共享内存编程介绍

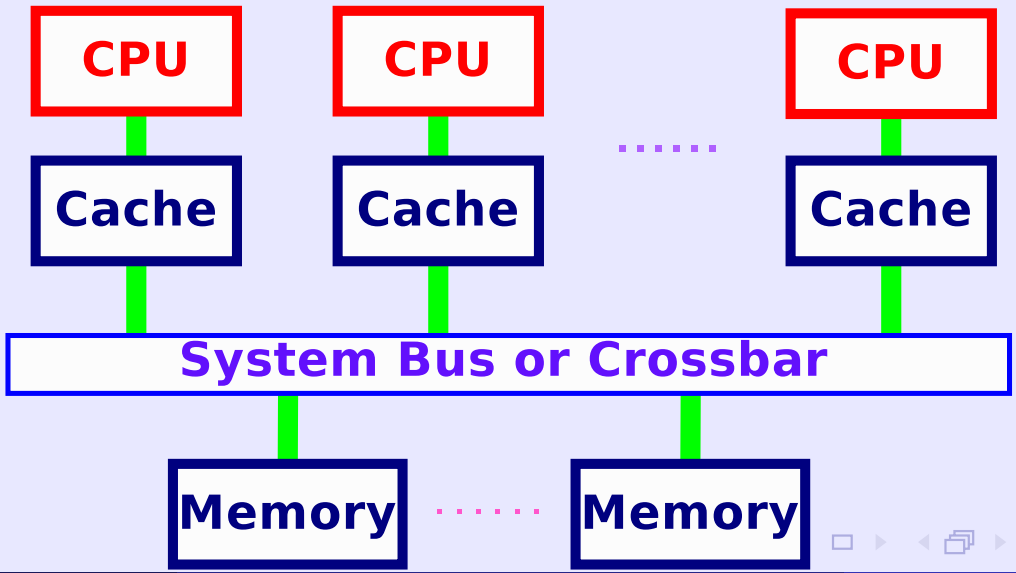
崔 涛

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

共享内存SMP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.



并行化方法 I

- 数据划分: 独立任务针对不同数据集。

```
for(i=0;i<100;i++)
    a[i] = b[i] + c[i];
```

- 功能划分：按功能分解为不同任务。

```
s1: m=a*b+3*b;
s2: s=(a+b)/2;
s3: v=m*s;
```

- 任务并行：不同功能、独立执行、无通信。

```
s1:a = funca(a1,a2);
s2:b = funcb(b1,b2);
```

并行化方法 II

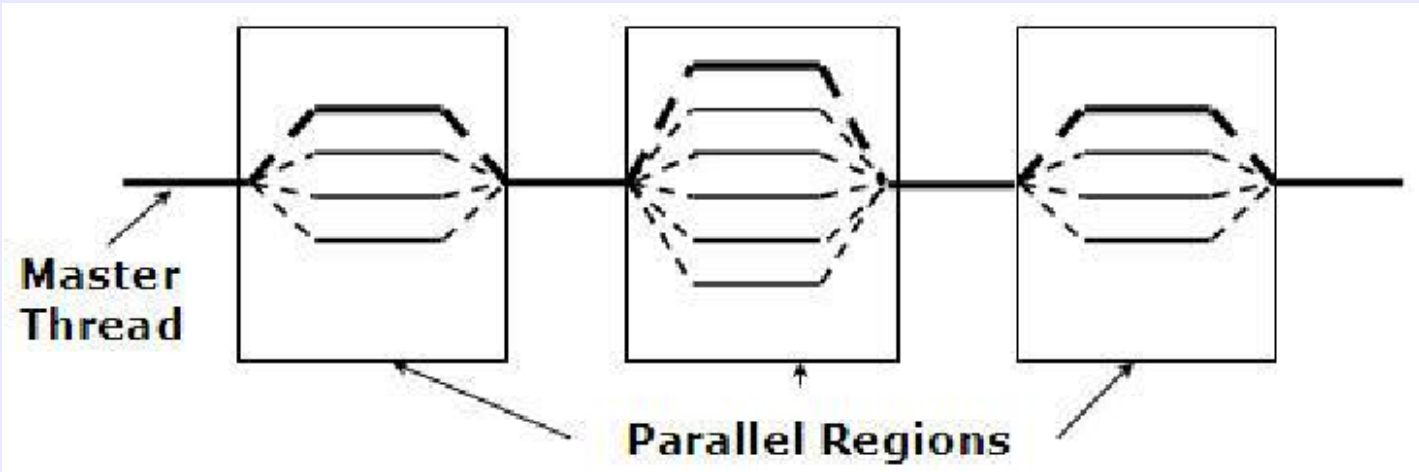
- 流水线并行

```
add eax,ebx->eax:     IF -> ID -> EX -> WB;
sub ecx,edx->ecx:           IF -> ID -> EX -> WB;
or  edi,esi->edi:                 IF -> ID -> EX -> WB;
```

取指(IF)、译码(ID)、执行(EX)、写回寄存器堆(WB)

并行编程模式

- 多进程：MPI、PVM等，主要适用于多机分布式环境
- 多线程: 主要用于单机环境, 主要采用Fork-Join 执行模式



  - 显式并行：WinThread/Pthread: example
  - 隐式并行：OpenMP

共享内存编程模式：OpenMP

OpenMP是什么？

- 多线程并行应用程序界面；
- 显式地指导编译器如何以及何时利用应用程序中的并行性。

基本构成：

OpenMP的功能由两种形式提供：编译指导语句与运行时库函数，并通过环境变量的方式灵活控制程序的运行。

第一个OpenMP程序

```
#include <stdio.h>
#include <omp.h>   /* 运行时库函数头文件*/
#define N 6

int main(int argc, char *argv[])
{
  int i;
  printf ("Hello World! Thread: %d\n",
          omp_get_thread_num());   /* 运行时库函数 */

  #pragma omp parallel for     /* 编译指导语句*/
    for (i = 0; i < N; ++i)                          | 并行 |
      print ("Hello World!  Thread: %d, i: %d\n",   | 执行 |
             omp_get_thread_num(), i);               | 代码 |
}
```

基本概念

- 串行执行区：只能有单个线程执行的代码
- 并行执行区：可有多个线程执行的代码
  - 循环
  - 独立的代码块
- 变量作用域：
  - 共享变量
  - 私有变量
- 数据相关
  - 流相关（flow dependence） `r1=r2+r0; r3=r1+2;`
  - 输出相关（output dependence） `r1=r0+a; r1=3+4*r0`
  - 反相关（anti-dependence）`a=b+c; b=3*r;`

编译指导

```
#pragma omp directive [clause, [clause], ...]
```

directive可以取：

- parallel：声明一个并行区。
- parallel for：声明将一个for循环并行化。
- parallel sections：声明将一段代码并行化。
- barrier：声明一个同步操作。
- master：声明仅有主线程执行的代码段。
- single：声明仅有一个线程执行的代码段。

编译指导支持的子句（I）

- `if (scalar expression)`：如果表达式成立，则多线程并行执行。
- `private (variable list)`：所有线程均拥有`list`所列变量的副本，该副本在并行区中没有初始化（不入串行区的值带入并行区），并行区结束后，副本的值也不带回串行区。
- `threadprivate(list)`：类似`private`，适用于全局变量。
- `shared (variable list)`：所有线程共享`list`所列的变量，这些变量将串行区中的值带入并行区。
- `default (none/shared)`：缺省情况下，串行区中声明的变量在并行区中均为共享变量；循环量为私有变量。子句default(none)取消一切缺省设置；子句default(shared)声明所有变量均为共享变量，包括循环变量。

编译指导支持的子句（II）

- `firstprivate (variable list)`：所有线程均拥有`list`所列变量的副本，并且用串行区的值初始化并行区中的变量副本（将串行区中变量的值带入并行区）。
- `lastprivate (variable list)`：所有线程均拥有`list`所列变量的副本，并且在并行区结束后，副本的值被写回串行区。
- `reduction (operator:variable)`
- `nowait`：取消隐含的同步。
- `num_threads(N)`：设置并行执行的线程数为`N`。

并行化for的注意事项

- 循环控制语句要规范，易于判断循环次数。
- 循环中不能包含允许循环提前推出的语句（将改变循环次数），如：`break`、`return`、`exit`。

循环调度与分块（负载平衡）

```
#pragma omp for schedule(kind[, chunk-size])
```

- static：为每个进程分配指定大小的循环块
- dynamic：动态地为空闲的进程分配指定大小的循环块
- guided：类似于dynamic，但循环块大小从大到小变化
- runtime：运行时，通过环境变量“OMP_SCHEDULE”决定采用上面哪种调度策略

串行计算区与并行计算区的数据交换

- `firstprivate(list)`: 将变量在主线程中的值复制到每个线程中的副本
- `lastprivate(list)`: 将变量在每个线程中的值复制到主线程中
- `copyin(list)`: 将主线程的`threadprivate`变量的值复制到执行并行区的每个线程的`threadprivate`变量中
- `copyprivate(list)`: 使用一个私有变量将某个值从一个成员线程广播到执行并行区的其他线程

保护共享变量的更新操作

- `#pragma omp critical`
- `#pragma omp automic`

常用库函数列表

函数	功能
omp_set_num_threads()	设置线程数
omp_get_num_threads()	获取线程数
omp_get_max_threads()	设置最大线程数
omp_get_thread_num()	获取当前线程的线程号
omp_get_num_procs()	获取最大处理器数
omp_in_parallel()	判断是否处于并行区
omp_set_dynamic()	激活线程数动态调整功能
omp_get_dynamic()	判断线程数是否可动态调整
omp_set_nested()	激活嵌套并行功能
omp_get_nested()	判断是否允许嵌套并行
omp_get_wtime()	获取强上时间
omp_get_wtick()	获取时钟精度

常用OpenMP环境变量

- OMP_SCHEDULE：控制for循环任务分配结构的调度
- OMP_NUM_THREADS：设置默认的线程个数
- OMP_DYNAMIC：真值表示允许动态调整线程数目
- OMP_NESTED：真值表示允许嵌套并行

程序实例

1. get_tid
2. reduction
3. parallel_for
4. condition_for
5. first_last_private
6. private
7. thread_private
8. section
9. schedule
10. critical

影响性能的主要因素 I

- 根据Amdahl定律，我们应当努力提高并行化代码在应用程序中的比率，这是通用的提高效率的方法。
- 编写OpenMP程序时需要考虑的程序优化的一些方面的问题：
  - OpenMP本身的开销——OpenMP多线程并行化需要一定的程序库的支持。在这些运行时库对程序并行加速的同时需要运行库的本身，因此，库中代码的运行必然会带来一定的开销。
  - 负载均衡——使用OpenMP进行并行程序编码要非常注意使得线程之间的负载大致均衡，能够让多个线程在大致相同的时间内完成工作，从而能够提高程序运行的效率。

影响性能的主要因素 II

- 局部性——在程序运行过程中，高速缓存将缓存最近刚刚访问过的数据以及这些数据相邻的数据。因此，在编写程序的时候，需要考虑到高速缓存的作用，有意地运用这种局部性带来的高速缓存的效率提高。
- 线程同步带来的开销——多个线程在进行同步的时候必然带来一定的同步开销，在使用多线程进行开发时需要考虑同步的必要性，消除不必要的同步，或者调整同步的顺序，就有可能带来性能上的提升。

练习作业

- 改造程序：矩阵乘法OpenMP并行
- 列举出下面循环中的流相关、输出相关、反相关

```
    for(i=1;i<10; i++){
s1:     a[i]=b[i]+a[i];
s2:     c[i+1]=a[i]+d[i];
s3:     a[i-1]=2*b[i];
s4:     b[i+1]=2*b[i];
    }
```