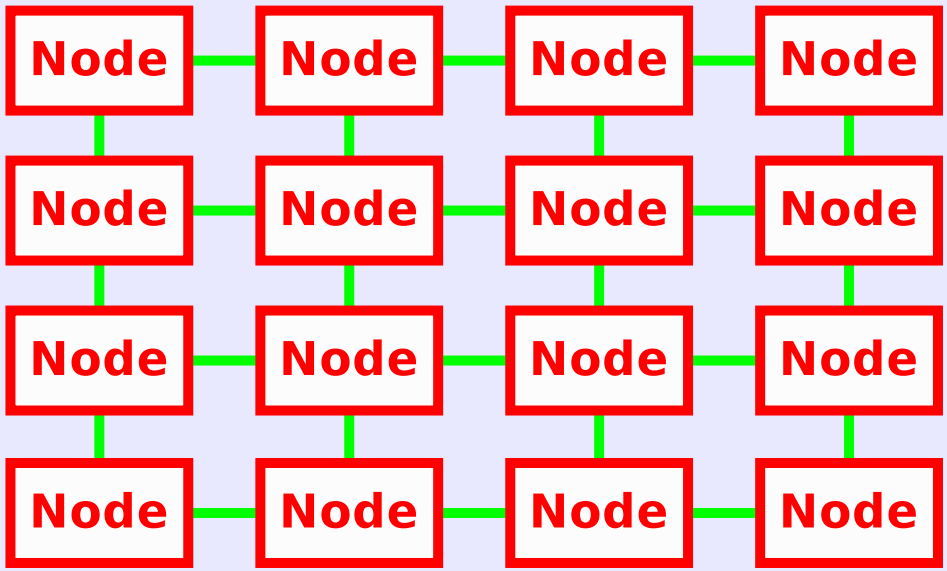
共享内存SMP型并行计算机

- 对称多处理器(Symmetric Multi–Processors), 或共享内存处理器(Shared Memory Processors).
- 多个处理器通过系统总线或交叉开关共享一个或多个内存模块.
- 优点: 使用简单, 维护方便.
- 缺点: 受系统总线带宽限制, 只能支持少量处理器(一般十几个).
- 并行编程方式: 通常采用OpenMP, 也可使用消息传递(MPI/PVM)及HPF.
- 代表机型: SGI Power Challenge, Sun E10000, 等.



分布式内存MPP型并行计算机

- Massively Parallel Processors的简称
- 指由大量具有局部内存的计算结点通过高速系统网络联接而构成的并行处理系统.
- MPP系统的系统网络通常具有某种拓扑结构(如tree, mesh, torus, hypercube).



DSM型并行计算机

分布共享内存: Distributed Shared Memory

- 多个物理上具有独立内存的处理单元, 通过高速网络联接在一起.
- 逻辑上作为共享内存并行机使用.
- 也称为NUMA结构(NonUniform Memory Access).
- 不同处理单元间内存的共享通过特殊的硬件/软件实现.
- 具有比SMP型并行机更好的可扩展性(超过100个CPU).
- 代表机型: SGI Origin 2000/3000.

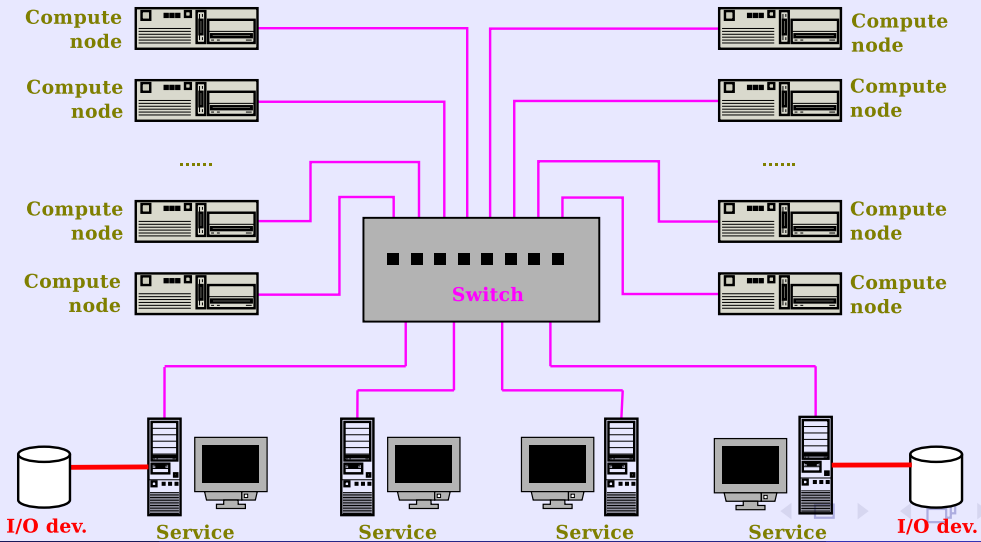
SMP/DSM机群

- 将多台SMP或DSM并行机通过互联网络连接而成.
- 目前国内外最高性能的并行机大多是这种结构.

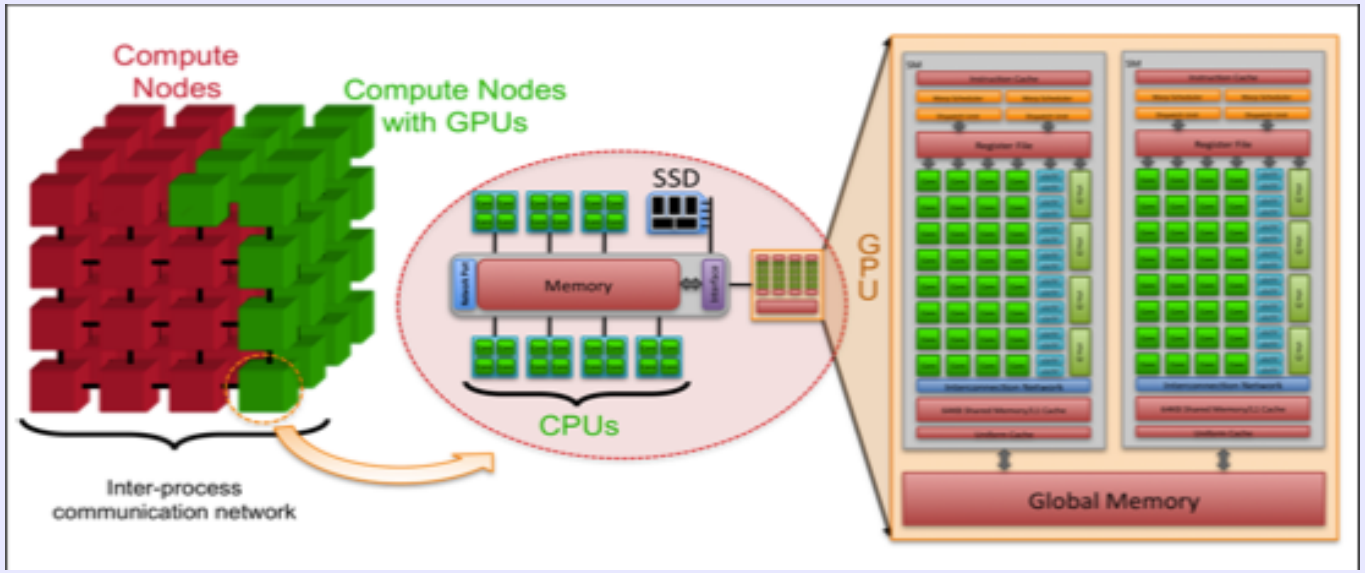
微机/工作站机群

- 微机机群(PC cluster, 又称为Beowulf cluster), 工作站机群(NOW, Network Of Workstations): 将联网的多台微机或工作站组织成一台并行计算机.
- 适合于构造中等规模的并行系统(多达数百个处理器).
- 根据机群中所使用的机型可为同构型和异构型两种.
- 根据机群的使用方式又可分为专用型和兼用型.
- 微机/工作站机群的优点是价格便宜、配置灵活.

配以适当的系统管理工具及作业调度、性能监控、并行程序调试开发环境等, 微机/工作站机群系统可以达到与商用MPP系统一样的使用效果.



Heterogeneous HPC



Agenda

1. 高性能并行计算机系统简介
2. 并行算法
3. 并行编程模式
4. Unix程序开发简介

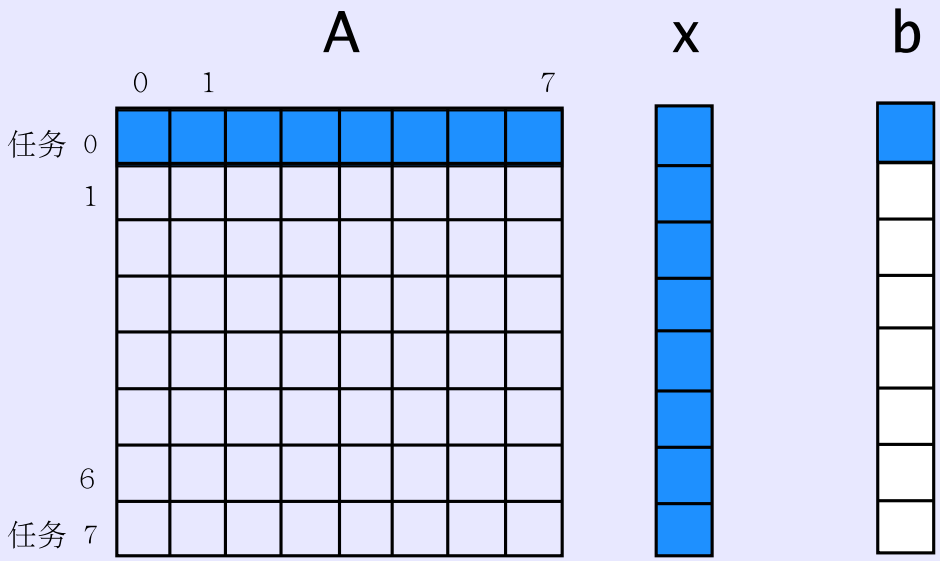
并行算法

- 什么是串行算法？
  解题方法的精确描述，是一组有穷规则，规定了解决某一特定类型问题的一系列运算。这些运算可在计算机上由单个进程按一定次序完成。
- 什么是并行算法？
  解题方法的精确描述，是一组有穷规则，规定了解决某一特定类型问题的一系列运算。这些运算可在并行计算机上由多个进程同时按一定的次序完成。
- 与串行算法之间的主要区别：
  - 基于一定的并行计算模型；
  - 需要考虑并行执行的各进程之间的关系。

矩阵向量乘法

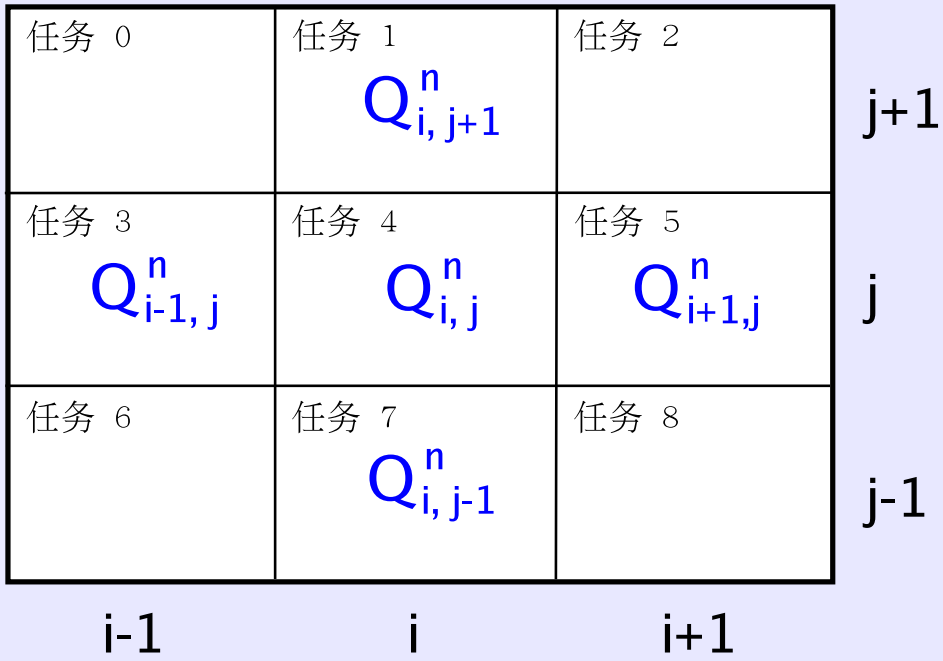
$$Ax = b$$

$$b[i] = \sum_{j=0}^{8} A[i][j]x[j], i = 0, 1, \cdots, 8$$



五点差分格式

$$Q_{i,j}^{n+1} = \frac{Q_{i-1,j}^{n} + Q_{i+1,j}^{n} + Q_{i,j-1}^{n} + Q_{i,j+1}^{n}}{4}$$



并行算法的发展历程

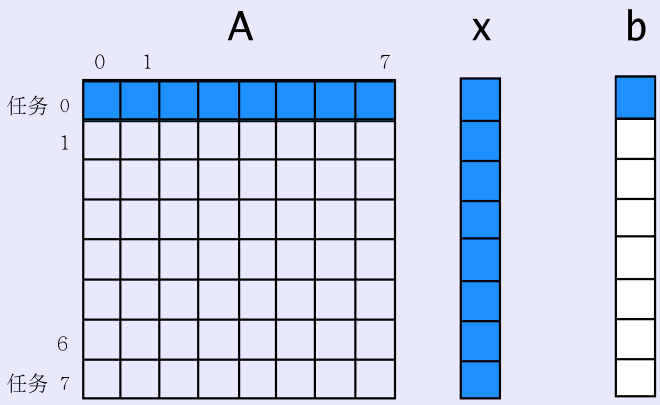
“并行算法”源于所求解问题内在的并发性

1. 并行算法的萌芽期（1972年以前）
   - 流水线结构和阵列式结构的计算机即将诞生
   - 针对科学计算中的偏微分方程求解
   - 理想化的并行模型
2. 并行算法的发展期（1972-1981）
   - SIMD并行机ILLIAC-IV诞生
   - MIMD并行机Cmmp问世
   - 同步并行算法发展、异步并行算法出现
3. 并行算法的成熟期（1981年以后）
   - 微电子技术和网络通信技术迅猛发展
   - 共享存储并行机、分布式存储并行机$\cdots$
   - OpenMP、PVM、MPI等并行编程环境
   - PETSc、UG、ScaLAPACK等

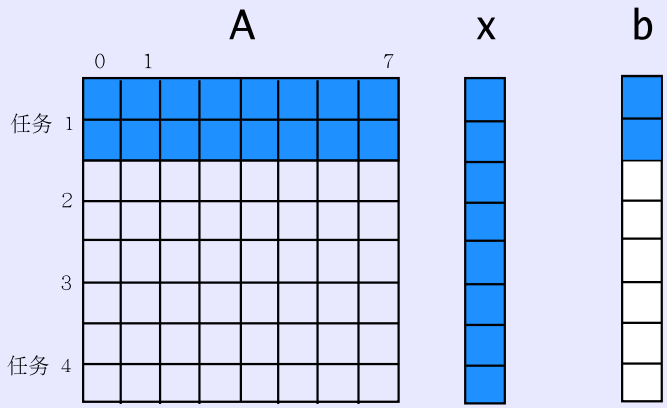
并行算法中的基本概念

- 任务划分：将大任务（问题）分解为小任务（问题）。任务的尺寸可用完成该任务的时间来刻画。划分任务的目的是发掘问题的并发性。
  - 数据划分：以待处理的数据为划分对象；
    划分原则：
    - 大数据块优先；
    - 访问频率高的数据块优先；
    - 划分出的数据块尺寸尽量相等。
  - 功能划分：以待完成的计算为划分对象；
    划分原则：
    - 尽可能少的数据重叠；
    - 尽可能少的额外计算。
  - 混合划分：结合使用上述两种方法。

- 并行粒度（Granularity）：描述任务的数量和尺寸。
  细粒度（fine-grained）：大量的小任务；
  粗粒度（coarse-grained）：少量的大任务。
- 并行度或并发度（degree of concurrency）：在指定时刻，可同时执行的任务的数量。

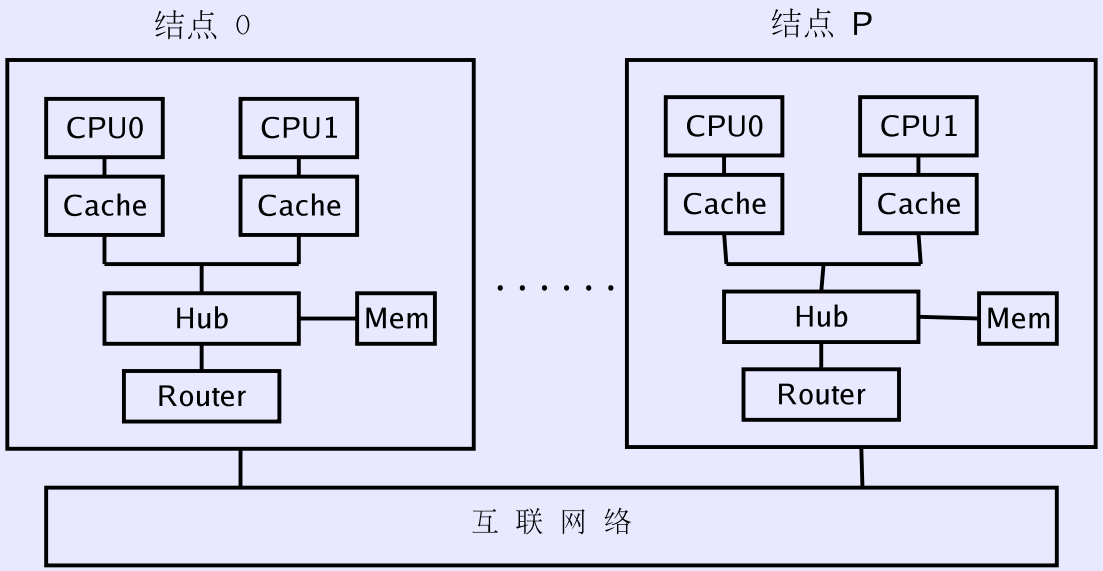


(a) 细粒度, 高并行度



(b) 粗粒度, 低并行度

- 进程（Process）：一段（串行）程序连同其数据在（单）处理机上的动态执行过程。每个进程拥有独立的内存地址空间。
- 线程（Thread）：由进程创建。属于同一进程的所有线程拥有相同的内存地址空间。
- 处理器（CPU）：晶体管、逻辑块、功能单元
- 结点（Node）：处理器、内存



- 并行性能：
  相对于串行而言，并行计算获得的收益。

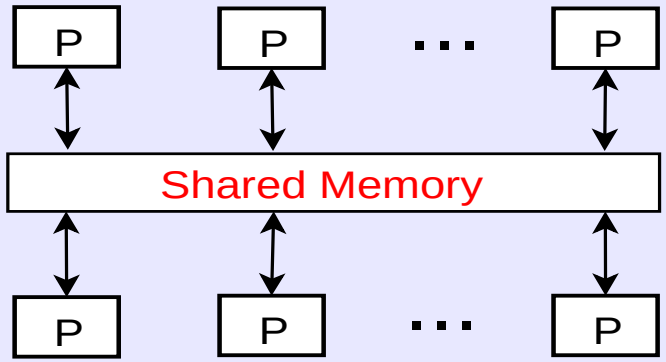
  并行程序的主要开销：
  - 每个进程完成的串行计算；
  - 由进程间交换数据产生的通信开销；
  - 由进程间相互等待产生的空闲；
  - 由于并行而引入的额外计算。

  并行性能的主要考察指标：
  - 加速比：串行执行时间与并行执行时间的比值；
  - 并行效率：加速比与进程数的比值,即$E_P = \frac{T_s}{P \times T_P}$
    - P 为并行程序执行进程数
    - T_P 为并行程序执行时间
    - T_S 为算法的串行实现执行时间
  - 可扩展性：并行效率与问题规模、处理机数目之间的关系。
    - 强可扩展性
    - 弱可扩展性

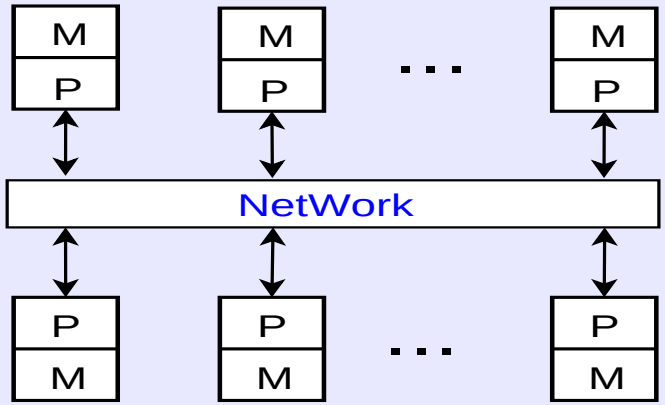
并行算法分类

- 数据并行—功能并行
- 数值并行算法—非数值并行算法
- 同步并行算法—异步并行算法—独立并行算法
- 细粒度—中粒度—粗粒度
- 共享内存—消息传递

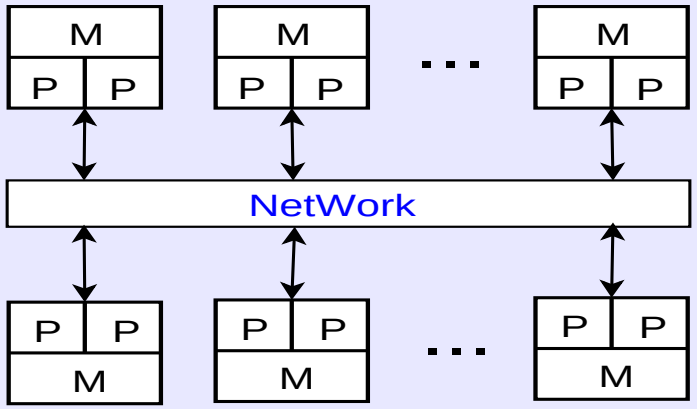
- 共享内存—消息传递



(c) 共享内存模型



(d) 消息传递模型



(e) 消息传递共享内存模型

并行算法设计过程

1. 分析物理（数学）问题，划分任务：
   - 任务尺寸尽量平均；
   - 任务间的数据通信尽可能少；
   - 任务数大于处理器数；
   - 任务数与问题规模成正比。
2. 分析任务之间的通信：
   - 尽可能少的全局通信；
   - 尽可能规则的通信；
   - 各任务的通信量尽量相当；
   - 各任务内部的通信和计算可同时执行。
3. 调整（合并、组合）任务：
   - 减少通信；
   - 减少冗余计算。
4. 将划分后的任务分配到所有进程（进程映射）：
   - 保证负载平衡。

Agenda

1. 高性能并行计算机系统简介
2. 并行算法
3. 并行编程模式
4. Unix程序开发简介

自动并行与手工并行

- 在SMP及DSM并行机上编译系统通常具有一定的对用户程序(C/Fortran) 进行自动并行化的能力, 但经常需要人工干预(通过指导语句、命令行选项等) 以达到理想的并行效率. 并行主要针对循环进行(细粒度并行).
- 在分布式内存并行机上尚无通用高效的自动并行工具, 主要依靠人工编写并行程序.
- 并行算法的设计及并行程序的编制已成为目前制约大规模并行计算机应用的主要障碍.

OpenMP

在串行程序的循环语句前插入特定的指导语句, 告知编译系统一些有助于对该循环进行并行的信息以及/或是强制编译系统按指定的方式将该循环并行化.

- 主要限于SMP及DSM型的并行系统. 现在也已发展到一些MPP系统.
- 通常结合编译系统的自动并行化功能使用.
- 也有一些自动并行化工具, 它们对程序结构及数据流进行分析, 然后自动插入适当的OpenMP语句.
- OpenMP的优点是学习及使用相对简单, 很多情况下不需要对原有算法及程序做大的改动. 缺点是只适合一类特定的机型, 并且程序的可扩展性通常不如用消息传递方式编写的并行程序.
- 对一些具有强数据相关性的计算过程需要通过改变计算顺序或修改算法甚至设计新的算法来实现并行计算.
- OpenMP程序实例: 矩阵乘

DSM编程模式

- 建立在某种内存一致性协议之上, 通过软件或软硬件结合的方式实现处理器间内存的共享.
- 通过要求DSM并行程序中对内存的读写遵循一定的规则来减小维护内存一致性的开销.
- 参看:
  http://lssc.cc.ac.cn/Parallel-programming/唐志敏/jiajia.ppt

高性能Fortran: HPF

- 基于数据并行的编程语言, 即用户通过指导语句指示编译系统将数据分布到各处理器上, 编译系统根据数据分布的情况生成并行程序.
- 主要面向Fortran 90/95.
- 优点: 编程相对简单, 串并行程序一致.
- 适合于SMP/DSM/MPP/cluster系统.
- 主要缺点是并行过程对用户的透明度不高, 程序的性能很大程度上依赖于所用的编译系统及用户对编译系统的了解, 需要针对不同的HPF编译器做不同的优化, 影响了程序的可移植性.
- HPF程序实例: 矩阵乘

消息传递并行编程模式

- 并行程序由一组独立运行的进程构成. 进程间通过相互发送消息来实现数据交换.
- 可以说消息传递并行编程是并行应用程序开发的最底层编程方式之一. 很多其它并行开发语言或工具(如一些HPF编译器)将程序转化成消息传递型并行程序来实现并行.
- 常用的消息传递平台有PVM (Parallel Virtual Machine) 和MPI (Message Passing Interface).
- 程序通用性好, 用PVM或MPI编写的程序可以在任何并行计算机上运行.
- 能够达到很高的并行效率, 具有很好的可扩展性.
- 缺点: 程序的编制与调试比较困难, 许多情况下要对程序甚至算法做大的改动.

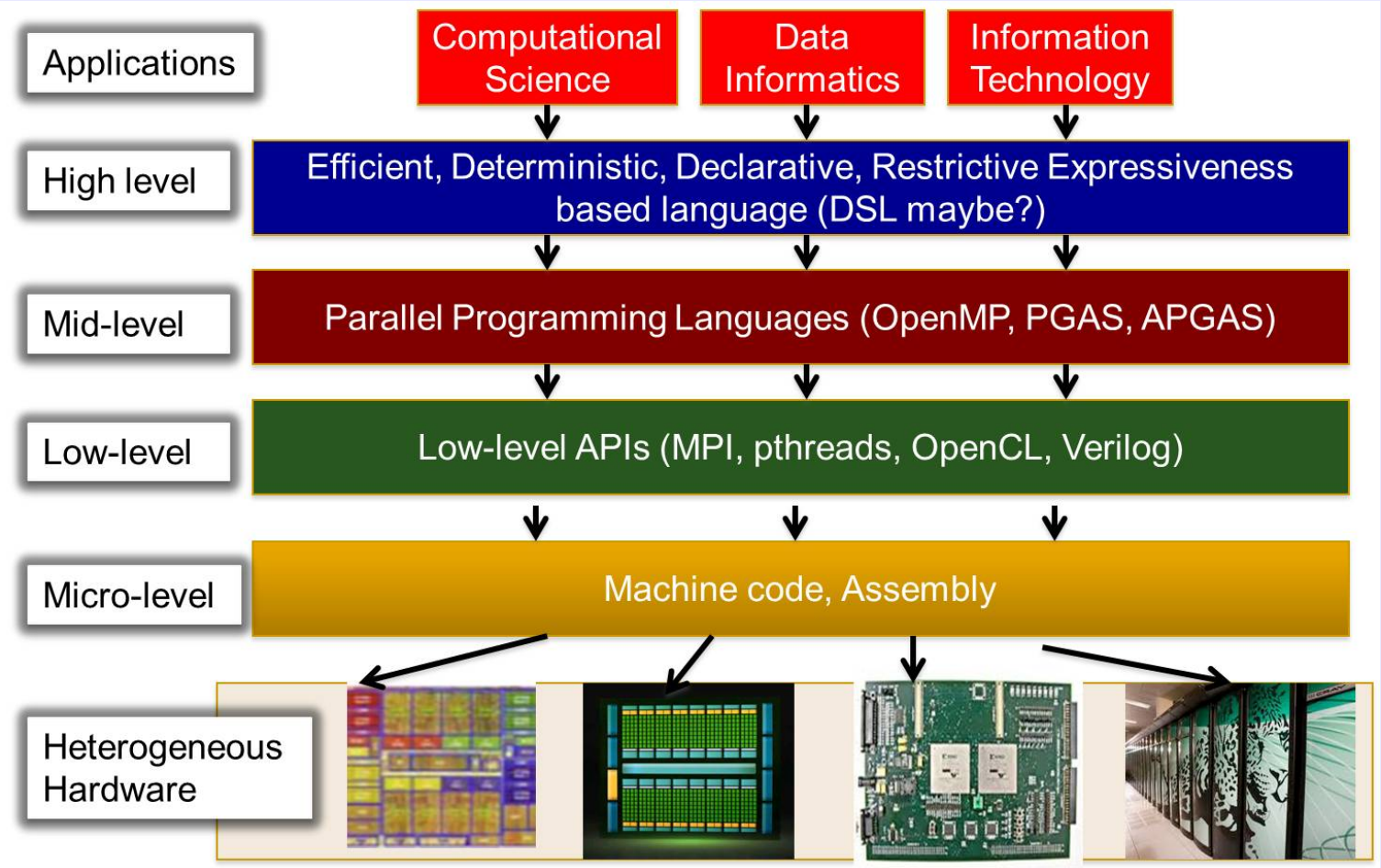
并行算法设计和编程的挑战

```
// Run one OpenMP thread per device per MPI node
#pragma omp parallel num_threads(devCount) if (initDevice())
 {
        // Block and grid dimensions
        dim3 dimBlock(12,12);
        kernel<<<1,dimBlock>>>();
        cudaThreadExit();
}
else
{
        printf("Device error on %s\n",processor_name);
}
        MPI_Finalize();
        return 0;
}
```

[8]

[8] www.cse.buffalo.edu/faculty/miller/Courses/CSE710/heavner.pdf

未来可能的编程模型



[9]

[9] From: Barbara Chapman,'Exascale: Why It is Different', P2S2 Workshop at ICPP

Agenda

1. 高性能并行计算机系统简介
2. 并行算法
3. 并行编程模式
4. Unix程序开发简介

Linux常用命令

- 切换目录：cd
- 列出当前目录：ls
- 删除文件：rm -f (filename)
- 创建目录：mkdir (dirname)
- 删除目录：rm -rf (dirname)
- 查看文件内容：more, cat
- 文件拷贝：cp (src) (dest)
- 目录拷贝：cp -r (src) (dest)
- 文本编辑器：nano（类似于记事本），(g)vim ，(X)emacs
- 帮助：man （命令名称）; info （命令名称）

编辑器vi

vi 基本概念

- 命令行模式(command mode): 控制屏幕光标的移动，字符的删除。
- 底行模式（last line mode）: 将文件保存或退出vi,……等。
- 插入模式（Insert mode）: 文字输入，按ESC键回到命令行模式。
- 可视模式(visual mode)

vi 基本操作

- 进入vi: vi myfile
- 切换至插入模式编辑文件: 在命令行模式下按一下字母i
- 退出vi及保存文件: 在命令行模式下，按一下冒号键进入底行模式
  - :w filename 将文章以指定的文件名filename保存）
  - :wq (存盘并退出vi)
  - :q! (不存盘强制退出vi)

远程登录

- Linux： ssh 159.226.92.64 -l pccourse
- Winodws 客户端:
  - putty pscp （下载后直接使用）
  - SSHSecureShellClient[10]
  - X-Server： Exceed , Xmanager

文件传输

- SSHSecureShellClient
- pscp
  - 上传： pscp -r localfile pccourse@159.226.92.64:/home/pccourse/
  - 下载： pscp -r pccourse@159.226.92.64:/home/pccourse/remotefile ./

[10] ftp://ftp.cc.ac.cn/

Unix中常用的编译系统

- 编译器由前端和后端组成. 通常用户只需使用前端命令即可完成编译、链接.
- C编译器: cc, gcc (GNU C) 等.
- Fortran 编译器: f77, fc, g77 (GNU Fortran), f90 (Fortran 90) 等.
- 可用man查看使用手册, 如: man cc, man f77等等.
- 命令行形式:

```
$cc [options] files [options]
$f77 [options] files [options]
```

Unix中常用的编译系统

- 文件的类型由文件的扩展名决定:
  - C源代码: .c;
  - Fortran 77源代码: .f;
  - 带预处理的Fortran源代码: .F;
  - Fortran 90源代码: .f90;
  - C++源代码: .c++, .C, .cpp, .cc, .cxx;
  - 汇编代码: .s, .S;
  - 目标文件: .o;
  - 库文件: .a;
  - 共享库: .so;

Unix中常用的编译系统

- 命令行选项:
  - -c: 只编译, 不链接, 即只生成.o文件.
  - -o *filename*: 指定输出文件名, 缺省为*.o, a.out等.
  - -I*path*: 指定(增加)包含文件(如*.h)的搜索目录.
  - -L*path*: 指定(增加)库文件的搜索目录.
  - -l*name*: 与库文件lib*name*.a(.so)链接.
  - 优化开关: -O, -O1, -O2, -O3, 等等.
  - 目标码中包含源文件名、行号等信息(用于程序调试): -g.
- 例:
  - `f77 -O2 -o prog file1.f file2.c file3.o file4.a`
  - `f77 -c file.f`
    `f77 -o out file.o`
  - `f77 -c -I/usr/local/mpi/include file.f`
    `f77 -o prog -L/usr/local/mpi/lib file.o -lmpi` (等价于:
    `f77 -o prog file.o /usr/local/mpi/lib/libmpi.a`)

程序调试

GDB

GDB是GNU开源组织发布的强大的UNIX下的程序调试工具。一般来说，GDB主要帮忙你完成下面四个方面的功能：

- 启动你的程序，可以按照你的自定义的要求随心所欲的运行程序。
- 可让被调试的程序在你所指定的调置的断点处停住。
- 当程序被停住时，可以检查此时你的程序中所发生的事。
- 动态的改变你程序的执行环境。

vlagrind

Valgrind是在linux系统下开发应用程序时用于调试内存问题的工具。

```
valgrind --tool=memcheck --leak-check=yes \
      --show-reachable=yes  --run-libc-freeres=yes  \
      ./yourprogram
```

Example

Example: crash.c

```
#include <stdio.h>
int main(int argc, char* argv[])
{
   int input =0;
   int *a;

   printf("Input an integer:");
   scanf("%d", input);
   printf("The integer you input is %d\n", input);
   a=(int*)malloc(input*sizeof(*a));

   return 0;
}
```

实用工具make

- 命令形式:

```
make [-f Makefile] [options] [target [target ...]]
```

其中-f选项给出定义规则的文件名(简称Makefile文件), 缺省使用当前目录下的Makefile或makefile文件. target指明要求生成的目标(在Makefile中定义), 当命令行中不给出target时make只生成Makefile中定义的第一个目标. 比较有用也较通用的命令行选项有下面一些:

-f文件名: 指定Makefile文件名.
-n: 只显示将要执行的命令而并不执行它们.
-p: 显示定义的全部规则及宏, 用于对Makefile的调试.

- 通过Makefile文件定义一组文件之间的依赖关系及处理命令, 方便程序开发过程中的编译与维护.
- 处理规则的建立以特定的文件扩展名及文件修改时间为基础. 缺省支持常用的程序扩展名: .c, .f, .F, .o, .a, .h, 等等. 用户可以通过.SUFFIXES:目标定义新的文件扩展名.

实用工具make

- 基本规则:

  目标: 依赖对象

  <tab> *处理命令*

  <tab> ... ...

  例:

```
prog: file1.f file2.f file3.o
    f77 -O2 -o prog file1.f file2.f file3.o
```

其含义为: 如果目标(prog)不存在, 或者任何一个依赖对象(file1.f, file2.f, file3.o) 比目标新, 则执行指定的命令.

实用工具make

- 宏定义:

```
SRC=file1.f file2.f file3.c
prog: $(SRC)|
    f77 -O2 -o prog $(SRC)
```

  环境变量可以在Makefile中作为宏使用, 如$(HOME).
- 常用预定义的宏:
  - $@: 代表目标名(上例中为prog)
  - $<: 第一个依赖对象名(上例中为file1.f)
  - $^: 全部依赖对象名(上例中为file1.f file2.f file3.c)
  - $?: 全部比目标新的依赖对象名
  - $*: 用在隐式规则中, 代表不含扩展名的依赖对象

实用工具make

- 隐式规则:

```
prog: file1.o file2.o file3.o
    f77 -O2 -o prog $?
.c.o:
    cc -O2 -c $*.c
.f.o:
    f77 -O2 -c $*.f
```

Makefile示例

```
CC      = mpicc
FFLAG   = -g
CFLAG   = -g

.f.o:
    $(MPIF77) $(FFLAGS) -c $*.f

.c.o:
    $(CC) $(CFLAGS) -c $*.c

sendrecv: sendrecv.c
        $(CC) $(CFLAG) -o $@ $<

clean:
        rm -f *~ *.o a.out
```

Git 示例

- 建立仓库

```
git init
```

- 向仓库导入项目文件

```
git add DIR NEWFILE
```

- 向仓库提交项目文件

```
git commit -m "My first backup"
```

- 在项目中删除文件或目录

```
git rm OLDFILE
```

- 显示最近提交列表，以及他们的SHA1哈希值

```
git log
```

- 恢复到指定提交状态

```
git reset --hard SHA1_HASH
```

或者

```
git checkout SHA1_HASH
```

练习作业

1. 编写矩阵相乘程序,并使用make, git实现程序的编译和维护。
2. 分别使用intel和GNU编译器以及不同的编译优化参数进行编译, 比较程序的性能差异。